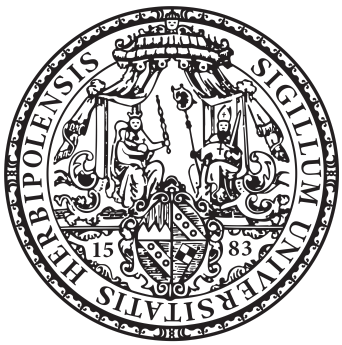
Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Machine Learning-Based Characterisation of RFI Signals in Radio Astronomical Measurement Data

Bachelorarbeit

Eingereicht von:	Lorenz, Erik
Studiengang:	Bachelor Luft- und Raumfahrtinformatik
Matrikelnummer:	2474335
Betreuer:	Univ.-Prof. Dr. Ingo Scholtes
	Univ.-Prof. Dr. Matthias Kadler
Bearbeitungszeit:	von 08.08.2024
	bis 31.10.2024



Julius-Maximilians-Universität Würzburg
Center for Artificial Intelligence and Data Science
Lehrstuhl für Informatik XV - Machine Learning for Complex Networks
Emil-Fischer-Strasse 50, 97074 Würzburg

Abstract

Radio-frequency interference (RFI) poses a persistent challenge to the quality of astronomical data captured by radio telescopes, with increased wireless technologies contributing significantly to this interference. This thesis presents the scientific context of RFI, focusing on the Effelsberg 100m telescope and its observations as part of the TELAMON project, which monitors active galactic nuclei (AGN) and high-energy astrophysical phenomena. The preprocessing phase outlines critical steps in extracting, calibrating, and denoising data from MBFITS files, including the creation of edge masks to enhance feature detection for anomaly identification. The anomaly detection section explores a variety of methods, both traditional and machine learning-based, such as signal-to-noise ratio, PCA, Isolation Forest, and One-Class SVM, each evaluated for efficacy in isolating anomalous signals within RFI-affected data. Clustering techniques, including K-means and DBSCAN, are applied to further classify anomalies detected in the light curves and edge masks, contributing a robust framework for distinguishing meaningful patterns. In conclusion, the thesis provides insights into both heuristic and machine learning-based methods for automating RFI detection, offering a foundation for improved data quality and efficiency in radio astronomical research.

Zusammenfassung

Radiofrequenzstörungen (RFI) stellen eine anhaltende Herausforderung für die Qualität astronomischer Daten dar, die von Radioteleskopen erfasst werden. Die zunehmende Nutzung drahtloser Technologien trägt dabei erheblich zu dieser Störung bei. Diese Arbeit beleuchtet den wissenschaftlichen Kontext von RFI und fokussiert sich auf das Effelsberg 100m-Teleskop und dessen Beobachtungen im Rahmen des TELAMON-Projekts, welches aktive galaktische Kerne (AGN) und hochenergetische astrophysikalische Phänomene überwacht. Die Vorverarbeitung beschreibt die wesentlichen Schritte zur Extraktion, Kalibrierung und Rauschreduktion von Daten aus MBFITS-Dateien, einschließlich der Erstellung von Kantenmasken zur Verbesserung der Merkmalsdetektion für die Anomalieerkennung. Die Anomalieerkennung untersucht verschiedene Ansätze, sowohl traditionelle als auch maschinelle Lernmethoden wie Signal-Rausch-Verhältnis, PCA, Isolation Forest und One-Class SVM, die hinsichtlich ihrer Wirksamkeit bei der Isolierung von Anomalien in RFI-gestörten Daten bewertet werden. Clustering-Techniken, einschließlich K-Means und DBSCAN, werden angewendet, um die in den Lichtkurven und Kantenmasken erkannten Anomalien weiter zu klassifizieren und somit ein robustes Framework zur Erkennung relevanter Muster zu schaffen. Abschließend bietet die Arbeit Einblicke in heuristische und auf maschinellem Lernen basierende Methoden zur Automatisierung der RFI-Erkennung und schafft so eine Grundlage zur Verbesserung der Datenqualität und Effizienz in der radioastronomischen Forschung.

Contents

1 Motivation

In the quest to understand the cosmos, radio telescopes play an essential role, capturing faint signals that reveal secrets of distant galaxies, pulsars, and black holes. However, the increasing prevalence of radio-frequency interference (RFI) presents a significant barrier to obtaining accurate measurements in radio astronomy. RFI, generated from everyday technologies such as mobile phones, Wi-Fi networks, and satellites, has grown to interfere with the delicate observations required in this field, often masking the faint cosmic signals researchers aim to study. This interference not only degrades data quality but also risks misinterpretation, potentially leading to incorrect scientific conclusions.

Traditional methods for identifying and mitigating RFI are predominantly manual or rule-based, which makes them both labour-intensive and less adaptable to evolving sources of interference. This thesis seeks to explore machine learning techniques for the automated detection and characterisation of RFI in radio astronomical data, offering a path to a more efficient and scalable solution. By leveraging machine learning, we can move beyond static, manual methods and towards a more dynamic, data-driven approach that can adapt to new and varied forms of interference. This advancement is not only vital for improving data accuracy but also for enabling radio astronomers to focus on interpreting cosmic phenomena, with minimal disturbance from human-made noise. In this way, the work aligns with the broader goal of preserving radio astronomy as an effective tool for exploration and discovery in an increasingly connected world.

2 Scientific Context

This chapter provides an overview of the scientific background related to radio-frequency interference (RFI) in radio astronomy, focusing on its implications for data quality and analysis. The TELAMON project, which uses the Effelsberg 100m radio telescope to monitor active galactic nuclei (AGN), serves as the primary source of observational data in this thesis. Key technical elements, including the structure of the MBFITS dataset, data acquisition processes, and visual representations like time-frequency plots, are discussed. Additionally, the chapter introduces the principles of edge detection, which are essential for identifying RFI within astronomical data. The goal of this thesis is to explore automated methods for detecting and characterising RFI, ultimately enhancing the reliability of radio astronomical observations.

2.1 TELAMON

The Tev Effelsberg Long-term Agn MONitoring (TELAMON) project, led by Kadler et al., uses the Effelsberg 100-meter radio telescope to observe the radio spectra of some of the most energetic sources in the universe: AGNs, with a special focus on TeV blazars and candidate neutrino-associated AGNs. These belong to the brightest sources in almost all regions of the electromagnetic spectrum. The project aims to characterise the radio variability of jets emitted by AGNs that operate in the TeV (teraelectronvolt) range. TELAMON also focuses on tracing the dynamical processes occurring within the parsec-scale jets of blazars associated with high-energy flares or neutrino detections. Both is a topic of significant interest in contemporary astrophysics, as it may help explain the origins of some of the most energetic particles in the universe.
Observations for the TELAMON project began in the fall of 2020 and have since been performed every 2-4 weeks. Up to now, the program has included 59 TeV blazars as well as around 200 candidate neutrino-associated AGN, for which radio observations were made using four different receivers: S7mm, S14mm, S20mm, and S45mm (Eppel et al., 2024).

2.2 The Effelsberg 100 m Telescope

The Effelsberg telescope is located near the Effelsberg mountain in the Eiffel region in Germany and is operated by the Max-Planck-Institut für Radioastronomie. Since its opening in 1972, it has played an important part in radio astronomy, being among the largest fully steerable radio dishes in the world. The telescope's 100-meter diameter allows it to capture extremely faint radio signals from across the universe, providing invaluable data on celestial objects such as pulsars, galaxies, and black holes.(Vogel,

2012).

Figure 1: The Effelsberg 100m Telescope is located in the Eifel region near Bonn. Taken from Ros et al. (2018).

2.3 The Dataset

The dataset from TELAMON used in this thesis is derived from radio observations captured by the S20mm receiver between September 2021 and November 2023.
The data is stored in the MBFITS file format, a variation of the standard Flexible Image Transport System (FITS), which is commonly used in astronomy for storing and sharing data. MBFITS is specifically designed for multibeam, multireceiver, and single-dish telescopes, and is employed at observatories like the IRAM 30m, Effelsberg 100m, and APEX. The MBFITS format follows a "scan-subscan-integration" structure. An integration represents the basic recorded data unit, consisting of a set of dumps (the smallest time interval of correlated data) with identical configurations. A subscan is a collection of integrations forming a pattern across a source, while a scan can be as simple as a single integration or as complex as a set of pointed subscans that map an extended source (Muders, 2007).
TELAMON uses cross-scans, where each scan consists of 8 subscans. In half of the subscans, the telescope moved across the source in the azimuth direction, and in the other half, it moved in the elevation direction. The S20mm receiver simultaneously captures data centred at 14 GHz and 17GHz, resulting in a total of 8 subscans per cross scan per frequency band. (Eppel et al., 2024). Overall, the dataset comprises 46,000 subscans, amounting to approximately 1.21 TB of data. Each subscan contains circular polarisation data, where the intensity of the signal is derived from Left Circular Polarisation (LCP) and Right Circular Polarisation (RCP). Typically, intensity values are averaged across subscans to reduce noise. However, in this case, this step was omitted to avoid inadvertently mitigating RFI signals as well.

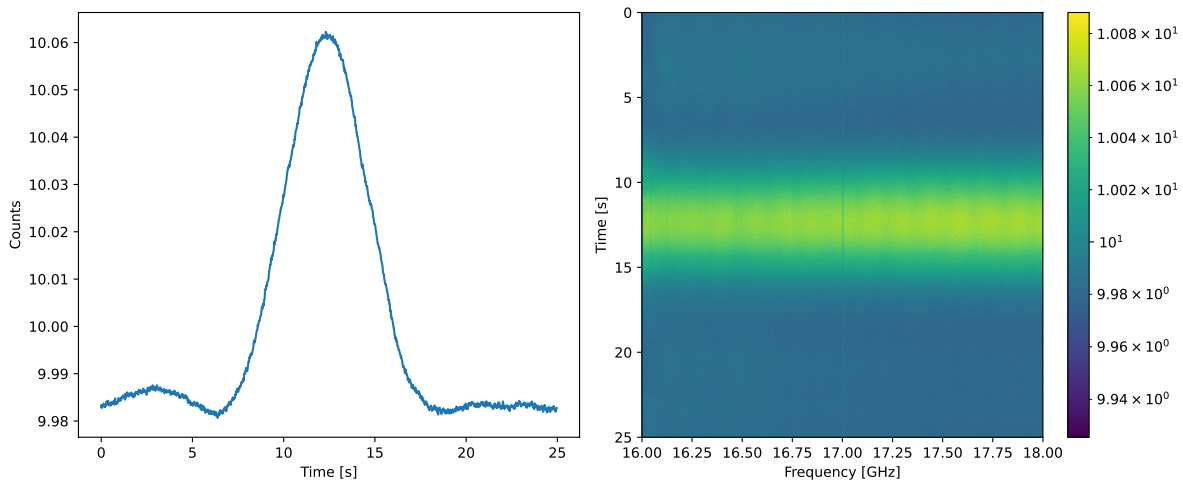


Figure 2: Time-frequency plot (right) and corresponding light curve (left), with the AGN source visible as a bell-shaped curve at the centre.

2.4 Important Plots

Each scan can be displayed as a time-frequency plot, with the intensity given by a colour map — brighter colours indicate higher intensity. The frequency axis extends over 2 GHz, divided into 1024 frequency bins and centred at either 14 or 17 GHz. The time axis has 0.015625-second resolution, which given the telescope scan duration of 25 seconds corresponds to 1600 time steps. From the time-frequency plot, two more plots can be derived:

The average intensity over all frequencies is called the light curve, which shows the average counts relative to time. Figure 2 shows the light curve next to the time-frequency plot. Additionally, an edge mask plot (see Figure 3) is used to highlight the edges within the time-frequency data, which may indicate RFI.

2.5 Edge Detection

In this data context, sudden and significant changes in brightness are not expected as the scan of the source should ideally produces a smooth, normal distribution. This means that sudden significant changes in brightness can be considered anomalous. Those sudden brightness changes are what is considered an edge.

Edged detection typically occurs in three stages: smoothing, differentiation and labelling. Smoothing is applied to reduce noise in the image as noise can create false edges, leading to inaccurate anomaly detection. Smoothing is often done through filtering with a Gaussian filter, which blurs the image slightly. Differentiation highlights areas where the intensity changes abruptly by measuring the rate of change in brightness. This is done using operators like the gradient or the Laplacian. Labelling refers to

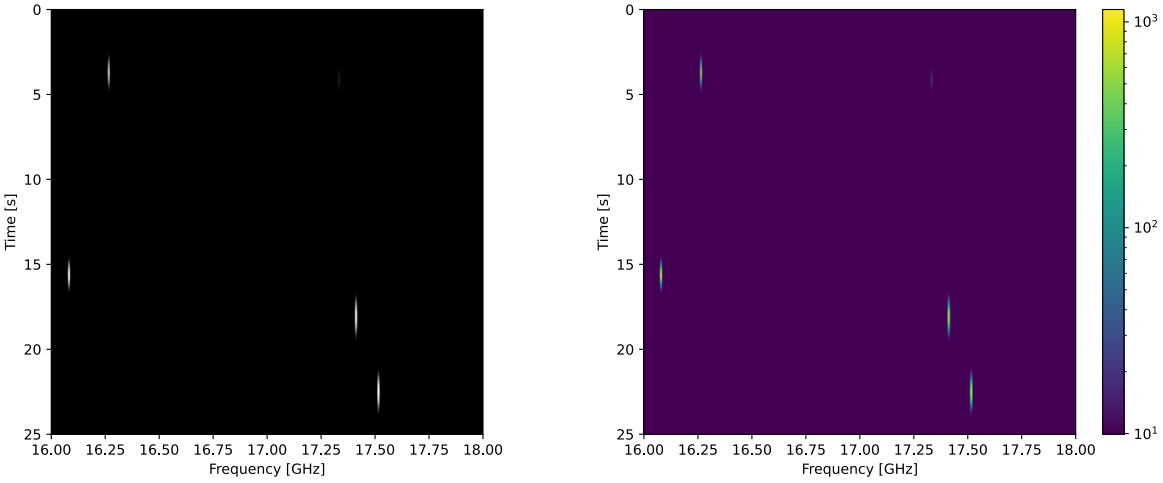


Figure 3: Time-frequency plot from Figure 2 overlaid with simulated RFI (1000 times brighter) on the right, and the corresponding edge mask on the left. Edges are shown in white, with the background in black.

identifying and marking the edges by applying a threshold to the differentiated output to classify pixels as edge or non-edge pixels. This helps to separate significant changes in brightness from minor variations.

There are multiple edge detection algorithms like Sobel or Canny edge detection, but for the following approaches, Laplacian edge detection was chosen. Reasons for this being the isotropic properties of the Laplacian, meaning that edges in all directions can be detected simultaneously. This makes the Laplacian simple and computationally not expensive since all computations are done in one step. It is also especially useful for the purpose of detecting small-scale anomalies as the Laplacian is effective in highlighting subtle changes and fine details.

The Laplacian operator is a second-order derivative. It identifies edges as zero-crossings, which are points where the second derivative of the image intensity changes sign (Ziou and Tabbone, 1998). The 2D Laplacian $L(x, y)$ of a 2D image with intensity values $I(x, y)$ is given as:

$$L(x, y) = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2} \tag{1}$$

This can be implemented using a discrete Laplacian kernel such as the 4-connectivity (standard Laplacian):

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \tag{2}$$

or the 8-connectivity:

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \tag{3}$$

During the convolution, the kernel will be applied to each pixel of an image. In the kernel, the central positive value corresponds to the centre pixel, while the surrounding values represent the surrounding pixels. The sum of the kernel values is zero, which means that the zero-sum property of flat regions, areas without edges, is preserved. The standard Laplacian considers only the values in x- and y-direction, not diagonal, as indicated by the zero values in the corners. This means that diagonal edges cannot be detected. The 8-connectivity kernel would be able to detect diagonal edges but by that would also introduce smoothing and produce a blurrier output. Since no distinct diagonal edges are expected (i.e. due to dispersion), using the simpler version suffices. A popular variation of Laplacian operator is the Laplacian of Gaussian (LoG) which combines the smoothing and differentiation into one operator. The Laplacian of Gaussian is defined as the convolution of an image with the second derivative of a Gaussian function:

$$LoG(x, y) = \nabla^2(G(x, y, \sigma)) \tag{4}$$

where:

- ∇^2 is the Laplacian operator:

$$\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \tag{5}$$

- $G(x, y, \sigma)$ is the Gaussian function:

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{6}$$

2.6 What is RFI

Radio astronomy relies on the detection of radio signals emitted by astronomical objects, which are inherently extremely weak by the time they reach Earth. These signals are many orders of magnitude weaker than those used in terrestrial communication systems. As a result, cosmic sources are very vulnerable to being masked or distorted by man-made RFI. This interference can degrade observational data or render it entirely useless, as illustrated in Figure 3, where the intended cosmic signal is no longer discernible. Even more concerning is the potential for interfering signals to be misinterpreted as genuine cosmic radio sources, leading to erroneous conclusions in scientific analysis.

Causes of Radio Frequency Interference

RFI occurs primarily when two signals are transmitted at the same or adjacent frequencies. This overlap can cause interference that disrupts the reception of the intended signal. Additionally, transmitters often unintentionally emit signals at harmonic frequencies—multiples of their intended operating frequency—which can extend interference into bands reserved for other purposes, including radio astronomy. These unintended emissions exacerbate the challenges faced by astronomers in isolating weak cosmic signals from the background of man-made noise.

Types of Radio Frequency Interference

RFI can be broadly categorized into broadband interference and narrowband interference. Each type has distinct characteristics and sources, which affect how they impact radio astronomical observations.
Broadband interference can result from both natural and man-made sources. Broadband interference causes noise that is spread across a wide range of frequencies with equal intensity. In this case, the entire bandwidth of the receiver experiences equal noise levels, rather than certain frequencies being more heavily affected.
Narrowband interference, on the other hand, is mostly man-made and confined to a narrow range of frequencies. It is particularly problematic because it affects specific frequencies within the receiver's bandwidth and may even imitate astronomical signals.

	Broadband Interference	**Narrowband Interference**
Continuous	Galactic noise, Solar noise, atmospheric noise, fluorescent lamps	Radio and television transmitters, WIFI signals, cellular towers
Intermittent	Broadband radar transmitters, modulators	Narrowband radar transmitters, radio beacons
Transient	Lightning strike, electronic discharge	Telemetry signals, remote controls

Table 1: Examples of RFI classified by their characteristics.

RFI at Effelsberg

To mitigate RFI, a radio quiet zone has been established around Effelsberg. Within this zone, communication on the ground and transmissions to and from satellites are largely prohibited (Bundesnetzagentur für Elektrizität, 2024). The restrictions vary by distance and application; for example, satellite phones are banned within a 30-kilometre radius, making Effelsberg the only place on Earth inaccessible via satellite phone (Westfalen-

Blatt, 2021). These measures aim to minimise local sources of interference.
Despite stringent regulations, several factors continue to contribute to RFI at Effelsberg. Signals from distant transmitters can reflect off the atmosphere or terrain, reaching the telescope despite the quiet zone.
Satellites also present a huge challenge. Not all satellites adhere to national regulations, and their transmissions can interfere with observations. Additionally, satellites that cover large areas, such as television satellites, are unable to exclude specific small regions like Effelsberg from their broadcast zones.
Not all radio signals are intentionally sent or man-made. Some internal and natural sources also contribute to RFI. Electronic equipment within the observatory itself can introduce interference, as can natural atmospheric phenomena.
Furthermore, even though specific frequency bands are designated for radio astronomy and are protected by the Bundesnetzagentur (Federal Network Agency) in Germany, transmitters operating in adjacent frequencies can still cause significant interference due to signal spillover Bundesnetzagentur für Elektrizität (2022).

2.7 Machine Learning

Machine Learning (ML) is a field of Artificial Intelligence (AI) that focuses on building systems capable of learning from data, identifying patterns and making predictions or decisions without explicit programming (Russell and Norvig, 2016). It allows algorithms to analyse data and autonomously adjust to new inputs, making automated decision-making processes possible. Machine learning techniques can be broadly divided into three categories:
supervised learning, unsupervised learning, and reinforcement learning, each of which addresses specific types of problems and use cases. In **supervised learning**, the algorithm is provided with a labelled dataset which contains both inputs and corresponding outputs. The goal of supervised learning is to learn a function that maps inputs to outputs accurately. Supervised learning can be further divided into:

- **Classification:** Predicting discrete-valued output, i.e. classifying emails as spam or identifying plant species based on their features.
- **Regression:** Predicting a continuous-valued output, i.e. predicting tomorrows temperature based on today's weather or valuing a house based on location, size, age, ...

Unlike supervised learning, **unsupervised learning** involves data that is not labelled. The algorithm is set to discover patterns and structures within the dataset without ex-

plicit guidance about the expected output. Common types of unsupervised learning include:

- **Clustering:** Grouping similar examples together based on shared features, i.e. discovering customers with similar buying habits or grouping documents with similar topics.
- **Dimensionality Reduction:** Reducing the number of variables in a dataset, while preserving as much information as possible. It is used for i.e. noise reduction, feature extraction and data compression.

In **reinforcement learning**, an agent interacts with its environment and learns by receiving feedback in the form of rewards or penalties based on its actions. Its primary goal is to maximize the cumulative reward over time. An example of reinforcement learning is teaching a robot how to navigate a room by rewarding it for avoiding obstacles and reaching its target (Russell and Norvig, 2016).

Since the data of the TELAMON project is unlabelled, and manual labelling would be tedious and time-consuming, unsupervised learning approaches were chosen to uncover underlaying patterns:

2.7.1 PCA

Principle Component Analysis (PCA) is a statistical technique that was first introduced by Pearson (1901). It is used to reduce dimensionality of large datasets, simplifying the data to uncover underlying patterns, while preserving as much information as possible. It does so by transforming the data into a new set of uncorrelated variables, known as principal components (PCs), which are linear combinations of the original features. PCA can be understood geometrically as finding the best-fitting lower-dimensional subspace that captures the most variance in the data. Applications for PCA can be found in feature extraction, data compression, data compression and noise reduction.

Despite its strengths, PCA is a linear technique and may not capture non-linear patterns in the data. A more detailed description of PCA in its modern-day usage can be found in Jolliffe and Cadima (2016).

2.7.2 t-SNE

t-SNE (t-distributed Stochastic Neighbour Embedding) is a non-linear dimensionality reduction technique for visualising high-dimensional data by representing each high-dimensional datapoint as a point in a two- or three-dimensional map. Developed by

Van der Maaten and Hinton (2008), t-SNE is particularly effective for preserving local relationships and is widely used to create interpretable visualizations that highlight clusters and patterns within complex datasets.
Unlike PCA, t-SNE excels at representing non-linear relationships in the data and making clusters more apparent in a low-dimensional space. However, t-SNE is not a clustering algorithm; it does not assign labels or define cluster boundaries. It is a purely exploratory tool, often used to visualize the results of clustering or to gain insights into the structure of the data before clustering is applied. Despite its advantages in visualization, PCA is often preferred for preprocessing prior to clustering due to its simplicity and speed, especially when working with large datasets. A combination of both can be used.

2.7.3 K-means

K-means is a clustering algorithm introduced by Macqueen (1967). It works by partitioning a dataset into k distinct, non-overlapping clusters by minimising the variance within each cluster. The algorithm begins by creating k clusters each containing a single random point. Each data is assigned to the cluster with the nearest mean after which the mean of the cluster is then recalculated to include the newly assigned points. This process continues until convergence, meaning that no points change clusters, or the variance within clusters is minimised. K-means is used for similarity grouping and nonlinear prediction, creating simplified models of complex data distributions.

One of the limitations of K-means is that the numbers of clusters, k, need to be specified beforehand. It may also struggle with identifying clusters of varying densities or sizes.

2.7.4 DBSCAN

DBSCAN (Density-Based Spatial Clustering of Applications with Noise), first proposed by Ester et al. (1996), is another clustering algorithm that groups points together based on their density. Unlike k-means, DBSCAN does not require the number of clusters to be predefined. Instead, it groups points based on their density which allows it to discover arbitrarily shaped clusters, making it more flexible than K-means.
The algorithm starts by selecting a random point and connecting it to all points within a defined neighbourhood. If a minimum number of points is within the neighbourhood, the point is considered a core point and a new cluster is formed. All reachable points are added to the cluster. If a point does not belong to any cluster, it is considered noise.

One of DBSCAN's main advantages is its ability to identify noise points points, which makes it suitable for datasets where noise or outliers need to be detected.

2.7.5 Autoencoders

Deep autoencoders, first introduced by Hinton and Salakhutdinov (2006), are a type of multi-layer neural network primarily used for unsupervised learning tasks. They consist of two main components: An encoder network, which compresses the input data into a low-dimensional representation (latent space), and a decoder network, which reconstructs the original data from the compressed representation (see Figure 4. Autoencoders train by minimising the discrepancy between original and reconstructed data. They are scalable to large datasets and particularly effective for detecting non-linear relationships in data. Typical use-cases for autoencoders include dimensionality reduction, feature extraction, noise reduction and image compression.

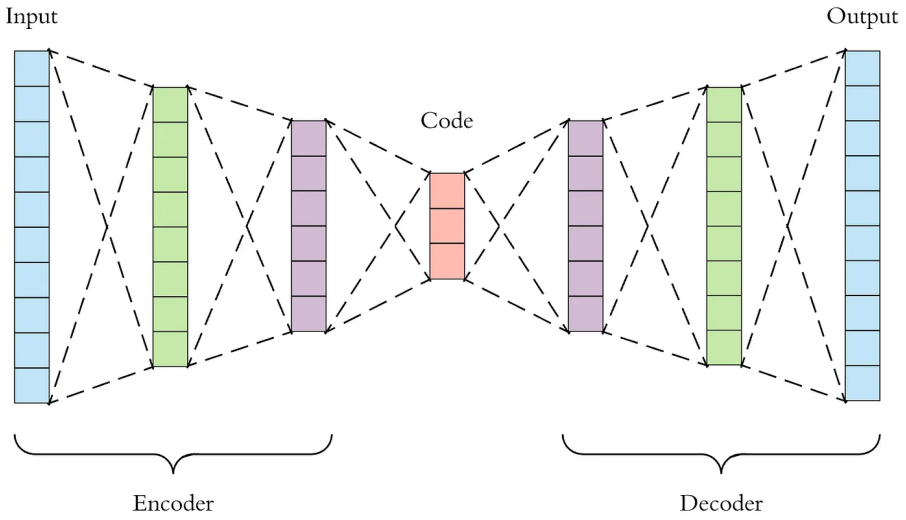


Figure 4: An autoencoder typically consists of multiple layers in both the encoder and decoder, arranged symmetrically. The central layer represents the compressed representation, also known as the latent space or code. Taken from `https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798`.

The standard autoencoder architecture is flexible, but it can be specialised for different tasks by integrating specialised layers, making them more suited to particular types of data. Two important variations are Recurrent Autoencoders (RAE) and Convolutional Autoencoders (CAE).

Recurrent Autoencoders (RAEs):
Recurrent Autoencoders combine the principle of autoencoders with Recurrent Neural Networks (RNNs). RNNs are a class of typically supervised neural networks, specifically designed for processing sequential data where the order of the elements is crucial. They don't treat input and output as independent but retain information from previous time steps, this way capturing dependencies and patterns within a sequence (aishwarya.27, 2024).
A key challenge with standard RNNs is their inability to learn relationships between

events separated by long intervals due to the vanishing gradient problem. In this scenario, error signals, essential for training the network, diminish exponentially as they propagate backwards through the network leading to poor performance.

The Long Short-Term Memory (LSTM) network solves this problem with a sophisticated internal structure that allows it to maintain a constant error flow, which enables error signals to persist over extended periods, ultimately empowering the network to learn connections between events separated by hundreds or even thousands of time steps. LSTM networks were first introduced by Hochreiter (1997).
LSTM Autoencoders incorporate LSTM layers into the encoder and decoder part of the autoencoder architecture, providing an unsupervised tool for learning representations in the domain of time series data, such as lightcurves, without the need for labels.

Convolutional Autoencoders (CAEs):
CAEs on the other hand combine the principles of autoencoders with Convolutional Neural Networks (CNNs), which are a class of typically supervised neural networks, specifically designed for processing data with a grid-like structure, such as images. CNNs treat each pixel in an image as an independent input. The fundamental building blocks of CNNs include:

- **Convolutional Layers:** applying filters tot he input to extract local patterns, producing a feature map.
- **Activation Layers:** Introducing non-linearity to the network, which is important for capturing complex features in the data.
- **Pooling Layers:** Reducing the spatial dimensions of the feature map to make the network computationally efficient and less susceptible to overfitting.

CNNs consist of sequences of these layers, where successive convolutional, activation, and pooling layers extract increasingly abstract and high-level features from the input (e.g., from edges in early layers to complex shapes in deeper layers).
A Convolutional Autoencoder incorporates the CNN layers into the encoder and decoder parts of the autoencoder architecture, allowing the model to learn compressed representations of data in an unsupervised manner. CAEs are a powerful tool for learning representations in the domain of images, or any other 2D data representation, like time-frequency data, without the need for labels (GeeksforGeeks, 2024).

2.7.6 Isolation Forest

Isolation Forest is a model-based anomaly detection method, first introduced by Liu et al. (2008). It focuses on isolating anomalies rather than profiling normal data points. Unlike other techniques that model the distribution of normal instances, Isolation Forest identifies anomalies by randomly partitioning the dataset using decision trees, referred to as isolation trees.

Each isolation tree is built by partitioning data points using randomly selected features and split values. The number of splits required to isolate a data point averaged over all trees in the forest corresponds to an anomaly score (see Figure 5). Data points that require fewer splits to isolate are considered anomalies.
Isolation Forest is efficient and effective in handling high-dimensional datasets and has become a popular choice for anomaly detection.

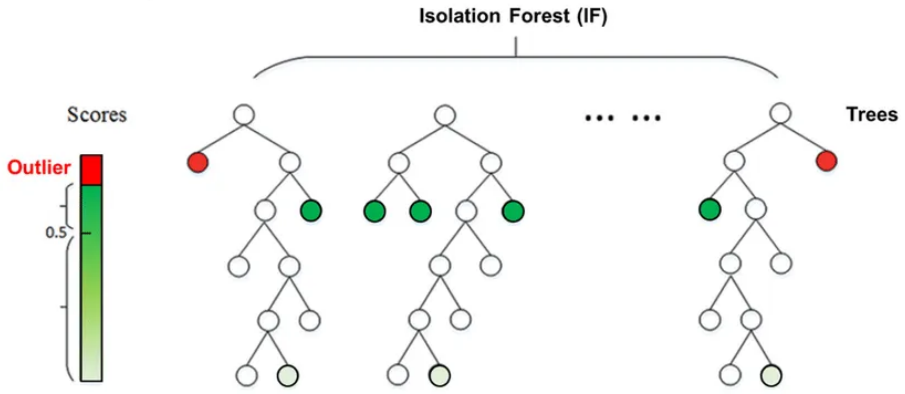


Figure 5: Several isolation trees comprising an isolation forest. The shorter the paths from the root node to the terminating node, the more likely a datapoint is considered an anomaly. Taken from `https://wiki.datrics.ai/isolation-forest-model`.

2.7.7 One-Class SVM

The One-Class Support Vector Machine (One-Class SVM) is a type of outlier detection method introduced by Schölkopf et al. (2001). It is based on the principle of Support Vector Machines (SVMs), which are traditionally used to separate data into classes using labelled data. This separation allows an SVM to classify new data points based on which class they align with.
The One-Class SVM is designed for cases where only one class is present in the data. The algorithm learns to encapsulate this majority class with a boundary function, effectively modelling the normal behaviour of the data. Any data points that fall outside this boundary are considered anomalies.

3 Data Preprocessing

This section outlines the preprocessing steps required to extract data from MBFITS files and turn them into a format which is useful to the anomaly detection algorithms. To show the key steps involved in this process, a subscan from November 2022 has been selected[1].

For clustering, additional preprocessing, such as dimensionality reduction or feature extraction, is necessary, which will be discussed in Section 5.

Keyword	Value	Description
FEBE	**S20mm-SPECPOL**	frontend-backend combination used
CHANNELS	**1024**	Number of spectral channels
NUSFEED	**4**	Number of feeds (receiving elements)
BANDWID	**2 GHz**	Bandwidth for the baseband
RESTFREQ	**17 GHz**	Rest frequency of the observation
TFIELDS	**2**	Number of columns (fields) in the data table
TTYPE1	**'MJD'**	First column contains MJD
TTYPE2	**'DATA'**	Second column contains the actual data
TDIM2	**'(1024,4)'**	Dimensions of each data entry
NAXIS2	**1764**	Total number of rows in the table

Table 2: Excerpt from the header of the ARRAYDATA-MBFITS table

3.1 Data Extraction and Intensity Calculation

MBFITS files usually consist of multiple tables. The MBFITS files in the TELAMON dataset contain two tables, the Primary Header, which contains metadata but no actual data, and the ARRAYDATA-MBFITS table which holds the observational data. The header of the ARRAYDATA-MBFITS provides some information that helps understand the data (see Table 2).
Some additional information are not given in the header but are known:

- **Scantime = 25 s:** The total scan time for an observation
- **$\triangle$t = 0.015625 s/row:** The temporal resolution

The four feeds correspond to polarisation parameters: lcp (left circular polarisation), rcp (right circular polarisation), cos, and sin.

[1] 2022-11-26, scan number 1369, subscan 3, baseband 2.

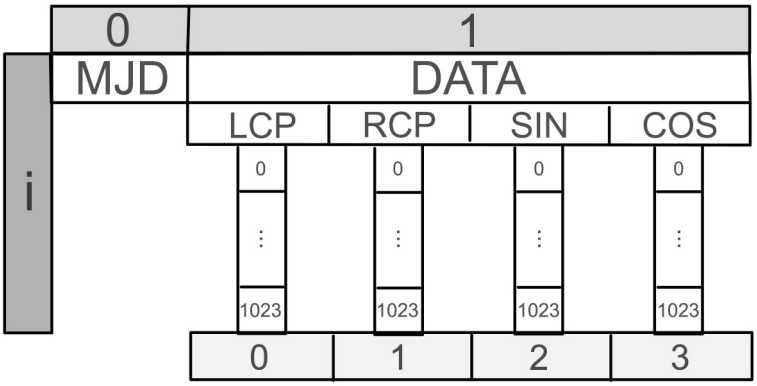


Figure 6: Structure of the data array: each row entry consists of 2 columns of which the second has 4 sub-columns with each 1024 entries.

With this information, spectra can be extracted using the `get_stokes_i` function (Listing 1). As described in the header, the table consists of 2 columns: The Modified Julian Date (MJD) and the actual data. It is also described that each data entry is an array of shape (1024,4), representing the frequency channels and feeds (see Figure 6). For calculating the intensity (Stokes I parameter), only the lcp and rcp feeds are necessary. The intensity is calculated by taking the element-wise sum of lcp and rcp. Given the 1764 rows, 1764 distinct time steps, and a temporal resolution of 0.015625 s/row, the total scantime is approximately 27.56 s. However, since the actual scan time is fixed at 25 seconds, the extra time results from the telescope recording before movement begins—this extra duration varies with each subscan. To align the data accurately, only the last 25 seconds (corresponding to 1,600 rows) are utilised. This results in the spectra as can be seen in Figure 7.

3.2 Bandpass Calibration

At this stage, the source is not yet visible in the data. To make it visible, a simple bandpass calibration is performed. The function `calibrate_bandpass` (Listing 2) effectively normalises each frequency bin by dividing each time step by a correction factor, which is the average intensity of that specific frequency bin across all time steps. The spectra after bandpass calibration can be seen in Figure 8.

3.3 Noise Removal and Calibration

After calibrating the bandpass, some distinct horizontal bands of varying intensity appear across the time axis. Those are the result of the telescope receiver switching between different observational phases. Those artefacts can easily be removed using the `remove_noise_cal` function (Listing 3): First, the spectra is separated into al-

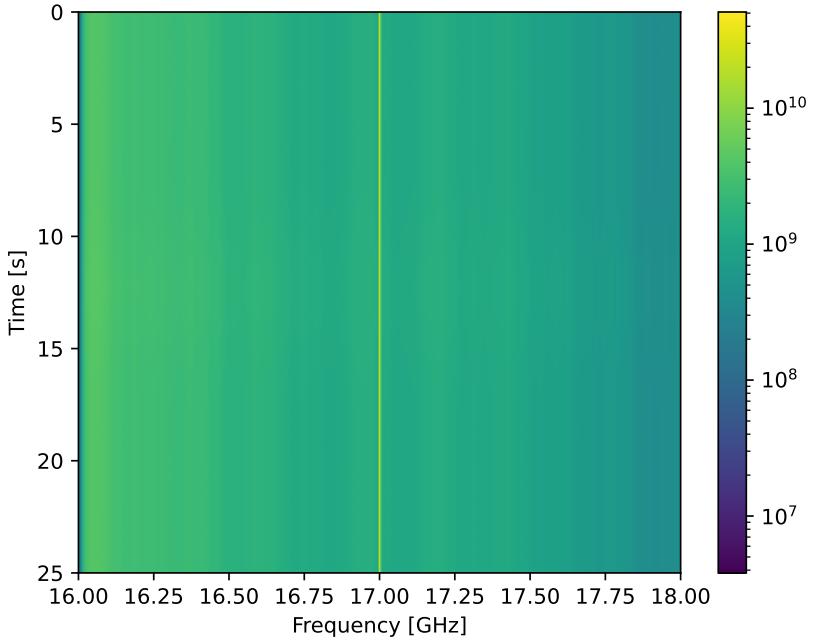


Figure 7: Time-frequency plot of the extracted spectra without any preprocessing. No source can be seen, the centre frequency is obscured by a bright line and some vertical bands are visible.

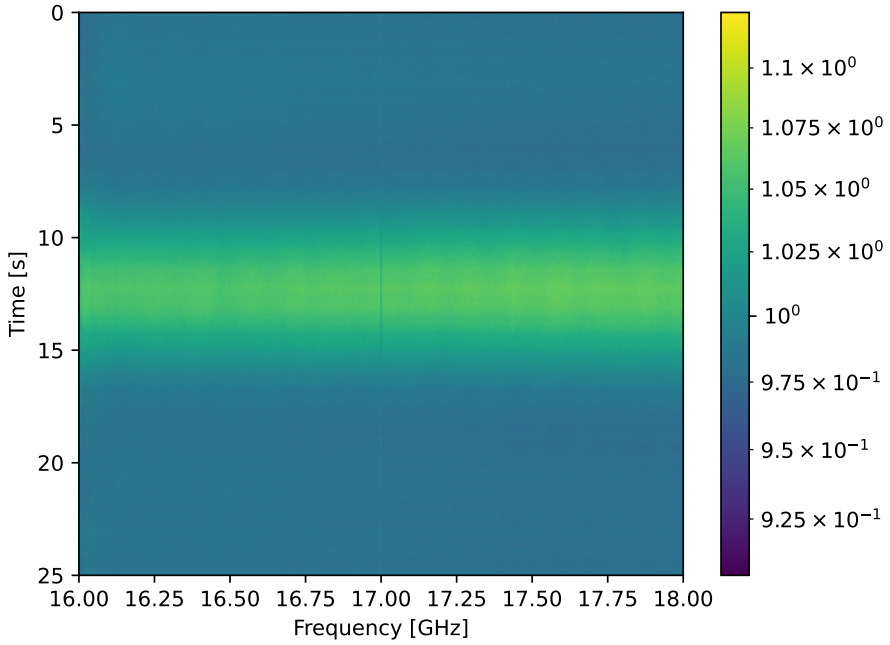


Figure 8: Time-frequency plot of the spectra after bandpass calibration. The source is clearly visible and the vertical line at centre frequency is less visible, but some horizontal bands can now be seen.

ternating phases. Then, each phase is normalised by subtracting its average intensity before reconstructing the spectra by combining the data in its original order. The final processed spectra (see Figure 9), shows significant improvements:

- The source is visible
- Horizontal banding is eliminated
- The vertical line at the centre frequency is substantially reduced

Overall, the plot appears more uniform and smooth, making it suitable for further preprocessing.

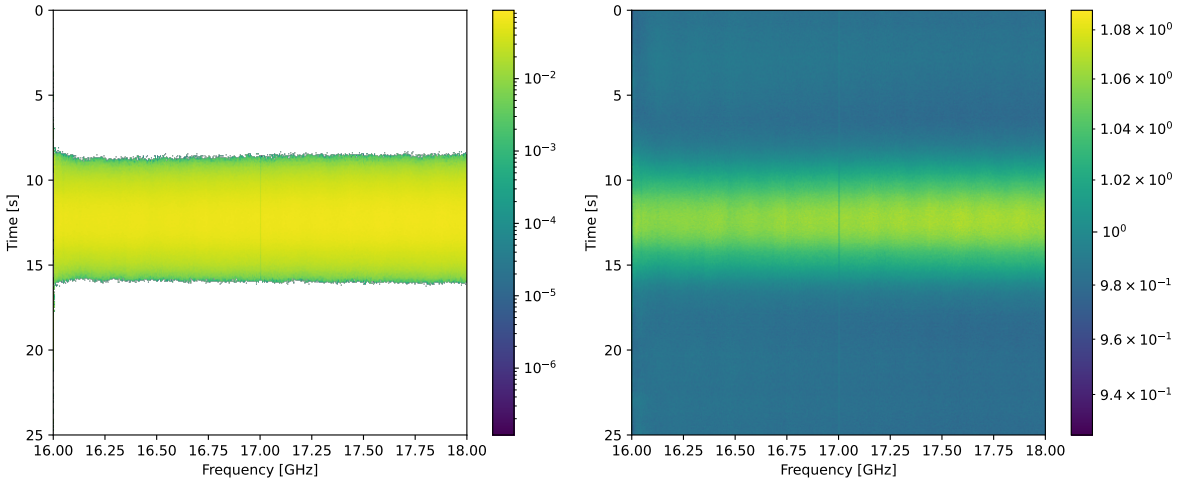


Figure 9: Time-frequency plot of the spectra after noise removal and calibration. The horizontal lines are no longer visible and the data overall looks smoother. The plot on the left shows the preprocessed spectra. The plot on the right enhances the visibility of the shape of the source by adding a value of 1 to each data point.

3.4 Source Removal

For anomaly detection or classification, the astronomical source is not relevant and it is essential to remove it from the data. If left unprocessed, the source's presence could even skew the results, especially when using an SNR-based approach for anomaly detection, as the source might appear stronger than the actual anomalies.
With the telescope moving steadily across the source, and the source being a point source, a Gaussian curve is produced. This curve can be fitted and after that removed using the `scipy.optimize.curve_fit` function. The most accurate method involves fitting and subtracting the Gaussian curve from each frequency bin separately, however this method is computationally intensive and impractical for large datasets. To save time, 2 approaches are possible:

1. Fitting a curve to the average of all frequency bins and then subtracting the fit from each frequency bin separately.
2. Fitting a curve to the lightcurve and subtracting it.

Since the average of all frequency bins is the lightcurve (as explained in Chapter 2.4), approach 1 and 2 essentially are the same. If approach 1 is done and the lightcurve is calculated afterwards, the resulting lightcurve will be the same as if approach 2 was done directly. This means that if for further processing only the lightcurve is needed, approach 2 will be done to save even more processing time.
The function `remove_source` handles both methods depending on the input (see Listing 4). The result can be seen in Figure 10.

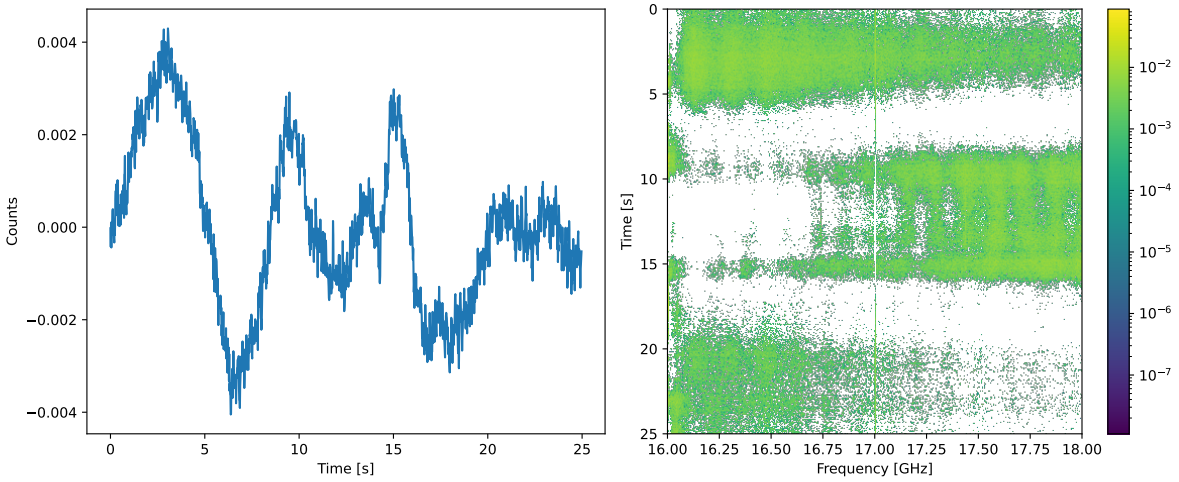


Figure 10: Lightcurve (left) and time-frequency plot (right) with the source removed.

With the source removed, the data is now ready for the SNR-based anomaly detection approach as well as the machine learning pipeline for the lightcurve approach.

3.5 Edge Mask Creation

For the approaches that are based on the edge mask, some additional preprocessing is necessary. The function `get_edge_mask` (Listing 5) is created to highlight anomalies and minimising the impact of noise. An approximation of the Laplacian of Gaussian (LoG) is applied to the data which combines the Laplacian operator with Gaussian smoothing (see Chapter 2.5).
Laplacian convolution is a weighted sum of the input data points resulting in high positive or negative values for regions of sharp transition. This high range of values can be seen in Figure 11. However, the anomalies themselves are still barely visible, since the rest of the data is still present. For this, a threshold is introduced, which is being defined as the sum of the mean of the filtered data and twice its standard deviation.

$$\text{Threshold} = \text{Mean} + 2 \times \text{Standard Deviation}, \tag{7}$$

Any absolute value in the convolved data that exceeds this threshold is considered significant and is marked as an edge by setting its value `True`. The result is a boolean

edge mask similar to the one in Figure 3, that includes only the detected edges, which most of the time will be some sort of anomaly.
Most of the time the source does not need to be removed, as it does not contain any sharp transitions that would be relevant to the Laplacian convolution.

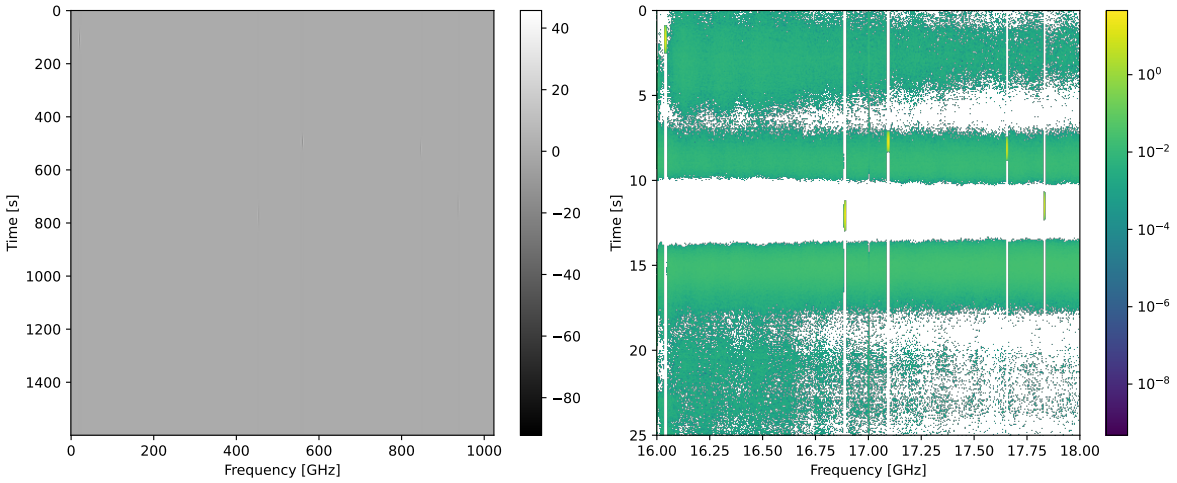


Figure 11: Time-frequency plot with simulated RFI (right) and the convolved data (left). The anomalies are barely visible.

4 Anomaly Detection

Anomaly detection is a discipline within data science and machine learning and describes the process of identifying unusual patterns or data points that deviate significantly from the norm within a dataset. RFI is exactly that, by showing up in an overall flat lightcurve or edge mask as some kind of spike or edge (as can be seen in Figures 2 and 12).
Several approaches — machine learning and non-machine learning — are being tested, some of which not only detect anomalies but also already classify them based on some simple features as a by-product.

How to determine what counts as an anomaly

Each method described in this chapter provides a metric for each subscan, indicating the likelihood of containing anomalies. Subscans can then be ranked based on this metric, with, depending on the method chosen, higher or lower values signifying a greater probability of anomalous features. However, determining a threshold for classifying a subscan as anomalous is challenging, as sometimes noise or an imperfectly removed source might generate a high value as well. Selecting a threshold, such as the 90th percentile, can be impractical, as the number of anomalous subscans is unknown. Since the data is unlabelled, approaches like receiver operating characteristic curve (ROC), which plots true positive rate (TPR) against the false positive rate (FPR) may be overly complex.
For most of the approaches, a heuristic approach like the elbow method is more suitable. The elbow method works by sorting the metric from highest to lowest, plotting the result, and identifying the point where the curve sharply transitions, which can be chosen as a threshold. Other approaches may require manual determination of the threshold, relying on informed estimation.

4.1 Lightcurve Approaches

The following approaches mostly use the data preprocessed as described in Chapters 3.1–3.4 in the lightcurve format. However, for the *SNR Lightcurve Approach*, the preprocessing step that limits the light curve to the last 1600 data points was not applied. At the time of calculating these results, it was not yet known that each scan is exactly 25 seconds long and any earlier datapoints needed to be omitted. Despite this, the effect on the results is likely minimal.

4.1.1 SNR Lightcurve Approach

The signal-to-noise ratio (SNR) is a measurement of a signals strength relative to the background noise. Noise can be defined as any unwanted signal that interferes with an information-bearing signal (Vaseghi, 2008). Here, an anomaly is considered an information-bearing signal, while everything that is not an anomaly will be considered noise. The SNR has several definitions based on its purpose to a specific application. Here, it is defined as:

$$\text{SNR} = \frac{\text{Signal Peak} - \text{Mean of Noise}}{\text{Standard Deviation of Noise}} \tag{8}$$

The basic principle of this approach is that a high SNR of the lightcurve indicates the presence of an anomaly, since in the ideal case, no signal in the lightcurve should be present after the source has been removed.

Boxcar Filter

To enhance the SNR and make fainter signals more visible, a boxcar filter is used:
A boxcar filter, also known as moving average filter, is a simple filter that replaces a group of consecutive data points with its average, effectively acting as a software-based low-pass filter (Zurich Instruments, 2021). Rapid fluctuations (noise) are filtered out leading to the data getting smoothed and signals becoming more visible.

The optimal boxcar window width w is selected from the set

$$w \in \{0.1s, 0.2s, 0.3s, \ldots, 0.9s, 1s, 2s, 3s, \ldots, 9s\}$$

and determined as follows:

1. A boxcar filter of width w_i is applied to **smooth** the data
2. The **highest** peak in the lightcurve is determined.
3. A window with the boxcar width is applied to the peak. The **signal window** extends $w_i/2$ to the left and right of the peak.
4. Everything outside the signal window is considered **noise**.
5. Only the peak of the data points inside the signal window is considered the **signal**. No averaging of the data inside the signal window is done.
6. **Calculating noise statistics**:
    - The **mean** of noise region serves as the baseline level of the signal.

- The **standard deviation** of the noise region quantifies the variability of the noise around its mean, representing its intensity.

7. The **true signal amplitude** is determined by subtracting the baseline level from the signal peak.
8. Using equation 8, the **SNR** for the specific width w_i is determined.

Steps 1 to 8 are repeated for all given boxcar widths. This is done using the functions `get_boxcar_filtered` (Listing 6), which applies the boxcar filter to the lightcurve, and `get_snr` (Listing 7), which then loops through the boxcar widths and determines the highest SNR. If the upper or lower bound of a signal window are out of bounds (index smaller then 0 or bigger then 1600), an SNR of 0 is assigned.

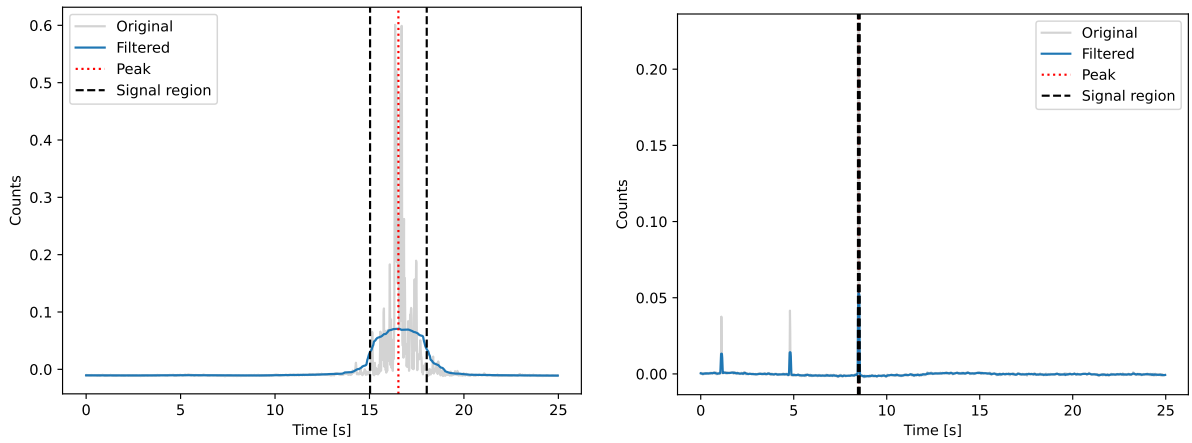
The boxcar filter not only helps in smoothing out the data and making signals more visible, but also defines a signals true width. The following scenarios can be considered:

- **true signal width < boxcar width:** Too many data points of too low amplitude would be included in the averaging step, thus reducing the signals amplitude and lowering the SNR.
- **true signal width > boxcar width:** The signal would not be entirely encompassed in the boxcar window, leaving too many data points of higher amplitude assigned to the noise region, which creates a higher noise intensity, lowering the SNR, but only slightly.
- **true signal width = boxcar width:** The filter window fully encompasses the signal which maximises the signal amplitude while minimising the noise and creating the highest possible SNR.

Because of this, the highest SNR will be achieved when the boxcar window matches the signals true width. A visualisation of this can be seen on 2 examples[2]in Figure 12.

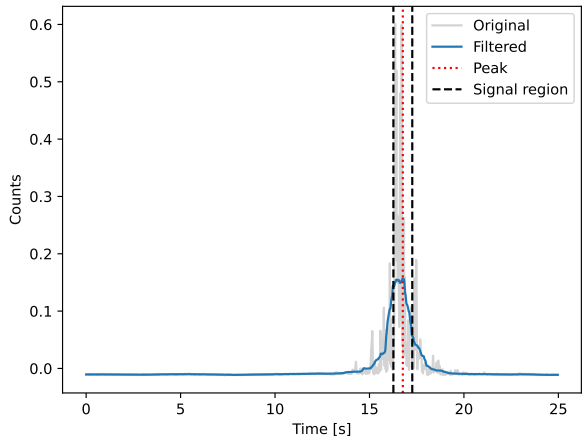
It has to be noted that this method of finding anomalies does not consider the existence of more than one anomalous signal inside the lightcurve, which may lead to a lower SNR. Determining the signal width using the boxcar filter most of the time will still work, but a wide boxcar window might encompass 2 or more signal peaks leading to an inaccurate result. Because of that, it is generally better to repeat the process twice: The first time with smaller boxcar widths (all widths $<$ 1s), and the second time with

[2]a),c),e): 2023-02-21, scan number 0533, subscan 8, baseband 1.
b), d), f): 2022-05-18, scan number 4572, subscan 6, baseband 2.

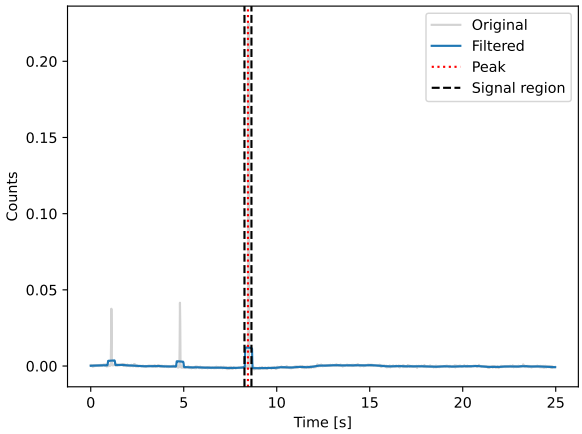


(a) SNR = 13.20, window size = 3.0 s

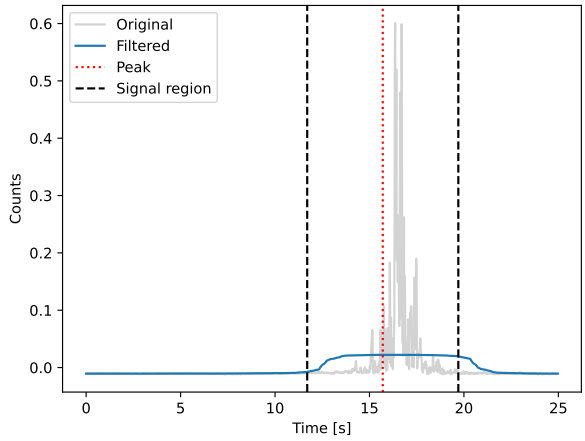
(b) SNR = 42.14, window size = 0.1 s



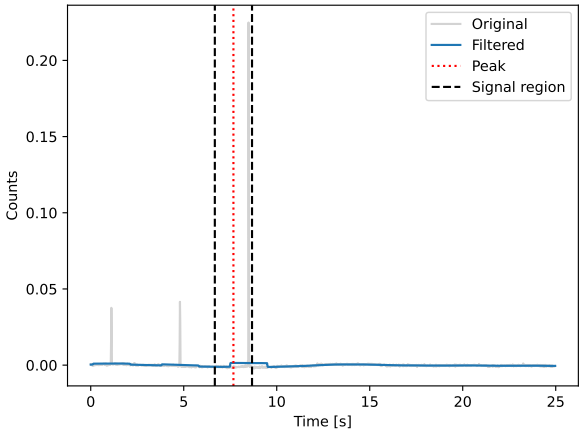
(c) SNR = 9.68, window size = 1.0s



(d) SNR = 13.41, window size = 0.4 s



(e) SNR = 5.23, window size = 8.0 s



(f) SNR = 2.55, window size = 2.0 s

Figure 12: The effects of varying boxcar widths are illustrated on two lightcurves. Plots (a) and (b) depict the optimal boxcar widths that achieve the highest SNR. Plots (e) and (f) show that an excessively wide boxcar width overly smooths the signal, leading to a significantly lower SNR. Even in plot (d), where the boxcar width is only slightly wider than ideal, there is a notable drop in SNR. Similarly, plot (c) exhibits a decreased SNR due to the boxcar window being too narrow.

bigger boxcar widths (all widths $\geq$ 1s). This way it is ensured that no mismatching is taking place, because a wide boxcar window encompassing more than one signal peak produces a higher SNR than a smaller boxcar window accurately fitting the true signal width.

Results

As described above, 2 iterations were done with a different subset of boxcar widths. Figure 13 displays the calculated SNR from the two iterations. The median SNR for both iterations is approximately 2.8, indicating that the peak in most curves is 2.8 times higher than the noise. This does not necessarily rule out the presence of a signal in some of those curves, it is more likely due to some remains from the source or random fluctuations. Notable are also the thousands of lightcurves having an SNR of exactly 0. This is not due those curves being completely flat but because of the signal's boundaries exceeding the curve, suggesting the peak signal is near or exactly at the edge of the curve. Naturally, the frequency of this occurring is higher for the wider widths.

Nevertheless, a significant number of lightcurves show a high or very high SNR with a maximum SNR of 44.7 for the wide boxcars and 107.1 for the small boxcars. Boxcar widths shorter than 1 second appear to not only produce higher but also more high SNR values, indicating that more signals can be found in the sub-second range. A threshold can be found by inspecting the curves in Figure 13 where the curve transitions from a sharp decline to a plateau, indicating the "elbow" point. This point occurs around the 600th index for the boxcar widths of 1 second or longer, corresponding to an SNR threshold of around 5.78, and at around the 700th index for widths shorter than 1 second, corresponding to an SNR threshold of 5. Manual inspection confirms these thresholds.

Using these thresholds classifies 1197 lightcurves as anomalous. Of these, 524 anomalies were only detected by the wider boxcars, while 531 were uniquely detected by the small boxcars. This means, only 124 anomalies were detected by both, emphasizing the importance of separating the analysis into small and wide boxcar widths.

4.1.2 PCA

The PCA (Principal Component Analysis) approach offers a method to reduce the dimensionality of data. It compresses the original data into a more compact representation but can also reconstruct this data from the compressed form. By calculating the error between the original and reconstructed data, PCA can reveal deviations from the norm, where light curves that differ significantly from common patterns tend to have higher reconstruction errors, marking them as potential anomalies.

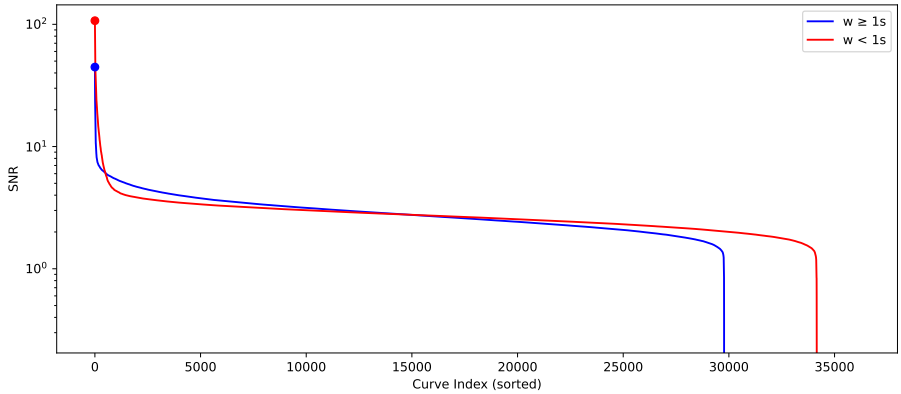


Figure 13: SNR from the two iterations sorted from highest to lowest, showing a high peak at the beginning, and an SNR of 0 for the last few thousand lightcurves.

The pipeline for this approach is as follows:

1. **Data Scaling**: Using scikit-learns `StandardScaler`, the data is first scaled to have a mean of 0 and a standard deviation of 1. This standardisation ensures that PCA, which is sensitive to scale, captures the shape of anomalies rather than variations in signal strength.

2. **Fitting PCA:** PCA is then applied to the scaled lightcurves to calculate the principle components (PCs). Since $N_{pc} = min(n_{samples}, n_{features})$ and the dataset contains 35000 lightcurves with each 1600 features, the maximum number of possible PCs is 1600. 1600 PCs would capture all the data's variance. However, to achieve a more compressed representation, only a subset of components is selected.

3. **Selecting Principle Components:** The goal is to retain as much variance as possible with the fewest components. According to Mikulski (2019), the aim is to capture 95% of the data's variance, which can be assessed by plotting the cumulative explained variance. For this dataset, the number of optimal PCs turns out to be 40. Reviewing the eigenvalues in a scree plot (where eigenvalues are plotted against PC numbers) shows an "elbow" point where variance stabilises, confirming this number (see Figures 14 and 15) (MANGALE).

4. **Data Transformation and Reconstruction:** The data is transformed using these 40 selected components and then reconstructed. The Mean Squared Error (MSE) between the original and reconstructed data is calculated for each light curve.

5. **Identifying Anomalies:** Lightcurves are sorted by descending reconstruction error, with the highest-error instances appearing anomalous, indicates around 50 anomalous lightcurves.

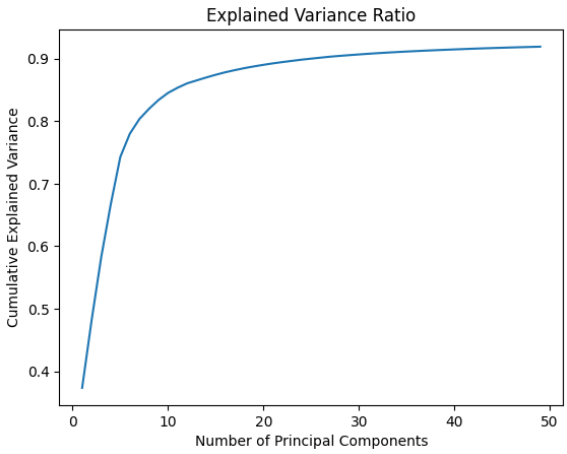


Figure 14: Explained Variance shows "elbow" at PC 20.

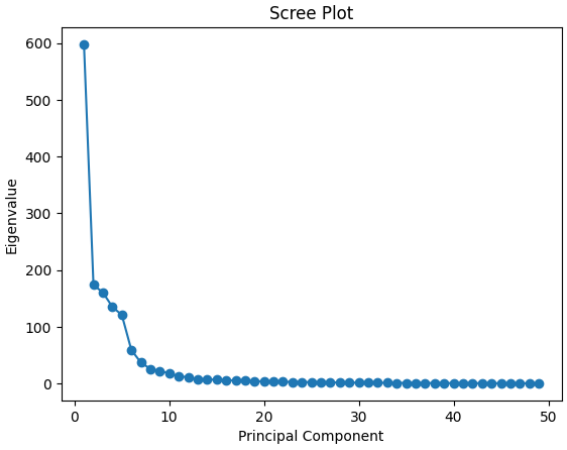


Figure 15: Scree Plot shows eigenvalues against PCs.

This is the textbook approach to using PCA. However, transforming the data using only the first principle component instead of 40 improves results, identifying around 400 lightcurves.
This improvement occurs, because the first PC captures the most significant variance — the most dominant pattern across all lightcurves — while subsequent PCs represent finer details and possibly noise. Deviations from the primary pattern are therefore more apparent using just the first component, which makes anomalies stand out more clearly.

4.1.3 Isolation Forest

To use Isolation Forest for anomaly detection,the data is again standardised to ensure consistency across data features. Several parameters influence the performance of Isolation Forest:

- **contamination** allows the algorithm to determine an appropriate threshold for anomaly detection without explicit guidance on anomaly density. Since the proportion of anomalies is unknown, this is set to the default value (automatic).
- **n_estimator** defines the number of trees used in the forest. A higher number of trees can improve the detection performance by creating more partitions but also improves the computation time. 200 trees were found to create the best result.

Isolation Forest assigns each lightcurve an anomaly score, where a lower score indicates a higher likelihood of being an anomaly. Sorting them accordingly, the lowest-scoring 550 lightcurves were identified as anomalous.

4.1.4 One-Class SVM

For this approach as well, the data has to be standardised, so that lightcurves are not only grouped based on their intensity. Some important parameters can be set when working with One-Class SVMs:

- The default **kernel** used is the Gaussian (RBF) kernel, which maps data into a higher-dimensional space. This makes it particularly suited for non-linear data, allowing it to detect anomalies within complex datasets.
- **gamma** controls the spread of the kernel and is set to 'auto', which calculates gamma as the inverse of the number of features.
- **nu (ν)** acts as an upper limit on the fraction of anomalies the model can classify. Looking at the previous combined result suggest at least 2000 lightcurves to be anomalous. To allow for previously unseen anomalies, ν was first set to 0.1, establishing an upper bound of 3,500 potential anomalies out of around 35,000 light curves.

The One-Class SVM also returns an anomaly score for each data point, with lower scores indicating a greater deviation from the norm. After sorting the data by these scores, around 400 anomalies can be identified.

4.2 Edge Mask Approaches

Ideally, edge masks without any kind of anomaly contain no `True` values, which means sorting out all edge masks that contain no `True` values leaves only the anomalous sub-scans. However, in reality more often than not, noise gets picked up as edges. Also the characteristic line at centre frequency described in Chapter 3 as well as the low-intensity lines created by the bandpass calibration at the frequency band of a signal (as can be seen in Figure 11 on the right) distort the outcome of simply counting the anomalous data points. A possible solution for that would be more smoothing or higher thresholding but this would inadvertently lead to more information loss. Instead, different mathematical approaches can be used to filter out edge masks containing noise or even detect true anomalies within the noise. For all the following approaches, the mask will be utilized by overlaying it onto the original data, so that only the important data points will be included in the calculations.

4.2.1 Peak Anomaly Intensity (PAI)

A simple approach that selects the highest anomaly as a measurement, ignoring low-intensity data points that are likely to be noise.

4.2.2 Signal-to-Noise Ratio (SNR)

Similar to the lightcurve-SNR approach, SNR can be used to emphasize even meaningful signals within random noise:

$$\text{SNR} = \frac{\text{PAI}}{\text{Standard Deviation}} \tag{9}$$

4.2.3 Max-to-Mean Ratio (MMR)

The MMR is similar to the SNR but instead of measuring how well the peak stands out from the noise variability, it focuses on how much the peak signal exceeds the average signal level:

$$\text{MMR} = \frac{\text{PAI}}{\text{Mean}} \tag{10}$$

4.2.4 Anomaly Significance Score (ASS)

Noise typically is characterised by being spread out and evenly distributed. The ASS utilises this characteristic of noise by measuring the density of edges and then multiplying it with the PAI, effectively punishing spaced out edges (noise) while rewarding clustered edges (signal) and highlighting strong anomalies:

$$\text{ASS} = \text{PAI} \times \frac{\text{Total Number of Edges}}{\text{Effective Size}} \tag{11}$$

with the effective size being the product of rows that contain edges and columns that contain edges.

Results

The metrics used to calculate anomalousness in the edge mask method are not directly comparable because they operate on different scales. To make them comparable, the scales were normalised (see Figure 16).

The plot shows similar characteristics to those seen in the SNR approach with initial high values, flattening out quickly. Clear similarities can be seen in the curves of the ASS and PAI approach, with both curves flattening at around index 1000, followed by a sharp drop at index 2000, before levelling off again. This might indicate a strong influence of the PAI on the ASS, as the PAI is a component of the ASS calculation (see Equation 11), potentially rendering it redundant. Further analysis reveals that of the

2000 first subscans in the lists, only about 300 appear exclusively in either list.
Also the SNR and MMR approach display similar curve patterns, with the only difference being higher initial values for the MMR. This occurs because the MMR, which represents the peak relative to the mean, is sensitive to extreme signals. Looking at the PAI curve again, it appears that the "bulge" in the MMR curve roughly aligns with the section of the PAI curve preceding a potential elbow.
Using the elbow method, thresholds have been determined at index 500 for the PAI, 1700 for the SNR, 2300 for the MMR and 700 for the ASS. In total, 2705 unique subscans were classified as anomalous of which none were uniquely found in the PAI subset, indicating that peak intensity alone is insufficient for classification.
A manual review of the ASS suggests that it is effective in isolating noise-free anomalous subscans, making it extremely useful if data with clear anomalies without noise is required.
Only about 300 unique subscans were classified as anomalous by the SNR approach compared to the MMR which classified almost 1000 that were not identified by other approaches using the edge mask.

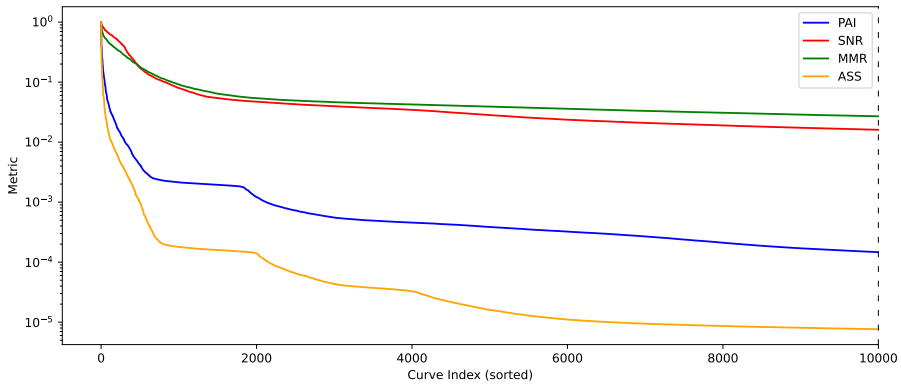


Figure 16: Different metrics used for the edge mask approach, arranged in descending order. The plot is cut off at index 10000 to highlight the most relevant portion. A total of around 35000 edge masks were analysed.

4.3 Combined Results

In total, non-machine learning methods identified 3457 anomalous subscans. Figure 17 displays the unique anomalies, meaning anomalies that no other non-machine learning method identified, in a stacked bar plot. The plot clearly highlights effectiveness of the edge mask MMR and the wide boxcar SNR approaches. These two methods found the largest number of unique anomalies, whereas all anomalies detected by the PAI method overlapped with findings from at least one other method. The pie chart in Figure 18 shows the overall anomaly counts detected by each approach, indicating that the Edge Mask SNR and MMR methods were the most effective, together captur-

ing 2,562 anomalies. This leaves only about 900 anomalies detected uniquely by the remaining methods. All edge mask approaches combined more than twice as much anomalous subscans than the lightcurve SNR method.

The machine learning approaches, One-Class SVM, isolation Fores and PCA, detected only a combined 650 unique anomalies.

Given anomaly scores below 0 are considered outliers by the models, the anomaly score distributions for both One-Class SVM and Isolation Forest (see Figure 19) suggested a far higher anomaly count, potentially between 2000 and 3000 each, with a few extreme outliers, as can be seen by the steep slope for the first few indices. PCA results point to a similar estimate (see Figure 20. Despite this, further manual inspection of the light curves beyond those initially identified by manual review confirmed no additional anomalies.
This, however, does not mean that the machine learning models did not detect outliers, but these outliers most of the time do not align with the anomalies of primary interest. These light curves likely differ subtly from the “average” curve, showing characteristics such as unusual noise levels or distinct shapes (e.g., gradual gradients in light intensity). While such variations may technically be anomalies, they may arise from technical issues, such as telescope misalignment, rather than being genuinely interesting or significant anomalies, like spikes or the like as can be seen in Figures 28 and **??**. Those kinds of anomalies may be too subtle for machine learning models to register as significant in lightcurves.
In conclusion, while the machine learning algorithms do identify anomalous data, they tend to capture different types of variations that are not the kinds of anomalies sought in this analysis.

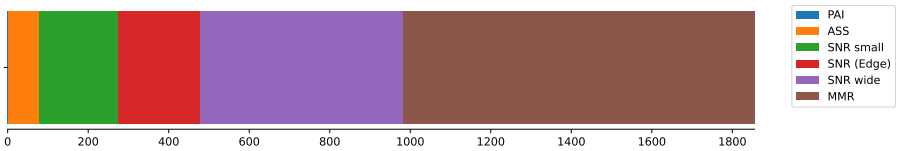


Figure 17: Stacked bar chart showing the number of anomalies found by each non-machine learning method that no other method detected.

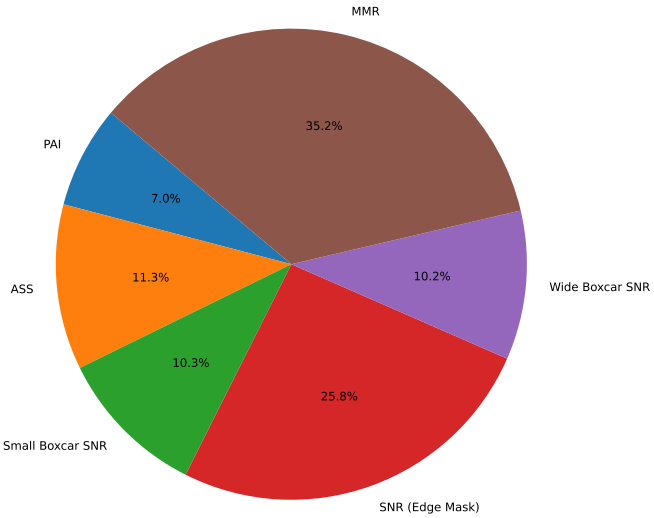


Figure 18: Pie chart showing the absolute number of anomalies each non-machine learning method detected.

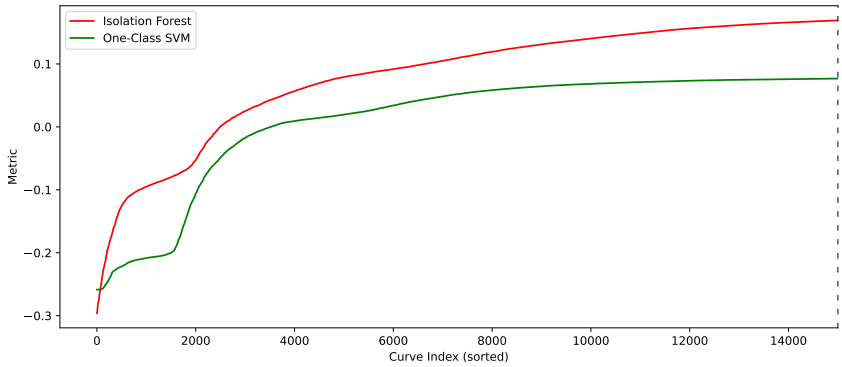


Figure 19: Anomaly scores of One-Class SVM and isolation Forest sorted.

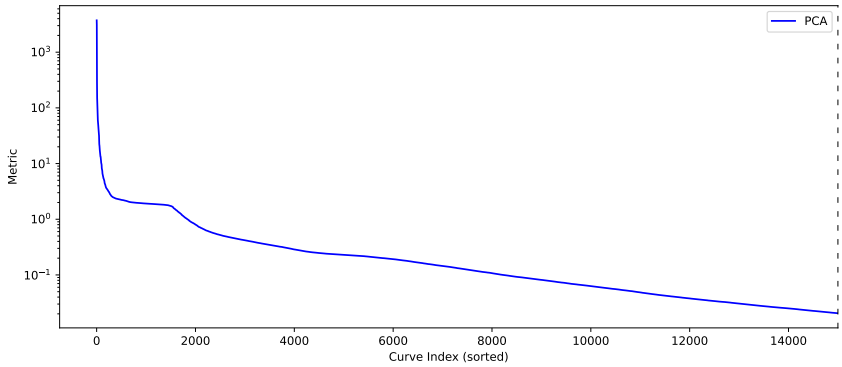


Figure 20: Reconstruction error sorted.

5 Clustering

Clustering is an unsupervised Machine Learning technique used to group unlabelled examples based on their similarity to one another. An example can be seen in Figure 30 which shows a hypothetical patient study undergoing a new treatment. In the study, patients were asked about the frequency and severity of symptoms they experienced. By analysing their responses, the patients can then be clustered into groups with similar treatment symptoms. However, in the unlabelled data, clusters were already clearly visible.
In the example, each data point has two features: the average number of symptoms per week and the average severity. With only two features, visualising and measuring the similarity between data points is relatively simple (Google). However, in the TELAMON dataset, each data point represents either a light curve or an edge mask, significantly increasing the feature complexity. Each light curve has a length of 1024, resulting in 1024 features per data point, whereas an edge mask has a shape of 1024x1600, resulting in 1,638,400 features per data point.

Due to the high dimensionality, preprocessing becomes essential. Techniques like dimensionality reduction using PCA or feature extraction using autoencoders can be employed to make clustering feasible.

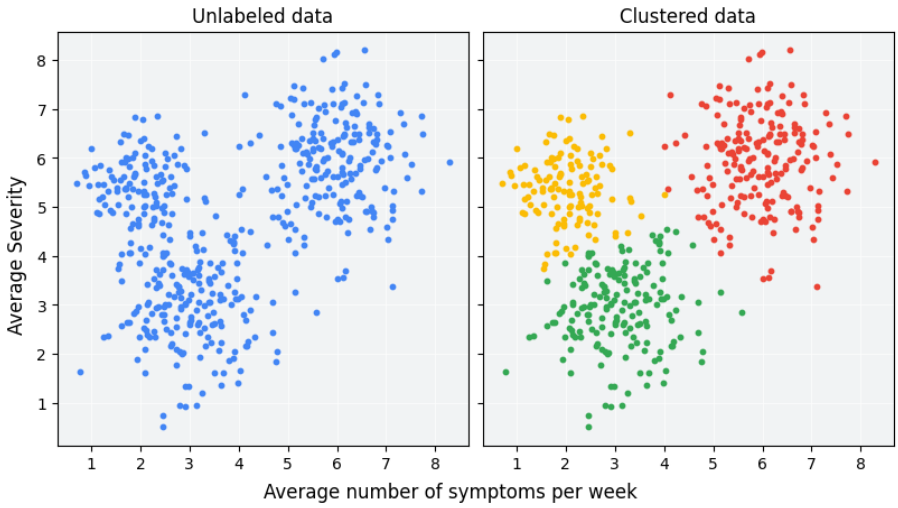


Figure 21: Simulated example data grouped into three clusters. Taken from `https://developers.google.com/machine-learning/clustering/overview`.

5.1 Lightcurve Approach

In the lightcurve approach, two separate methods for preparing the data for effective clustering were tested out: Principle Component Analysis(PCA), which reduces the dimensionality of the data to its principle components, and an LSTM Autoencoder,

which is a deep neural network, specified for extracting relevant features from the data. Both techniques were applied to the preprocessed data described in Chapters 3.1–3.4.

5.1.1 PCA

The PCA transformation was done similarly to the process described in Chapter 4.1.2. This time however, the variance of the data needs to be represented as good as possible. Choosing 40 principle components retains 95% of the variance which enhances the models ability to represent even subtle variations in the data. This approach not only helps with clustering but also enables effective visualization in a two-dimensional plot, making it easier to identify distinct patterns and outliers within the dataset.

5.1.2 LSTM Autoencoder

Data Standardisation and Preparation
Before training the LSTM Autoencoder, the data underwent standardization to ensure a mean of zero and a standard deviation of one. The scaling is important for making the model sensitive to the shape of anomalies rather than their scale. After standardization, the data was reshaped to fit the input structure required by the autoencoder and converted into PyTorch tensors for processing within the model.

Model Architecture
The LSTM Autoencoder is designed with an encoder-decoder structure, both utilizing LSTM networks to handle sequential data effectively:
Encoder: The encoder processes each input sequence through an LSTM network to produce a compressed latent representation. This step is vital for identifying and encapsulating the essential features of the data in a lower-dimensional form. The main parameters of the encoder include:

- The **input_size** is set to 1, as each sequence is univariate with one feature per time step.
- A **hidden_size** of 256 is chosen. It determines the number of features in the hidden state. A higher hidden size allows the LSTM to capture more complex patterns but increases computational load.
- **num_layers** determines the number of stacked LSTM layers, each progressively learning higher-level features of the data. Setting it to 8 balances the model's ability to capture abstract representations with computational efficiency.
- The **dropout_rate** is set to 50%, meaning each neuron's activation has a 50% chance of being "dropped out" or set to zero. This helps prevent overfitting and

forces the model to learn robust features.

Decoder: The decoder mirrors the encoder's structure and attempts to reconstruct the original input sequence from the latent representation. This reconstruction focuses the model on capturing patterns and features necessary to recreate the data accurately. The decoder's main parameters are aligned with the encoder to ensure symmetry:

- **input_size** matches the encoder's input size to maintain structural consistency.
- The **hidden_size** is also the same as the encoder's to ensure that no information is lost or distorted due to size mismatch.
- The decoder also matches the encoder's **num_layers** to maintain an aligned feature structure.
- **output_size** is set to 1, matching the input size so that each original feature at each time step is reconstructed, allowing for subsequent loss calculation.

Training Configuration and Loss Function

The model is trained using the Mean Squared Error (MSE) loss function, calculated between the reconstructed output and the original input sequence. The model learns by minimising this reconstruction error. The Adam (Adaptive Moment Estimation) optimiser is utilised to update the model parameters, with a learning rate of 0.001 determining the step size during weight updates.

To prevent overfitting — which occurs when the model learns patterns specific to the training data and fails to generalize to new data — early stopping was implemented. If the early stopping logic sees no significant loss improvement within the last 5 epochs, the training is stopped. The reconstruction loss was tracked using TensorBoard throughout the training process, providing clear visualizations of the model's convergence over epochs.

Latent Space Extraction

After training, the encoder's latent representations are extracted for each input sequence. These latent vectors serve as core features for clustering, encapsulating the essential patterns and anomalies within the light curves. By using these representations, the clustering process becomes more effective in identifying distinct groups and outliers in the dataset.

Training Results

The training process was successful and the loss function convolved with some fluctu-

ations at the end which led to the early stopping logic being triggered after 15 epochs. The model can be used for clustering attempts.

5.1.3 PCA + t-SNE + DBSCAN/K-Means

To investigate the underlying structures and patterns within the PCA-reduced dataset of light curves, clustering algorithms are applied to group similar data points.

Dimensionality Reduction with t-SNE

Before clustering, t-SNE is utilised to project the still high-dimensional PCA-reduced data into a two-dimensional space for visualisation purposes. openTSNE was chosen as it effectively processes large datasets in batches, which is crucial given the size of the data. The t-SNE plot (Figure 22) reveals some distinct clusters of points. Notable is the one large dense region accompanied by some smaller ones. This distribution suggests that while the majority of the light curves share similar features—forming the large cluster—there are also groups of light curves with unique characteristics, represented by the smaller clusters. These smaller clusters could potentially indicate anomalies or rare events within the dataset.

To assign labels to the clusters, DBSCAN and k-Means clustering are used separately.

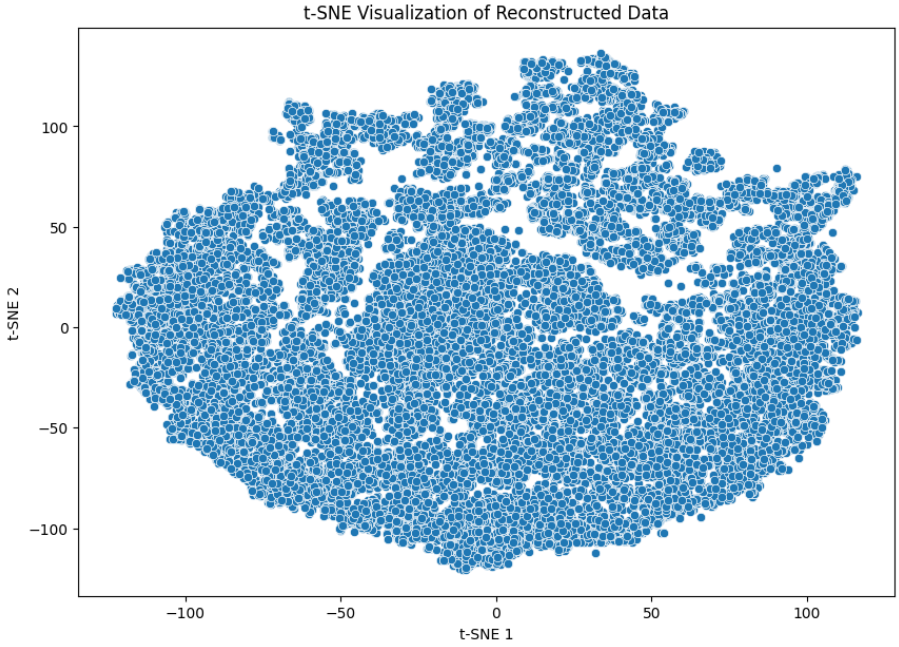


Figure 22: T-SNE applied to PCA-reduced data.

DBSCAN Clustering

Applying DBSCAN to the t-SNE data yields the following observations (see Figure 23):

- **Core Cluster:** The majority of the data points are clustered into a single large

core cluster (labelled as 0 and depicted in dark purple). This aligns with expectations, as most light curves are similar in nature.

- **Sparse Clusters:** A few additional sparse clusters are identified, scattered around the primary cluster. These clusters contain significantly fewer points, suggesting they represent distinct subgroups or potential anomalies within the dataset.
- **Noise Points:** DBSCAN assigns a label of -1 to noise points that do not fit into any cluster. In the results, there are hardly any noise points, indicating that nearly all data points were assigned to a cluster.

K-Means Clustering:
Unlike DBSCAN, K-Means requires the number of clusters to be specified beforehand. The optimal number of clusters is determined by fitting K-Means models across a range of cluster counts and analysing the results to find the "elbow point" in the distortion score plot. The elbow point is identified at 10 clusters, suggesting that this number balances model complexity with explained variance.
Key findings from the K-Means clustering (see Figure 24) include:

- **Primary Cluster:** Similar to DBSCAN, the majority of data points fall into a single large cluster (depicted in dark purple). This consistency reinforces the presence of a dominant group of similar light curves.
- **Small Clusters:** Several isolated small clusters are scattered around the primary cluster. These clusters have a very limited number of data points, indicating the presence of distinct subgroups or anomalies.
- **Cluster Shape and Size:** K-Means tends to create clusters that are similar in shape and size due to its reliance on minimizing variance within clusters. This characteristic is evident in the relatively compact and well-separated small clusters identified.

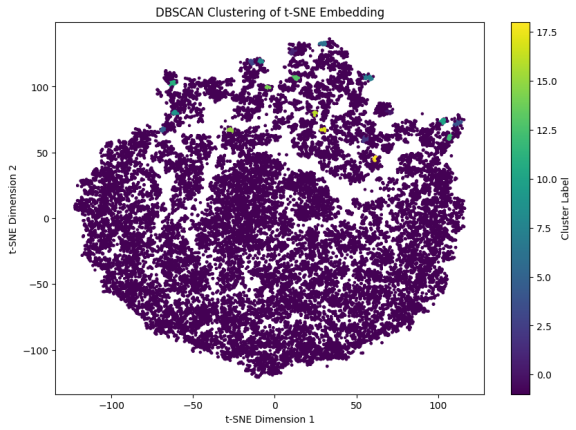


Figure 23: DBSCAN showing one major cluster and several small ones.

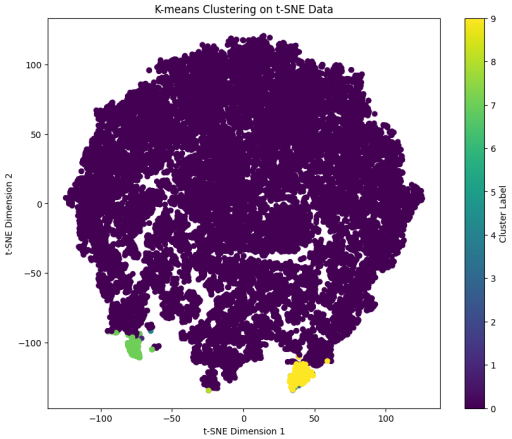


Figure 24: K-Means showing also one major cluster as well as 2 distinct smaller ones.

Evaluation of Clustering Results:

DBSCAN: When inspecting the lightcurves from different DBSCAN clusters, no apparent grouping based on the shape or other features of the lightcurve is visible. The clusters do not reveal significant differences, and the assignment of data points appear somewhat random. The lack of discernible patterns suggests that DBSCAN may not have effectively captured meaningful subgroups within this dataset, possibly due to the high density of the primary cluster.

K-Means: The largest cluster contains almost all lightcurves with very few room for outliers. Seven of the 10 clusters each contain only one lightcurve. These single lightcurves are indeed anomalous and of interest to the thesis, but given that they contain only one lightcurve each, this can hardly be called a cluster.
Additionally, two real clusters were discovered. One group containes light curves that start with a high initial value and decrease linearly, while the other group contains light curves that start low and increase linearly.

This indicates that, even though they are not of particular interest, K-Means provided more interpretable and meaningful clusters to the PCA-reduced data.

5.1.4 LSTM Autoencoder + t-SNE + DBSCAN/K-Means

This time the features extracted by the LSTM Autoencoder are being used to cluster lightcurves.

Dimensionality Reduction with t-SNE

t-SNE is used to project the high-dimensional latent representation into a two-dimensional space for visualisation and easy interpretation. After fitting, the t-SNE visualisation (see Figure 25) reveals a significantly different structure compared to the PCA-based t-SNE

plot. Unlike the previous example where the data formed a more cohesive and dense structure, this visualization displays a large number of distinct and well-separated clusters. The boundaries between clusters are notably clearer, with clusters appearing more compact and isolated from each other. This suggests that the autoencoder effectively transformed the data, grouping similar light curves closely together while separating dissimilar patterns.
Some clusters also show slight patterns or arrangements among themselves, indicating a gradual change or relationship between certain groups of lightcurves. This suggests that there may be transitional features where one cluster evolves into another based on variations in specific attributes of the lightcurves.
DBSCAN and K-Means are used to assign labels to the clusters

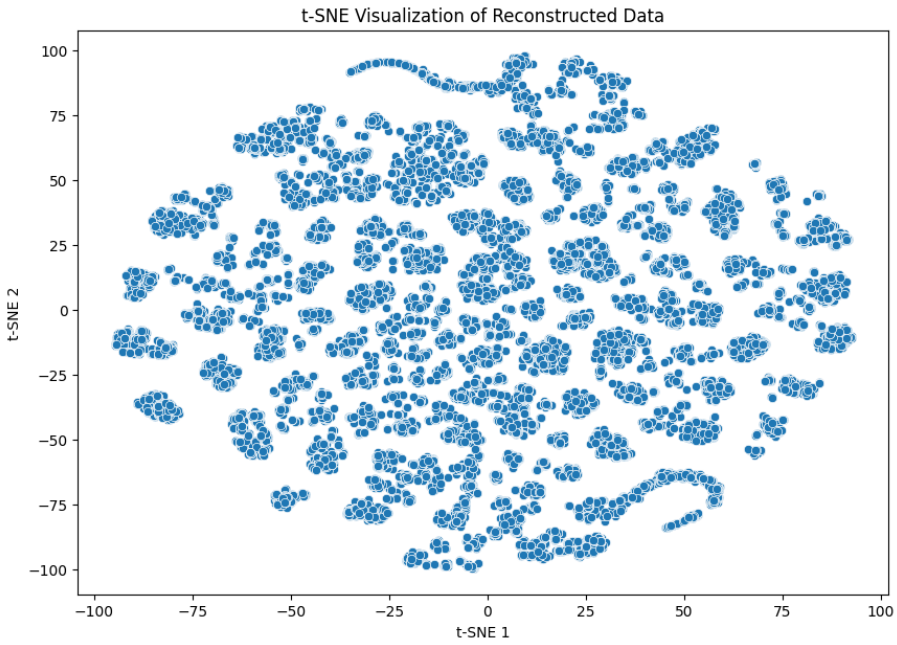


Figure 25: T-SNE applied to the latent representation.

DBSCAN Clustering

Applying DBSCAN to the t-SNE embedding of the autoencoder's latent space yields the following observations (see Figure 26):

- **Multiple Clusters:** DBSCAN identified numerous distinct clusters within the data, in contrast to the previous DBSCAN result where most data points formed a single large cluster with minimal subclusters.
- **Clear Separation:** The clusters appear clearly separated in the visualisation reflecting the compactness and distinctiveness of the groups identified.

- **Fine-Grained Variations:** The algorithm captured fine-grained variations within the data, suggesting that the autoencoder's latent space provides a detailed representation that highlights subtle differences between light curves.

K-Means Clustering

The optimal number of clusters is determined using the elbow method, which again suggests 10 clusters. Key findings from the K-Means clustering include (see Figure 27):

- **Well-defined Clusters:** The clusters formed are well-defined and distinct, with minimal overlap between them. This contrasts with the PCA-based K-Means results, where a single large cluster dominated.
- **Compactness and Separation:** The clusters are compact and well-separated, indicating that K-Means effectively segmented the data in the autoencoder's latent space.
- **Multiple Significant Clusters:** Unlike the previous approach, K-Means identified multiple significant clusters rather than one dominant cluster and several minor ones. This suggests that the autoencoder's features allowed for a betetr segmentation of the data.

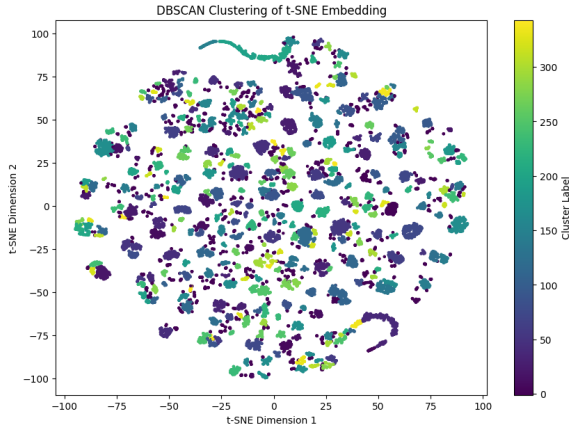


Figure 26: DBSCAN Clustering showing many small clusters.

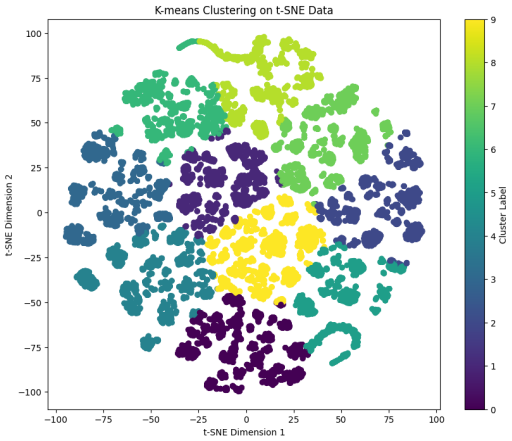


Figure 27: K-Means Clustering showing 10 equally sized clusters.

Dbscan identified a large number of clusters, capturing fine-grained variations within the data while K-Means created an optimal number of clusters based on the elbow method, resulting in a balanced segmentation without any noise points. The high number of small subgroups labelled by DBSCAN gives a more detailed but also harder

to interpret result while K-Means offers a more general categorisation that can more easily be interpreted.

Evaluation of Clustering Results

K-Means: Upon visual inspection, the light curves within each K-Means cluster do not exhibit clear similarities in shape or features. While some clusters might suggest to have some underlying structural similarities, these patterns were not consistent througout all members of the cluster.

DBSCAN: Clusters identified by DBSCAN display clearer similarities among their member light-curves. While not all show clear similarities, many have some similarity in their shape, noise level or general orientation. While interesting anomalies were found, they are not isolated into their own clusters. Instead, they appear within clusters that shared the underlying shape of the underlying curve, indicating that the anomalies might be subtle deviations from common patterns.

The evaluation suggests that DBSCAN was more efficient in grouping the extracted features based on similarities, potentially due to its sensitivity to local density variations and ability to capture clusters of different shape and size. However, the sheer number of clusters and the complexity of interpreting them pose challenges.

It can be concluded, that the LSTM Autoencoder successfully extracted valuable features from the lightcurves. However, it seems that the anomalies are too subtle to be distinctly isolated by the clustering algorithms. They tended to be grouped within clusters dominated by the underlying curve's shape, making them difficult to identify solely based on cluster membership.

5.2 Edge Mask Approach

Due to the substantial size of each mask, being 1024 time the size of a lightcurve, it is impractical (on the device used impossible) to preprocess the data as described in Chapter 3 and store it in memory. For the same reason, Principle Component Analysis (PCA) is not feasible as a preprocessing technique for the edge masks because it requires all data to be loaded simultaneously, which exceeds memory limitations. To address the data size issue, a dataloader will be used, which preprocesses each batch before feeding it to the Convolutional Autoencoder.

5.2.1 Convolutional Autoencoder

Dataset Preparation and Loading

A custom dataset class is implemented to load and process each sample on-the-fly, at all times only storing one batch size of edge masks in memory. Each loaded subscan

undergoes the preprocessing steps described in Chapter 3. To reduce computational load and allow for larger batch sizes without sacrificing essential structural information, `max pooling` with a kernel size of 2x2 is applied to downsample the images, effectively halving their spatial dimensions. A dataloader batches the preprocessed edge masks for efficient processing during training

Model Architecture
The convolutional autoencoder consists of an encoder and a decoder, designed to compress the input edge masks into a lower-dimensional latent space and then reconstruct them:
Encoder: The encoder compresses the input edge mask into a latent representation, capturing relevant spatial features of the edges. It comprises multiple convolutional layers:

- **First Layer:** A 2D convolutional layer with 8 filters and a kernel size of 3x3 extracts basic edge patterns while preserving spatial details. Padding is applied to ensure the output dimensions remain consistent with the input.
- **Intermediate Layers:** As the network deepens, subsequent convolutional layers with an increasing number of filters (16, 32, and 64) capture progressively complex patterns. Each layer employs a Rectified Linear Unit (ReLU) activation function to introduce non-linearity. Dropout layers are included after each convolutional layer, starting with lower rates and increasing in deeper layers to regulate overfitting.
- **Pooling Layers:** Max pooling with a kernel size of 2x2 is applied after each convolutional layer to downsample the feature maps. This step reduces the spatial dimensions, preventing the feature maps from becoming computationally intractable as the network depth increases.

Decoder: The decoder reconstructs the original image from the compressed latent representation through a series of upsampling operations:

- **Transposed Convolutional Layers:** Mirroring the encoder, the decoder uses transposed convolutional layers to upsample the latent feature maps. Each layer progressively restores the spatial dimensions with strides of 2, effectively reversing the compression performed by the encoder.
- **Final Output Layer:** The last transposed convolutional layer outputs a single-channel image which is the reconstructed edge mask. This output mirrors the input dimensions, allowing for direct comparison with the original edge mask.

Training Configuration and Loss Function

The model is trained using the Adam optimiser with a learning rate of 0.0001, allowing the optimiser to make small, precise adjustments to the network's weights, promoting stable and gradual convergence during training. To calculate the difference between the original and reconstructed edge masks, Binary Cross-Entropy (BCE) is used. This loss function is optimal for binary data, such as edge masks where each pixel is either an edge or not.

However, the dataset exhibits a class imbalance, with non-edge (`False`) values significantly outnumbering edge (`True`) values. To address this, weighted BCE loss is utilised, increasing the reward for correctly predicting `True` values.

The `BCEWithLogitsLoss` function is used, which internally applies a sigmoid function to the logits (raw output to the model's final layer) before computing the BCE loss. The parameter `pos_weight` is set to the overrepresentation factor of the `False` values which can be calculated by dividing the number of `False` values by the number of `True` values in a subset of the data. It is calculated to be 20 in this case. This weighing makes the model more sensitive to the minority class.

A small batch size of 32 was chosen as a compromise between computational efficiency, stable gradient updates, and generalization capabilities.

Monitoring and Early Stopping

Training is conducted over 20 epochs to provide the model with sufficient time to learn meaningful representations of the data. Similar to the LSTM Autoencoder, to prevent overfitting, a patience-based early stopping mechanism was implemented. Training is stopped if no meaningful improvement in the loss was observed over five consecutive epochs. This strategy ensures that the model maintains generalization performance without unnecessary training cycles. Training performance metrics are logged using TensorBoard.

Latent Space Extraction for Clustering

After training, the encoder component of the model is used independently to extract latent representations of each input edge mask. These latent representations can be used for subsequent clustering.

Training Results

The training unfortunately did not yield the expected results. Throughout the training process, the model's loss converged quickly towards 0.69 and remained there after. This indicates that the model was effectively making random guesses rather than learning meaningful patterns from the data. This loss value is characteristic of a model performing no better than chance in binary classification tasks:

The BCE loss function, used for training the autoencoder, is defined as:

$$\text{Loss} = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \cdot \log(p_i) + (1 - y_i) \cdot \log(1 - p_i)\right] \tag{12}$$

where:

- N is the number of samples.
- y_i is the true binary label (o fir non-edge, 1 for edge).
- p_i is the predicted probability that sample i is an edge.

When the model predicts a probability $p_i = 0.5$, regardless of the true label y_i, the loss can be simplified to:

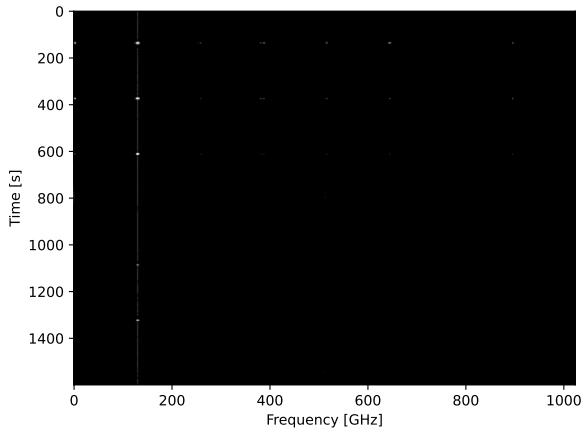
$$\text{Loss} = -\log(0.5) = 0.6931 \tag{13}$$

Several attempts were made to address the issue, including removing some dropout layers, which led to the model moving beyond random guessing. However, the model quickly converged, reaching minimal loss in less than one epoch. This was a sign of overfitting.
In an attempt to mitigate overfitting, some dropout layers were reintroduced, which reverted the model back to random guessing behaviour, with the loss returning to something around 0.69.

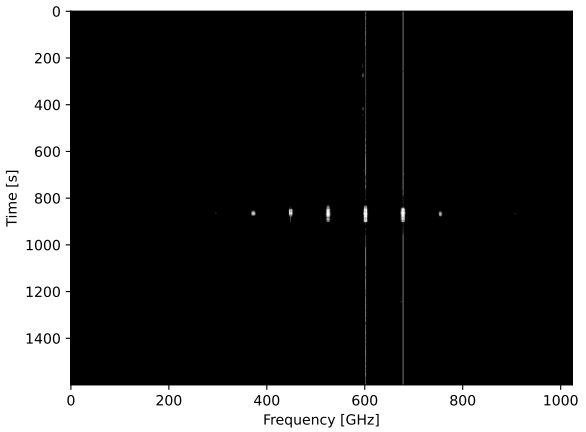
Subsequently, no attempts in clustering were made based on the edge masks, as no properly trained model exists.

5.3 Heuristic Approach

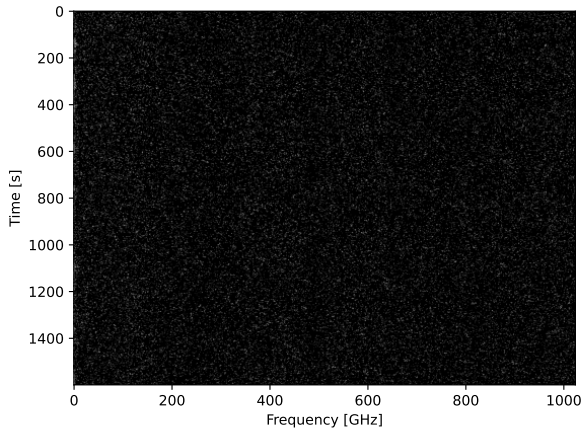
Given that no clustering method really worked as intended, clustering based on visual inspection is a possible solution. Given that using the anomaly detection methods, many anomalous subscans have been singled out, this not as a tedious task as anticipated in the beginning. Sorting all the subscans in some class was not done, but the following pages contain one samples of major classes.



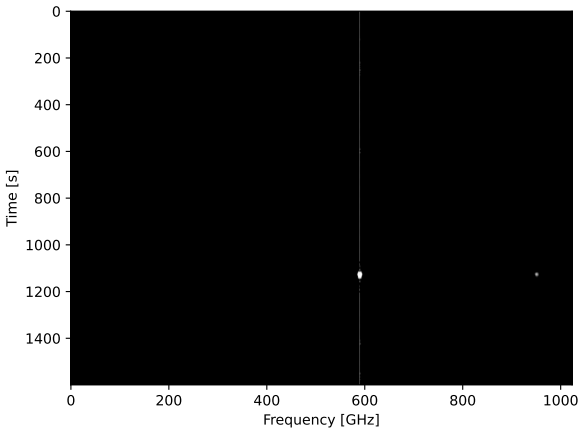
(a) DATA_2021-12-08_7848_77-21_subscan3_S20mm-SPECPOL-ARRAYDATA-1



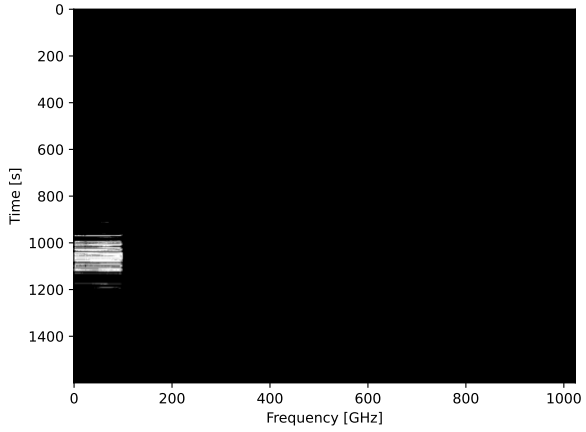
(b) DATA_2023-05-01_7635_82-22_subscan7_S20mm-SPECPOL-ARRAYDATA-1



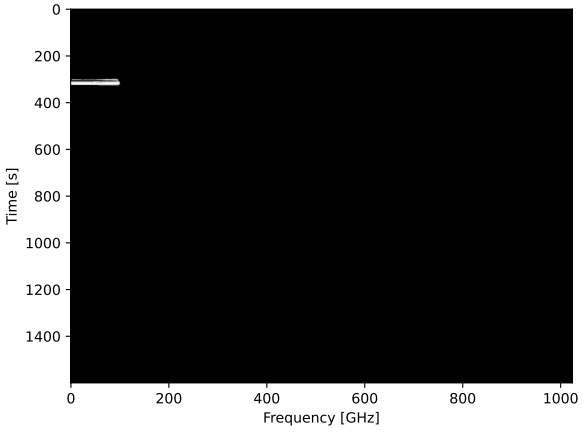
(c) DATA_2022-03-23_8456_77-21_subscan4_S20mm-SPECPOL-ARRAYDATA-1



(d) DATA_2022-04-10_0390_77-21_subscan5_S20mm-SPECPOL-ARRAYDATA-1

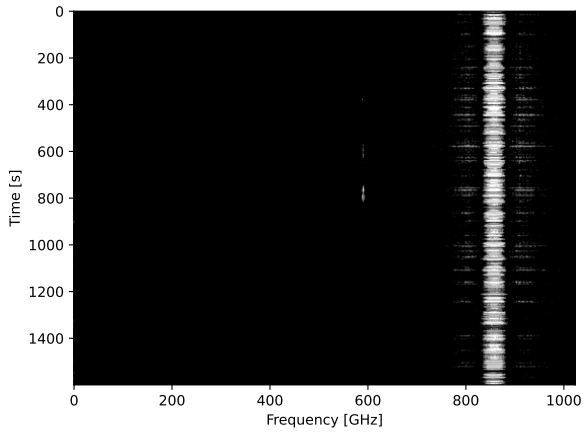


(e) DATA_2022-05-18_4572_77-21_subscan6_S20mm-SPECPOL-ARRAYDATA-2

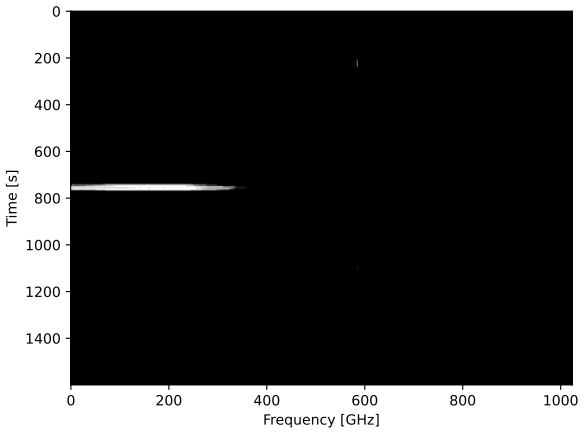


(f) DATA_2022-05-18_4572_77-21_subscan8_S20mm-SPECPOL-ARRAYDATA-2

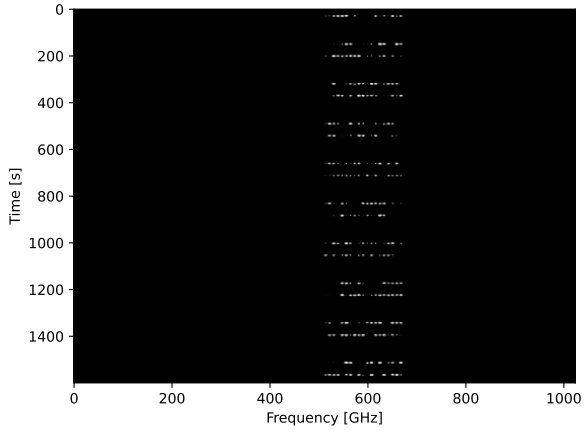
Figure 28: Some interesting anomaly types, that can be found multiple times throughout the dataset.



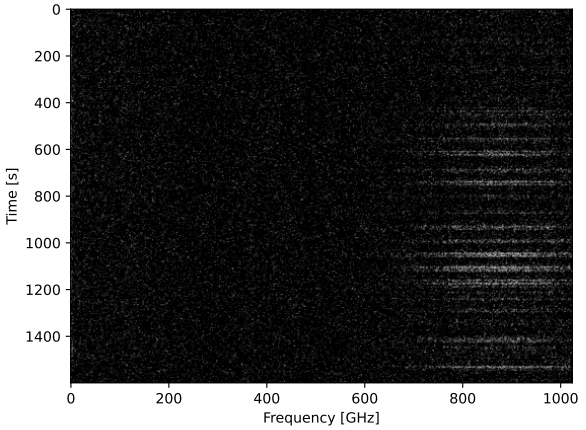
(a) DATA_2023-05-18_9267_82-22_subscan8_S20mm-SPECPOL-ARRAYDATA-1



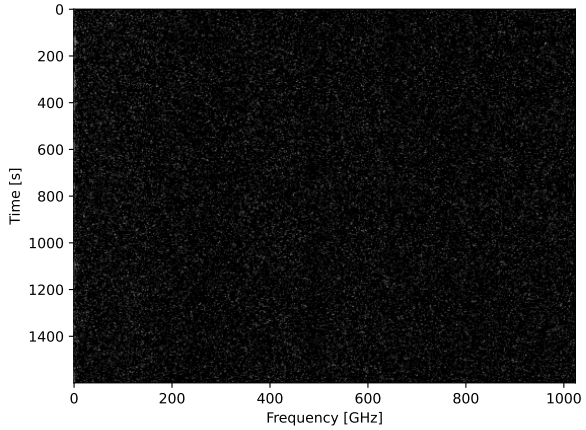
(b) DATA_2022-12-16_3306_82-22_subscan1_S20mm-SPECPOL-ARRAYDATA-1



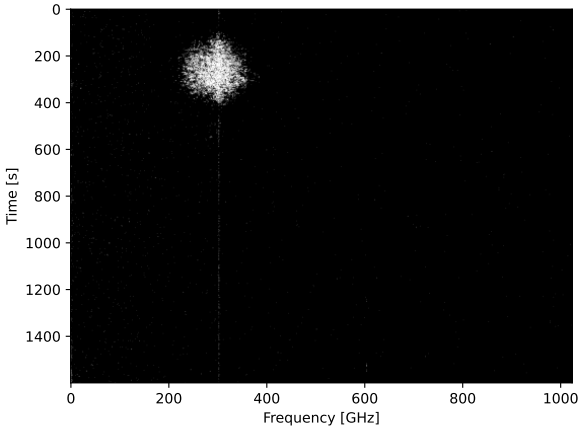
(c) DATA_2023-01-06_5278_82-22_subscan3_S20mm-SPECPOL-ARRAYDATA-2



(d) DATA_2023-03-16_2035_82-22_subscan7_S20mm-SPECPOL-ARRAYDATA-1



(e) DATA_2022-03-23_8456_77-21_subscan4_S20mm-SPECPOL-ARRAYDATA-1



(f) DATA_2023-03-22_2826_82-22_subscan3_S20mm-SPECPOL-ARRAYDATA-1

Figure 29: Some more interesting anomaly types, that can be found multiple times throughout the dataset. Only type f) was in this form only detected once.

6 Outlook

This thesis could continue forever, there are millions of different parameter combinations to try out and just as many completely new approaches:

Tweaking Parameters

The first step to be done is to make the LSTM Autoencoder sensitive to not the underlying signal but the actual anomalous signals. Changing the amount of hidden layers or other parameters might change the entire output. It was unfortunately impossible for me to try out and analyse all combinations, but someone else might.
Similar things can be said about the CAE Edge mask approach. The edge masks are a very promising data format, since the underlying signal is not present, meaning as soon as one gets the CAE to learn a representation of the edge mask, clustering the actual anomalies should be straightforward.

Training on a Subset

A next step that could help the model to focus on the representation of the anomalies is to train the model only on the known anomalous subscans. Many have been identified by the very successful anomaly detection methods...

Semi-Supervised Learning Approaches

Labelling a few subscans as anomalous or as a specific group of anomalies could help a model train better, as it provides reference points for the types of features that distinguish anomalies from typical signals. Semi-supervised learning leverages this small set of labelled data to guide the model in identifying similar patterns within the unlabelled data, offering a more structured learning approach than unsupervised methods alone. Techniques such as pseudo-labelling, in which the model generates provisional labels for unlabelled data based on its confidence, could reinforce training by treating confidently predicted samples as additional labelled data.

References

aishwarya.27. Introduction to recurrent neural network. https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/, July 23 2024. GeeksforGeeks.

Telekommunikation Post und Eisenbahnen Bundesnetzagentur für Elektrizität, Gas. Frequenzplan gemäß § 54 TKG über die Aufteilung des Frequenzbereichs von 0 kHz bis 3000 GHz auf die Frequenznutzungen sowie über die Festlegungen für diese Frequenznutzungen (Stand: MÄRZ 2022) [frequency plan in accordance with section 54 of the telecommunications act on the allocation of the frequency range from 0 khz to 3000 ghz to the frequency uses and on the provisions for these frequency uses (as of march 2022)]. Technical report, Bundesnetzagentur für Elektrizität, Gas, Telekommunikation, Post und Eisenbahnen, 2022. URL https://data.bundesnetzagentur.de/Bundesnetzagentur/SharedDocs/Downloads/DE/Sachgebiete/Telekommunikation/Unternehmen_Institutionen/Frequenzen/20210114_frequenzplan.pdf.

Telekommunikation Post und Eisenbahnen Bundesnetzagentur für Elektrizität, Gas. List of public satellite networks assigned in Germany, 8 2024. URL https://www.bundesnetzagentur.de/SharedDocs/Downloads/DE/Sachgebiete/Telekommunikation/Unternehmen_Institutionen/Frequenzen/SpezielleAnwendungen/Satellitenfunk/satellitenfunknetze.pdf?__blob=publicationFile&v=6.

F Eppel, M Kadler, J Heßdörfer, P Benke, L Debbrecht, J Eich, A Gokus, S Hämmerich, D Kirchner, GF Paraschos, et al. Telamon: Effelsberg monitoring of agn jets with very-high-energy astroparticle emission-i. program description and sample characterization. *Astronomy & Astrophysics*, 684:A11, 2024.

Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, volume 96, pages 226–231, 1996.

GeeksforGeeks. Introduction to convolution neural network. https://www.geeksforgeeks.org/introduction-convolution-neural-network/, October 10 2024.

Google. What is clustering? | Machine Learning | Google for Developers — developers.google.com. https://developers.google.com/machine-learning/clustering/overview. [Accessed 31-10-2024].

Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

S Hochreiter. Long short-term memory. *Neural Computation MIT-Press*, 1997.

Zurich Instruments. Principles of boxcar averaging. Whitepaper, Zurich Instruments, 2021.

Ian T Jolliffe and Jorge Cadima. Principal component analysis: a review and recent developments. *Philosophical transactions of the royal society A: Mathematical, Physical and Engineering Sciences*, 374(2065):20150202, 2016.

Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *2008 eighth ieee international conference on data mining*, pages 413–422. IEEE, 2008.

J Macqueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability/University of California Press*, 1967.

SANCHITA MANGALE. Scree Plot — sanchitamangale12.medium.com. `https://sanchitamangale12.medium.com/scree-plot-733ed72c8608`. [Accessed 31-10-2024].

Bartosz Mikulski. PCA—how to choose the number of components? – Bartosz Mikulski - AI consultant — mikulskibartosz.name. `https://mikulskibartosz.name/pca-how-to-choose-the-number-of-components`, 2019. [Accessed 31-10-2024].

Dirk Muders. Multi-beam fits raw data. Technical report, Atacama Pathfinder EXperiment, 2007.

Karl Pearson. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin philosophical magazine and journal of science*, 2(11):559–572, 1901.

Eduardo Ros, Dominik Schwarz, and Christian Vocks. Community paper on radio astronomy infrastructures, 02 2018.

Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Pearson, 2016.

Bernhard Schölkopf, John C Platt, John Shawe-Taylor, Alex J Smola, and Robert C Williamson. Estimating the support of a high-dimensional distribution. *Neural computation*, 13(7):1443–1471, 2001.

Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

Saeed V Vaseghi. *Advanced digital signal processing and noise reduction*. John Wiley & Sons, 2008.

Manuel Vogel. *Oxford Dictionary of Astronomy, by Ian Ridpath*, volume 53. Oxford University Press, 7 2012. doi: 10.1080/00107514.2012.699474. URL https://doi.org/10.1080/00107514.2012.699474.

Westfalen-Blatt. Flutkatastrophe: Warum die Telefone der Bundeswehr nicht funktionierten, 10 2021. URL https://www.presseportal.de/pm/66306/5060201.

Djemel Ziou and Salvatore Tabbone. Edge detection techniques-an overview. */Pattern Recognition and Image Analysis: Advances in Mathematical Theory and Applications*, 8(4):537–559, 1998.

Appendix A

Figure 30: Following the link, you will find the GitHub page to this thesis. It also contains lists containing all detected anomalous signals:
`https://github.com/Elorex/Bachelors-Thesis`.

Appendix B

Appendix B contains some important functions used in preprocessing the data. Everything else, including these functions, can be found in appendix A.

Listing 1: Extracting Stokes-I Parameter

```
def get_stokes_i(data, scantime, dt):
    spectra = []

    # Loop over each time entry in the data
    for i in range(len(data)):
        spectrum = []

        # Extract the Left Circular Polarisation (LCP) and
        # Right Circular Polarisation (RCP) arrays
        lcp = np.array(data[i][1][0])
        rcp = np.array(data[i][1][1])

        # Calculate Stokes I by summing LCP and RCP
        I = lcp + rcp

        # Append the calculated Stokes I spectrum for the
        # current time point to the spectra list
        spectra.append(I)

    # Determine the number of time steps to retain, based on
    # scantime and time interval dt
    length = int(scantime // dt)
    spectra = np.array(spectra)

    # Return only the last 'length' time steps in the spectra
    return spectra[-length:, :]
```

Listing 2: Bandpass Calibration

```
def calibrate_bandpass(spectra):
    # Iterate over each frequency bin in the spectra
    for i in range(len(spectra[0])):
        add_count = 0

        # Sum the intensity values across all time points
        # for the current frequency bin
        for j in range(len(spectra)):
            add_count += spectra[j][i]

        # Calculate the correction factor for the current
        # frequency bin
        corr_factor = add_count / len(spectra)

        # Apply the correction factor to normalize each value
        # in the current frequency bin
        for j in range(len(spectra)):
            spectra[j][i] = spectra[j][i] / corr_factor

    return spectra
```

Listing 3: Removing Noise Calibration Effects

```
def remove_noise_cal(spectra):
    # Iterate over each frequency bin in the spectra
    for j in range(len(spectra[0])):

        # Separate the two phases of the signal into two bins
        bin1 = []
        bin2 = []
        count = 2
        switch = True

        # Loop through each time step in the spectra
        for i in range(len(spectra)):
            # Toggle phase every two time steps
            if count == 0:
                switch = not switch
                count = 2

            # Append data to bin1 or bin2 based on the
            # current phase
            if switch:
                bin1.append(spectra[i][j])
            else:
                bin2.append(spectra[i][j])

            count -= 1

        # Calculate and subtract the average (mean) from
        # each bin (calibration effect)
        fit1 = np.full(len(bin1), np.average(bin1))
        fit2 = np.full(len(bin2), np.average(bin2))
        bin1 = bin1 - fit1
        bin2 = bin2 - fit2

        # Reconstruct the signal by interleaving adjusted
        # values from both bins
        final_time = []
        i = 0
        while i < len(bin1):
            final_time.append(bin1[i])
            if i + 1 < len(bin1):
                final_time.append(bin1[i + 1])
            if i < len(bin2):
                final_time.append(bin2[i])
```

```
            if i + 1 < len(bin2):
                final_time.append(bin2[i + 1])
            i += 2  # Move to next pair

        # Update the spectra with the noise-calibrated values
        # for the current frequency bin
        for i in range(len(spectra)):
            spectra[i][j] = final_time[i]

    return spectra
```

Listing 4: Removing the Source from the Spectra/Lightcurve

```
def remove_source(spectra, isLC=False):
    # Check if the input is a light curve (single spectrum)
    if isLC:
        averaged_spectrum = spectra
    else:
        # Calculate the average across time if it's 2D data
        averaged_spectrum = np.mean(spectra, axis=1)

    try:
        # Gaussian curve fitting on the averaged spectrum
        parameters, covariance = curve_fit(
            gauss,
            # x values: range of indices
            np.arange(len(averaged_spectrum)),
            # y values: averaged spectrum
            np.array(averaged_spectrum),
            p0 = np.asarray([0.1, 1 / 100,
                len(averaged_spectrum) / 2,
                np.average(averaged_spectrum)]), # Initial guess
            maxfev = 2000, # Max iterations for curve fitting
            bounds = ([0, 0, 0, -np.inf],
                      [np.inf, np.inf, len(averaged_spectrum),
                       np.inf])
        )
    except Exception as e:
        # Handle fitting failure and set parameters to zero
        parameters = [0, 0, 0, 0]
        print(f"Curve fitting failed: {e}")

    # Unpack the fitted Gaussian parameters
    # (Amplitude, Width, Center, Offset)
    fit_A1, fit_B1, fit_C1, fit_D1 = parameters

    # Generate the fitted Gaussian curve using the
    # fitted parameters
    fit = gauss(np.arange(len(averaged_spectrum)),
          fit_A1, fit_B1, fit_C1, fit_D1)

    # Subtract the fitted Gaussian from each row of spectra
    # to remove the source component
    spectra = (spectra.T - fit).T

    return spectra
```

Listing 5: Calculating the Edge Mask

```
def get_edge_mask(spectra):
     # Apply a Gaussian filter to smooth the data
    smoothed_spectra = scipy.ndimage.
                    gaussian_filter(spectra, sigma=1)

    # Define a 2D Laplacian kernel
    laplacian_kernel = np.array([[0,  -1, 0],
                                 [-1, 4, -1],
                                 [0,  -1, 0]])

    # Apply the Laplacian convolution to the smoothed data
    filtered_spectra = scipy.ndimage.
                    convolve(smoothed_spectra, laplacian_kernel)

    # Calculate the threshold value for anomaly detection
    threshold = np.mean(filtered_spectra)
            + 2 * np.std(filtered_spectra)

    # Create a binary mask of anomalies
    anomalies = np.abs(filtered_spectra) > threshold

    return anomalies
```

Listing 6: Applying the Boxcar Filter to the Lightcurve

```
def get_boxcar_filtered(intensities, window_sizes):
    filtered_intensities_list = []

    # Apply boxcar filter with different window sizes
    # to Lightcurve intensities
    for boxcar_width in window_sizes:
        filtered_intensities_list.append(
            uniform_filter1d(intensities, size=boxcar_width))

    return filtered_intensities_list
```

Listing 7: Calculating the SNR

```
def get_snr(filtered_intensities_list, window_sizes):
    SNR_list = []

    for filtered_intensities, boxcar_width in
    zip(filtered_intensities_list, window_sizes):
        half_boxcar = boxcar_width // 2
        # Find the peak in the light curve
        peak_index = np.argmax(filtered_intensities)
        peak = filtered_intensities[peak_index]

        # Check if the peak_index is within the valid
        # range considering the boxcar window
        if not (half_boxcar <= peak_index <
            len(filtered_intensities) - half_boxcar):
            SNR_list.append(0)
            continue

        # Define the signal region around the peak
        signal_indices = np.arange(
            max(0, peak_index - half_boxcar),
            min(len(filtered_intensities),
                peak_index + half_boxcar + 1))
        signal_region = filtered_intensities[signal_indices]
        # Define the noise region (everything outside the
        # signal region)
        noise_indices = np.setdiff1d(np.arange(
            len(filtered_intensities)), signal_indices)
        noise_region = filtered_intensities[noise_indices]
        # Calculate the standard deviation of the noise region
        std_noise = np.std(noise_region)
        # Calculate the mean of the noise region (Baseline)
        mean_noise = np.mean(noise_region)
        # Calculate the signal-to-noise ratio
        SNR_list.append((peak - mean_noise) / std_noise)

    return SNR_list
```

Eidesstattliche Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Würzburg, den January 7, 2025 Vorname Nachname