Jan Kowalczyk

LiDAR Degradation Quantification for Robot Navigation in Hazy Environments

MASTER'S THESIS

submitted to

Graz University of Technology

Supervisors

Univ.-Prof. Dipl.-Ing. Dr.mont Franz Pernkopf

Signal Processing and Speech Communication Laboratory

in cooperation with

Virtual Vehicle Research GmbH
Graz, Austria

Graz, October, 2025

Affidavit

Artificial Intelligence Usage Disclaimer

During the creation of this thesis, an LLM-based Artificial Intelligence tool was used for stylistic and grammatical revision of the author's own work.

Abstract

Autonomous robots are increasingly used in search and rescue (SAR) missions. In these missions, LiDAR sensors are often the most important source of environmental data. However, LiDAR data can degrade under hazardous conditions, especially when airborne particles such as smoke or dust are present. This degradation can lead to errors in mapping and navigation and may endanger both the robot and humans. Therefore, robots need a way to estimate the reliability of their LiDAR data, so that they can make better-informed decisions.

This thesis investigates whether anomaly detection methods can be used to quantify LiDAR data degradation caused by airborne particles such as smoke and dust. We apply a semi-supervised deep learning approach called DeepSAD, which produces an anomaly score for each LiDAR scan, serving as a measure of data reliability.

We evaluate this method against baseline methods on a subterranean dataset that includes LiDAR scans degraded by artificial smoke. Our results show that DeepSAD consistently outperforms the baselines and can clearly distinguish degraded from normal scans. At the same time, we find that the limited availability of labeled data and the lack of robust ground truth remain major challenges. Despite these limitations, our work demonstrates that anomaly detection methods are a promising tool for LiDAR degradation quantification in SAR scenarios.

Contents

1 Introduction

Autonomous robots are increasingly used in search and rescue (SAR) missions, as they can take on dangerous tasks without putting additional human lives at risk. These tasks include navigating hazardous environments such as collapsed structures, identifying and locating victims, and assessing whether the environment is safe for human rescue teams. To perceive their surroundings, robots rely on multiple sensor systems such as LiDAR, radar, time-of-flight (ToF), ultrasound, optical cameras, or infrared cameras. Among these, LiDAR is the most widely used due to its high accuracy. The data from these sensors allows robots to map their environments, navigate through them, and make decisions such as which paths to prioritize. Many of the underlying algorithms are based on deep learning, trained on large datasets from which the models learn characteristic patterns.

Search and rescue environments pose difficult conditions for sensor systems to produce reliable data. A prominent challenge is the presence of aerosol particles such as smoke and dust, which can obstruct visibility and cause sensors to generate erroneous data. If such degraded conditions were not represented in the training data of the robot's algorithms, these errors may lead to unexpected outputs and potentially endanger both the robot and human rescue targets. This is especially critical for autonomous robots, whose decisions rely entirely on sensor data without human oversight. To mitigate these risks, robots must be able to assess the trustworthiness of their sensor data.

For remotely controlled robots, a human operator can make these decisions, but many search and rescue missions do not allow remote control due to environmental factors, such as radio signal attenuation or the search area's size, and therefore demand autonomous robots. Therefore, during the design of such robots, we arrive at the following critical research question:

> Can autonomous robots quantify the reliability of LiDAR sensor data in hazardous environments to make more informed decisions?

In this thesis, we aim to answer this question by assessing a deep learning-based anomaly detection method and its performance when quantifying the sensor data's degradation. The employed algorithm is a semi-supervised anomaly detection algorithm that uses manually labeled training data to improve its performance over unsupervised methods. We compare the method's performance with common baseline methods from the same class of algorithms. The model's output is an anomaly score that quantifies the data reliability and can be used by algorithms that rely on the sensor data. These reliant algorithms may decide to, for example, slow down the robot to collect more data, choose alternative routes, signal for help, or rely more heavily on other sensors' input data.

Our experiments demonstrate that anomaly detection methods are indeed applicable to this task, allowing LiDAR data degradation to be quantified on subterranean datasets representative of SAR environments. Among the tested approaches, the semi-supervised method consistently outperformed established baselines. At the same time, the lack of suitable training data—and in particular the scarcity of reliable evaluation labels—proved to be a major limitation, constraining the extent to which the expected real-world performance of these methods could be assessed.

1.1 Scope of Research

In this thesis, we focus our research on the unique challenges faced by autonomous rescue robots, specifically the degradation of sensor data caused by airborne particles. While degradation in sensor data can also arise from adverse weather, material properties, or dynamic elements such as moving leaves, these factors are considered less relevant to the rescue scenarios targeted by our study and are therefore excluded. Although our method is versatile enough to quantify various types of degradation, our evaluation is limited to degradation from airborne particles, as this is the most prevalent issue in the operational environments of autonomous rescue robots.

While robotic computer vision systems often incorporate a variety of sensors—such as time-of-flight cameras, infrared cameras, and ultrasound sensors—we found that autonomous rescue robots primarily depend on LiDAR data for mapping and navigation. LiDAR sensors offer high accuracy, high resolution, and an extensive field of view (often a full 360° horizontal and a substantial vertical coverage), which are essential for constructing comprehensive environmental maps in challenging scenarios. Furthermore, the cost of LiDAR sensors has decreased significantly in recent decades, driven by their widespread adoption in autonomous driving, drones, and robotics, as well as manufacturing advancements like microelectromechanical systems (MEMS). For these reasons, our research is focused exclusively on LiDAR sensor data—specifically, the point clouds generated within a defined coordinate system. Although sensor fusion techniques are commonly used to enhance data accuracy and confidence, incorporating fused data would not only add significant complexity to our study but also limit our analysis to platforms equipped with all the sensor types involved. Consequently, we concentrate on quantifying sensor degradation solely through LiDAR data.

The method we employ produces an analog score that reflects the confidence in the sensor data, with lower confidence indicating higher degradation. Although we do not investigate the direct applications of this score, potential uses include simple thresholding to decide whether to proceed with a given action, as well as dynamically adjusting the robot's speed based on data quality to collect additional data when confidence is low. Importantly, this output score is a snapshot for each LiDAR scan and does not incorporate temporal information. While many LiDAR sensors capture multiple scans per second—enabling the possibility of time-series analyses such as running averages or more advanced statistical evaluations—we focus solely on individual scans without examining the differences between successive scans.

1.2 Structure of the Thesis

The remainder of this thesis is organized as follows. Chapter 2 introduces the theoretical background and related work, covering anomaly detection methods, semi-supervised learning algorithms, autoencoders, and the fundamentals of LiDAR sensing. Chapter 3 presents the DeepSAD algorithm in detail, including its optimization objective, network architecture, and hyperparameters. In Chapter 4, we describe the dataset, the preprocessing pipeline, and our labeling strategies. Chapter 5 outlines the experimental design, implementation details, and evaluation protocol. The results are presented and discussed in Chapter 6, where we analyze the performance of DeepSAD compared to baseline methods. Finally, Chapter 7 concludes the thesis by summarizing the main findings, highlighting limitations, and discussing open questions and directions for future work.

2 Background and Related Work

This thesis tackles a broad, interdisciplinary challenge at the intersection of robotics, computer vision, and data science. In this chapter, we introduce the background of anomaly detection, which we formulate our degradation quantification problem as. Anomaly detection has its roots in statistical analysis and has been successfully applied in various domains. Recently, the incorporation of learning-based techniques, particularly deep learning, has enabled more efficient and effective analysis of large datasets.

Because anomalies are, by nature, often unpredictable in form and structure, unsupervised learning methods are widely used since they do not require pre-assigned labels—a significant advantage when dealing with unforeseen data patterns. However, these methods can be further refined through the integration of a small amount of labeled data, giving rise to semi-supervised approaches. The method evaluated in this thesis, DeepSAD, is a semi-supervised deep learning approach that also leverages an autoencoder architecture in its design. Autoencoders have gained widespread adoption in deep learning for their ability to extract features from unlabeled data, which is particularly useful for handling complex data types such as LiDAR scans.

LiDAR sensors function by projecting lasers in multiple directions near-simultaneously, measuring the time it takes for each reflected ray to return. Using the angles and travel times, the sensor constructs a point cloud that is often accurate enough to map the sensor's surroundings. In the following sections, we will delve into these technologies, review how they work, how they are generally used, how we employ them in this thesis, and explore related work from these backgrounds.

2.1 Anomaly Detection

Anomaly detection refers to the process of detecting unexpected patterns of data, outliers that deviate significantly from the majority of data, which is implicitly defined as normal by its prevalence. In classic statistical analysis, these techniques have been studied as early as the 19th century [1]. Since then, a multitude of methods and use cases for them have been proposed and studied. Examples of applications include healthcare, where computer vision algorithms are used to detect anomalies in medical images for diagnostics and early detection of diseases [2], detection of fraud in decentralized financial systems based on blockchain technology [3], as well as fault detection in industrial machinery using acoustic sound data [4].

Figure 2.1 depicts a simple but illustrative example of data that can be classified as either normal or anomalous and shows the problem that anomaly detection methods try to generally solve. A successful anomaly detection method would somehow learn to differentiate normal from anomalous data, for example, by learning the boundaries around the available normal data and classifying it as either normal or anomalous based on its location inside or outside of those boundaries. Another possible approach could calculate an analog value that correlates with the likelihood of a sample being anomalous, for example, by using the sample's distance from the closest normal data cluster's center.

By their very nature, anomalies are rare occurrences and oftentimes unpredictable in nature, which makes it hard to define all possible anomalies in any system. It also makes it very

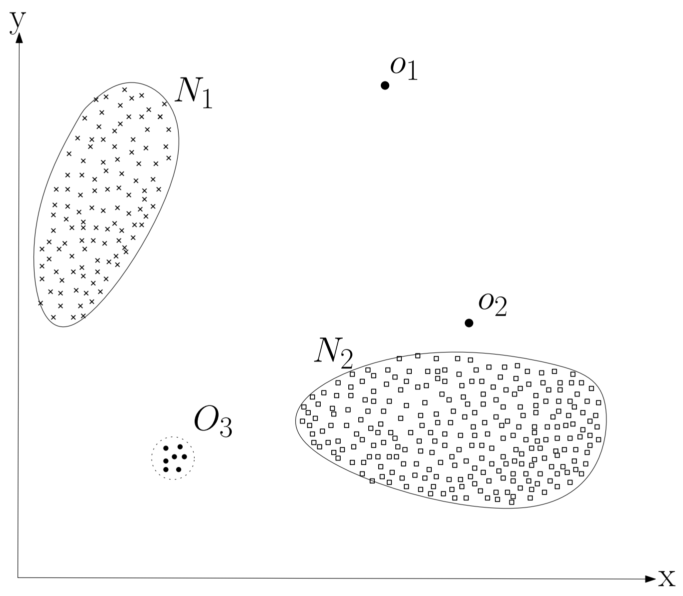


Figure 2.1: An illustrative example of anomalous and normal data containing 2-dimensional data with clusters of normal data N_1 and N_2, as well as two single anomalies o_1 and o_2 and a cluster of anomalies O_3. Reproduced from [5].

challenging to create an algorithm that is capable of detecting anomalies that may have never occurred before and may not have been known to exist during the creation of the detection algorithm. There are many possible approaches to this problem, though they can be roughly grouped into six distinct categories based on the techniques used [5]:

1. **Classification-Based**
   A classification technique, such as Support Vector Machine (SVM) [6], is used to classify samples as either normal or anomalous based on labeled training data. Alternatively, if not enough labeled training data is available, a one-class classification algorithm can be employed. In that case, the algorithm assumes all training samples to be normal and then learns a boundary around the normal samples to differentiate them from anomalous samples.

2. **Clustering-Based**
   Clustering techniques such as K-Means [7] or DBSCAN [8] aim to group similar data into clusters, differentiating it from dissimilar data, which may belong to another cluster or no cluster at all. Anomaly detection methods from this category employ such a technique, with the assumption that normal data will assemble into one or more clusters due to their similar properties, while anomalies may create their own smaller clusters, not belonging to any cluster at all, or at least be at an appreciable distance from the closest normal cluster's center.

3. **Nearest Neighbor Based**
   Similar to the clustering-based category, these techniques assume normal data is more closely clustered than anomalies and therefore utilize either a sample's distance to its k^{th} nearest neighbor or the density of its local neighborhood to judge whether a sample is anomalous.

4. **Statistical**
   These methods try to fit a statistical model of the normal behavior to the data. After the distribution from which normal data originates is defined, samples can be found to be normal or anomalous based on their likelihood of arising from that distribution.

5. **Information-Theoretic**
   The main assumption for information-theoretic anomaly detection methods is that anomalies differ somehow in their information content from anomalous data. An information-theoretic measure is therefore used to determine irregularities in the data's information content, enabling the detection of anomalous samples.

6. **Spectral**
   Spectral approaches assume the possibility of mapping data into a lower-dimensional space, where normal data appears significantly different from anomalous data. To this end, a dimensionality reduction technique such as Principal Component Analysis (PCA) [9] is used to embed the data into a lower-dimensional subspace. Spectral methods are often used as a pre-processing step followed by another anomaly detection method operating on the data's subspace.

In this thesis, we used an anomaly detection method, namely "Deep Semi-Supervised Anomaly Detection" (DeepSAD) [10], to model our problem—how to quantify the degradation of LiDAR sensor data—as an anomaly detection problem. We do this by classifying good-quality data as normal and degraded data as anomalous, and rely on a method that can express each sample's likelihood of being anomalous as an analog anomaly score, which enables us to interpret it as the data degradation quantification value.

Chapter 3 describes DeepSAD in more detail, which shows that it is a clustering-based approach with a spectral pre-processing component, in that it uses a neural network to reduce the input's dimensionality while simultaneously clustering normal data closely around a given centroid. It then produces an anomaly score by calculating the geometric distance between a data sample and the aforementioned cluster centroid, assuming the distance is shorter for normal than for anomalous data. Since our data is high-dimensional, it makes sense to use a spectral method to reduce its dimensionality. Moreover, reporting an analog value rather than a binary classification is useful for our use case since we want to quantify not only classify the data degradation.

There is a wide set of problems in domains similar to the one we research in this thesis, for which modeling them as anomaly detection problems has been proven successful. The degradation of point clouds, produced by an industrial 3D sensor, has been modeled as an anomaly detection task in [11]. Bergmann and Sattlegger propose a student-teacher model capable of inferring a pointwise anomaly score for degradation in point clouds. The teacher network is trained on an anomaly-free dataset to extract dense features of the point clouds' local geometries, after which an identical student network is trained to emulate the teacher network's outputs. For degraded point clouds, the regression between the teacher's and student's outputs is calculated and interpreted as the anomaly score, with the rationalization that the student network has not observed features produced by anomalous geometries during training, leaving it incapable of producing a similar output as the teacher for those regions. Another example would be [12], which proposes a method to detect and classify pole-like objects in urban point cloud data, to differentiate between natural and man-made objects such as street signs, for autonomous driving purposes. An anomaly detection method was used to identify the vertical pole-like objects in the point clouds, and then the preprocessed objects were grouped by similarity using a clustering algorithm to classify them as either trees or man-made poles.

As already shortly mentioned at the beginning of this section, anomaly detection methods and their usage are oftentimes challenged by the limited availability of anomalous data, owing to the very nature of anomalies, which are rare occurrences. Oftentimes, the intended use case is to even find unknown anomalies in a given dataset that have not yet been identified. In addition, it can be challenging to classify anomalies correctly for complex data, since the very definition of an anomaly is dependent on many factors, such as the type of data, the intended use case, or even how the data evolves over time. For these reasons, most types of anomaly detection approaches limit their reliance on anomalous data during training, and many of them do not differentiate between normal and anomalous data at all. DeepSAD is a semi-supervised method that is characterized by using a mixture of labeled and unlabeled data.

2.2 Semi-Supervised Learning Algorithms

Machine learning refers to algorithms capable of learning patterns from existing data to perform tasks on previously unseen data, without being explicitly programmed to do so [13]. Central to many approaches is the definition of an objective function that measures how well the model is performing. The model's parameters are then adjusted to optimize this objective. By leveraging these data-driven methods, machine learning can handle complex tasks across a wide range of domains.

Among the techniques employed in machine learning, neural networks have become especially prominent over the past few decades due to their ability to achieve state-of-the-art results across a wide variety of domains. They are most commonly composed of layers of interconnected artificial neurons. Each neuron computes a weighted sum of its inputs, adds a bias term, and then applies a nonlinear activation function, enabling the network to model complex nonlinear relationships. These layers are typically organized into three types:

- Input layer, which receives raw data.
- Hidden layers, where the network transforms and extracts complex features by combining signals through successive nonlinear operations. Networks with at least two hidden layers are typically called deep learning networks.
- Output layer, which produces the network's final prediction.

As outlined above, neural network training is formulated as an optimization problem: we define an objective function that measures how well the model is achieving its task, and then we adjust the network's parameters to optimize that objective. The most common approach is stochastic gradient descent (SGD) or one of its variants. In each training iteration, the network first performs a forward pass to compute its outputs and evaluate the objective, then a backward pass—known as backpropagation—to calculate gradients of the objective with respect to every weight in the network. These gradients indicate the direction in which each weight should change to improve performance, and the weights are updated accordingly. Repeating this process over many iterations (also called epochs) allows the network to progressively refine its parameters and better fulfill its task.

Aside from the underlying technique, one can also categorize machine learning algorithms by the type of feedback provided during learning, for the network to improve. Broadly speaking, three main categories—supervised, unsupervised, and reinforcement learning—exist, although many other approaches do not exactly fit any of these categories and have spawned less common categories like semi-supervised or self-supervised learning.

In supervised learning, each input sample is paired with a "ground-truth" label representing the desired output. During training, the model makes a prediction, and a loss function quantifies the difference between the prediction and the truth label. The learning algorithm then adjusts its parameters to minimize this loss, improving its performance over time. Labels are typically categorical (used for classification tasks, such as distinguishing "cat" from "dog") or continuous (used for regression tasks, like predicting a temperature or distance). Figure 2.2 (b) illustrates this principle with a classification example, where labeled data is used to learn a boundary between two classes.

In unsupervised learning, models work directly with raw data, without any ground-truth labels to guide the learning process. Instead, they optimize an objective that reflects the discovery

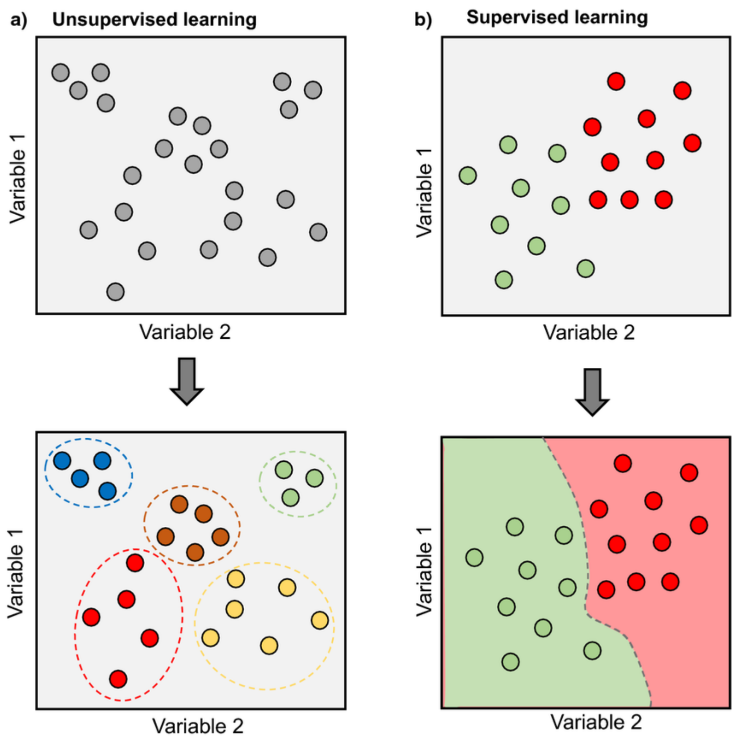


Figure 2.2: Conceptual illustration of unsupervised (a) and supervised (b) learning. In (a), the inputs are two-dimensional data without labels, and the algorithm groups them into clusters without external guidance. In (b), the inputs have class labels (colors), which serve as training signals for learning a boundary between the two classes. Reproduced from [14].

of useful structure—whether that is grouping similar data points together or finding a compact representation of the data. For example, cluster analysis partitions the dataset into groups so that points within the same cluster are more similar to each other (according to a chosen similarity metric) than to points in other clusters, which can be seen in the toy example in Figure 2.2 (a). Dimensionality reduction methods, on the other hand, project high-dimensional data into a lower-dimensional space, optimizing for minimal loss of the original data's meaningful information.

In reinforcement learning, an agent learns by trial and error while interacting with an environment. After each action, it receives feedback in the form of rewards or penalties and adapts its strategy to maximize the total reward over time. This makes reinforcement learning particularly suited for sequential decision-making tasks such as robotics or game playing.

Semi-supervised learning algorithms are an in-between category of supervised and unsupervised algorithms, in that they use a mixture of labeled and unlabeled data. Typically, vastly more unlabeled data is used during training of such algorithms than labeled data, due to the effort and expertise required to label large quantities of data correctly. Semi-supervised methods are often an effort to improve a machine learning algorithm belonging to either the supervised or unsupervised category. Supervised methods, such as classification tasks, are enhanced by using large amounts of unlabeled data to augment the supervised training without the need for additional labeling work. Alternatively, unsupervised methods like clustering algorithms may not only use unlabeled data but also improve their performance by considering some hand-labeled data during training.

Machine learning based anomaly detection methods can utilize techniques from all of the aforementioned categories, although their suitability varies. While supervised anomaly detection methods exist, their usability not only depends on the availability of labeled training data but also on a reasonable proportionality between normal and anomalous data. Both requirements can be challenging due to labeling often being labor-intensive and anomalies' intrinsic property to occur rarely when compared to normal data, making the capture of enough anomalous behavior a hard problem. Semi-supervised anomaly detection methods are of special interest in that they may overcome these difficulties inherently present in many anomaly detection tasks [15]. These methods typically have the same goal as unsupervised anomaly detection methods, which is to model the normal class behavior and delimitate it from anomalies, but they can incorporate some hand-labeled examples of normal and/or anomalous behavior to improve their performance over fully unsupervised methods. DeepSAD is a semi-supervised method that extends its unsupervised predecessor Deep SVDD [16] by including some labeled samples during training. Both DeepSAD and Deep SVDD also utilize an autoencoder in a pretraining step, a machine learning architecture, which we will look at next.

2.3 Autoencoder

Autoencoders are a type of neural network architecture whose main goal is learning to encode input data into a representative state, from which the same input can be reconstructed, hence the name. They typically consist of two functions, an encoder and a decoder with a latent space in between them, as depicted in the toy example in Figure 2.3. The encoder learns to extract the most significant features from the input and to convert them into the input's latent space representation. The reconstruction goal ensures that the most prominent features of the input are retained during the encoding phase, due to the inherent inability to reconstruct the input if too much relevant information is missing. The decoder simultaneously learns to reconstruct the original input from its encoded latent space representation by minimizing the error between the input sample and the autoencoder's output. This optimization goal complicates the categorization of autoencoders as unsupervised methods. Although they do not require labeled data, they still compute an error against a known target—the input itself. For this reason, some authors describe them as a form of self-supervised learning, where the data provides its own supervisory signal without requiring expert labeling.

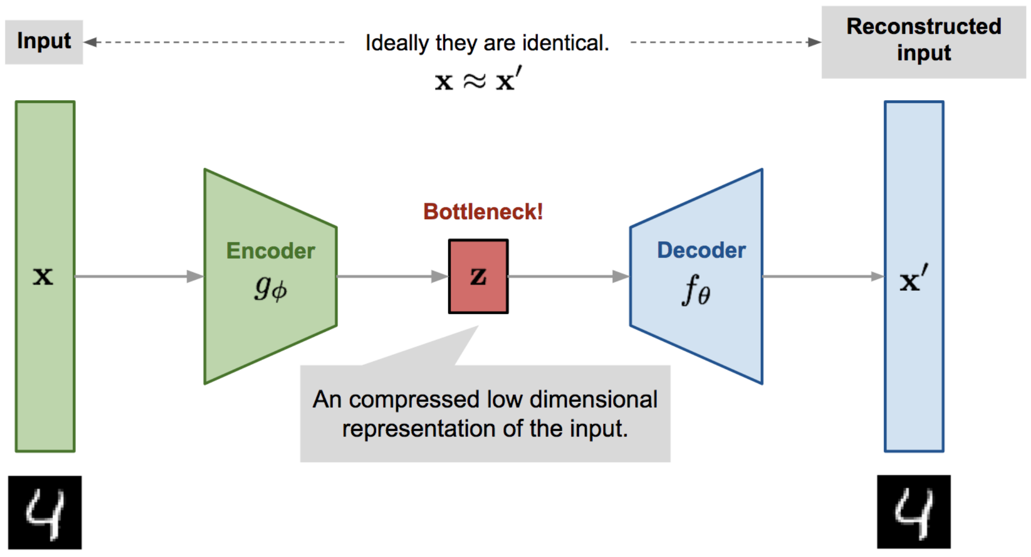

Input
Ideally they are identical.
$\mathbf{x} \approx \mathbf{x}'$
Reconstructed input
$\mathbf{x}$
Encoder
g_ϕ
Bottleneck!
$\mathbf{z}$
Decoder
f_θ
$\mathbf{x}'$
An compressed low dimensional representation of the input.

Figure 2.3: Illustration of an autoencoder's working principle. The encoder $\mathbf{g}_\phi$ compresses the input into a lower-dimensional bottleneck representation $\mathbf{z}$, which is then reconstructed by the decoder $\mathbf{f}_\theta$. During training, the difference between input and output serves as the loss signal to optimize both the encoder's feature extraction and the decoder's reconstruction. Reproduced from [17].

One key use case of autoencoders is to employ them as a dimensionality reduction technique.

In that case, the latent space in between the encoder and decoder is of a lower dimensionality than the input data itself. Due to the aforementioned reconstruction goal, the shared information between the input data and its latent space representation is maximized, which is known as following the Infomax principle [18]. After training such an autoencoder, it may be used to generate lower-dimensional representations of the given datatype, enabling more performant computations that may have been infeasible to achieve on the original data. DeepSAD uses an autoencoder in a pretraining step to achieve this goal, among others.

Autoencoders have been shown to be useful in the anomaly detection domain by assuming that autoencoders trained on more normal than anomalous data are better at reconstructing normal behavior than anomalous one. This assumption allows methods to utilize the reconstruction error as an anomaly score. Examples of this are the methods in [19] or [20] that both employ an autoencoder and the aforementioned assumption. Autoencoders have also been shown to be a suitable dimensionality reduction technique for LiDAR data, which is frequently high-dimensional and sparse, making feature extraction and dimensionality reduction popular preprocessing steps. As an example, [21] shows the feasibility and advantages of using an autoencoder architecture to reduce LiDAR-orthophoto fused features' dimensionality for their building detection method, which can recognize buildings in visual data taken from an airplane. Similarly, we can make use of the dimensionality reduction in DeepSAD's pretraining step, since our method is intended to work with high-dimensional LiDAR data.

2.4 LiDAR - Light Detection and Ranging

LiDAR (Light Detection and Ranging) measures distance by emitting short laser pulses and timing how long they take to return, an approach many may be familiar with from the more commonly known radar technology, which uses radio-frequency pulses and measures their return time to gauge an object's range. Unlike radar, however, LiDAR operates at much shorter wavelengths and can fire millions of pulses per second, achieving millimeter-level precision and dense, high-resolution 3D point clouds. This fine granularity makes LiDAR ideal for applications such as detailed obstacle mapping, surface reconstruction, and autonomous navigation in complex environments.

Because the speed of light in air is effectively constant, multiplying half the round-trip time by that speed gives the distance between the LiDAR sensor and the reflecting object, as can be seen in Figure 2.4. Modern spinning multi-beam LiDAR systems emit up to millions of these pulses every second. Each pulse is sent at a known combination of horizontal and vertical angles, creating a regular grid of measurements: for example, 32 vertical channels swept through 360° horizontally at a fixed angular spacing. While newer solid-state designs (flash, MEMS, phased-array) are emerging, spinning multi-beam LiDAR remains the most commonly seen type in autonomous vehicles and robotics because of its proven range, reliability, and mature manufacturing base.

Each time a LiDAR emits and receives a laser pulse, it can use the ray's direction and the calculated distance to produce a single three-dimensional point. By collecting up to millions of such points each second, the sensor constructs a "point cloud"—a dense set of 3D coordinates relative to the LiDAR's own position. In addition to X, Y, and Z, many LiDARs also record the intensity or reflectivity of each return, providing extra information about the surface properties of the object hit by the pulse.

LiDAR's high accuracy, long range, and full-circle field of view make it indispensable for tasks like obstacle detection, simultaneous localization and mapping (SLAM) [23], and terrain modeling in autonomous driving and mobile robotics. While complementary sensors—such as time-of-flight cameras, ultrasonic sensors, and RGB cameras—have their strengths at short range or in particular lighting, only LiDAR delivers the combination of precise 3D measurements

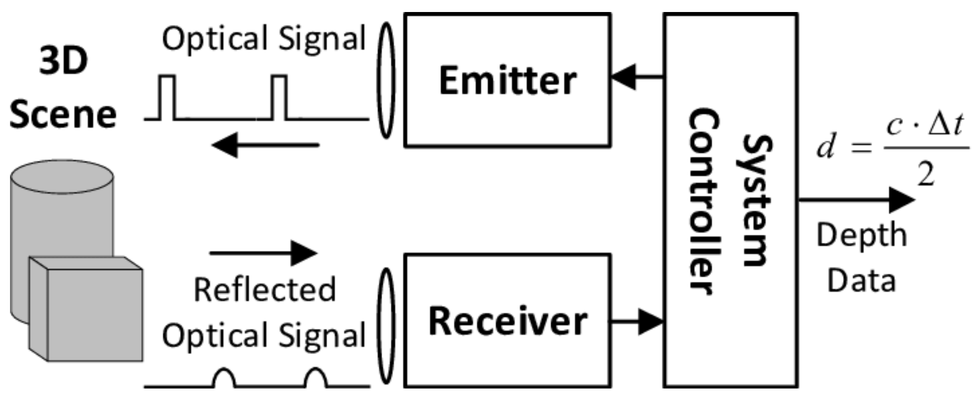


Figure 2.4: Illustration of the working principle of a LiDAR sensor. The emitter sends out an optical signal that is reflected by objects in the scene and captured by the receiver. The system controller measures the time delay Δt between emission and reception to calculate distance $d = c \cdot \Delta t/2$. By repeating this process across many directions—either with multiple emitter/receiver pairs or sequentially in a spinning LiDAR—the sensor obtains a dense set of distances that, combined with their emission angles, form a 3D point cloud of the environment. Reproduced from [22].

over medium to long distances, consistent performance regardless of illumination, and the point cloud density needed for safe navigation. LiDAR systems do exhibit intrinsic noise (e.g., range quantization or occasional multi-return ambiguities), but in most robotic applications, these effects are minor compared to environmental degradation.

In subterranean and rescue domain scenarios, the dominant challenge is airborne particles: dust kicked up by debris or smoke from fires. These aerosols create early returns that can mask real obstacles and cause missing data behind particle clouds, undermining SLAM and perception algorithms designed for cleaner data. This degradation is a type of atmospheric scattering, which can be caused by any kind of airborne particulates (e.g., snowflakes) or liquids (e.g., water droplets). Other kinds of environmental noise exist as well, such as specular reflections caused by smooth surfaces, beam occlusion due to close objects blocking the sensor's field of view, or even thermal drift–temperature changes affecting the sensor's circuits and mechanics, introducing biases in the measurements.

All of these may create unwanted noise in the point cloud created by the LiDAR, making this domain an important research topic. In [24], an overview of the current state of research into denoising methods for LiDAR in adverse environments is given. It categorizes them according to their approach (distance-, intensity-, or learning-based) and concludes that all approaches have merits but also open challenges to solve, for autonomous systems to safely navigate these adverse environments. The current research is heavily focused on the automotive domain, which can be observed by the vastly higher number of methods filtering noise from adverse weather effects–environmental scattering from rain, snow, and fog-than from dust, smoke, or other particles occurring rarely in the automotive domain.

A learning-based method to filter dust-caused degradation from LiDAR is introduced in [25]. The authors employ a convolutional neural network to classify dust particles in LiDAR point clouds, enabling the filtering of those points, and compare their methods to more conservative approaches, such as various outlier removal algorithms. Another relevant example would be the filtering method proposed in [26], which enables the filtration of point clouds degraded by smoke or dust in subterranean environments, with a focus on the search and rescue domain. To achieve this, they formulated a filtration framework that relies on dynamic onboard statistical cluster outlier removal to classify and remove dust particles in point clouds.

Our method does not aim to remove the noise or degraded points in the LiDAR data, but to quantify its degradation to inform other systems of the autonomous robot about the data's quality, enabling more informed decisions. One such approach, though from the autonomous driving and not from the search and rescue domain, can be found in [27], where a learning-based method to quantify the LiDAR sensor data degradation caused by adverse weather effects was proposed. They posed the problem as an anomaly detection task and utilized DeepSAD to learn degraded data to be an anomaly and high-quality data to be normal behaviour. DeepSAD's anomaly score was used as the degradation quantification score. From this example, we decided to imitate this method and adapt it for the search and rescue domain, although this proved challenging due to the more limited data availability. Since it was effective for this closely related use case, we also employed DeepSAD, whose detailed workings we present in the following chapter.

3 DeepSAD: Semi-Supervised Anomaly Detection

In this chapter, we explore the method DeepSAD [10], which we employ to quantify the degradation of LiDAR scans caused by airborne particles in the form of artificially introduced water vapor from a theater smoke machine. A similar approach—modeling degradation quantification as an anomaly detection task—was successfully applied in [27] to assess the impact of adverse weather conditions on LiDAR data for autonomous driving applications. DeepSAD leverages deep learning to capture complex anomalous patterns that classical statistical methods might miss. Furthermore, by incorporating a limited amount of hand-labeled data (both normal and anomalous), it can more effectively differentiate between known anomalies and normal data compared to purely unsupervised methods, which typically learn only the most prevalent patterns in the dataset [10].

3.1 Algorithm Description

DeepSAD's overall mechanics are similar to clustering-based anomaly detection methods, which, according to [5], typically follow a two-step approach. First, a clustering algorithm groups data points around a centroid; then, the distances of individual data points from this centroid are calculated and used as anomaly scores. In DeepSAD, these concepts are implemented by employing a neural network, which is jointly trained to map input data onto a latent space and to minimize the volume of a data-encompassing hypersphere, whose center is the aforementioned centroid. The data's geometric distance in the latent space to the hypersphere center is used as the anomaly score, where a larger distance between data and centroid corresponds to a higher probability of a sample being anomalous. This is achieved by shrinking the data-encompassing hypersphere during training, proportionally to all training data, of which it is required that there is significantly more normal than anomalous data present. The outcome of this approach is that normal data gets clustered more closely around the centroid, while anomalies appear further away from it, as can be seen in the toy example depicted in Figure 3.1.

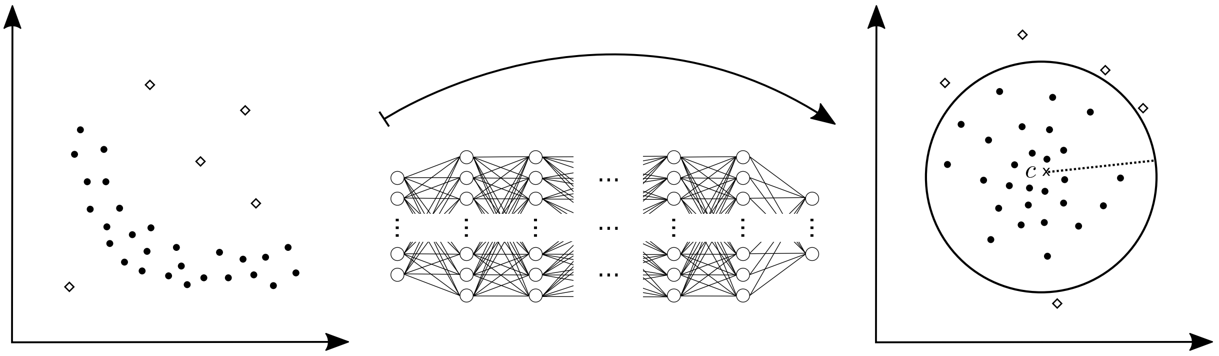


*Figure 3.1: DeepSAD teaches a neural network to transform data into a latent space and minimize the volume of a data-encompassing hypersphere centered around a predetermined centroid **c**. Reproduced from [16].*

Before DeepSAD's training can begin, a pretraining step is required, during which an autoencoder is trained on all available input data. One of DeepSAD's goals is to map input data onto a lower-dimensional latent space, in which the separation between normal and anomalous data can be achieved. To this end, DeepSAD and its predecessor Deep SVDD make use of the autoencoder's reconstruction goal, whose successful training ensures confidence in the encoder architecture's suitability for extracting the input data's most prominent information to the latent space between the encoder and decoder. DeepSAD goes on to use just the encoder as its main network architecture, discarding the decoder at this step, since reconstruction of the input is unnecessary.

The pretraining results are used in two more key ways. First, the encoder weights obtained from the autoencoder pretraining initialize DeepSAD's network for the main training phase. Second, we perform an initial forward pass through the encoder on all training samples, and the mean of these latent representations is set as the hypersphere center, $\mathbf{c}$. According to Ruff et al., this initialization method leads to faster convergence during the main training phase compared to using a randomly selected centroid. An alternative would be to compute $\mathbf{c}$ using only the labeled normal examples, which would prevent the center from being influenced by anomalous samples; however, this requires a sufficient number of labeled normal samples. Once defined, the hypersphere center $\mathbf{c}$ remains fixed, as allowing it to be optimized freely could, in the unsupervised case, lead to a hypersphere collapse-a trivial solution where the network learns to map all inputs directly onto the centroid $\mathbf{c}$.

In the main training step, DeepSAD's network is trained using SGD backpropagation. The unlabeled training data is used with the goal of minimizing a data-encompassing hypersphere. Since one of the preconditions of training was the significant prevalence of normal data over anomalies in the training set, normal samples collectively cluster more tightly around the centroid, while the rarer anomalous samples do not contribute as significantly to the optimization, resulting in them staying further from the hypersphere center. The labeled data includes binary class labels signifying their status as either normal or anomalous samples. Labeled anomalies are pushed away from the center by defining their optimization target as maximizing the distance between them and $\mathbf{c}$. Labeled normal samples are treated similarly to unlabeled samples, with the difference that DeepSAD includes a hyperparameter capable of controlling the proportion with which labeled and unlabeled data contribute to the overall optimization. The resulting network has learned to map normal data samples closer to $\mathbf{c}$ in the latent space and anomalies further away.

To infer if a previously unknown data sample is normal or anomalous, the sample is fed in a forward pass through the fully trained network. During inference, the centroid $\mathbf{c}$ needs to be known to calculate the geometric distance between the sample's latent representation and $\mathbf{c}$. This distance serves as an anomaly score, which correlates with the likelihood of the sample being anomalous. Due to differences in input data type, training success, and latent space dimensionality, the anomaly score's magnitude has to be judged on an individual basis for each trained network. This means that scores produced by one network, which signify normal data, may very well clearly indicate an anomaly for another network. The geometric distance between two points in space is a scalar analog value; therefore, post-processing of the score is necessary to achieve a binary classification of normal and anomalous if desired.

DeepSAD's full training and inference procedure is visualized in Figure 3.2, which gives a comprehensive overview of the dataflows, tunable hyperparameters, and individual steps involved.

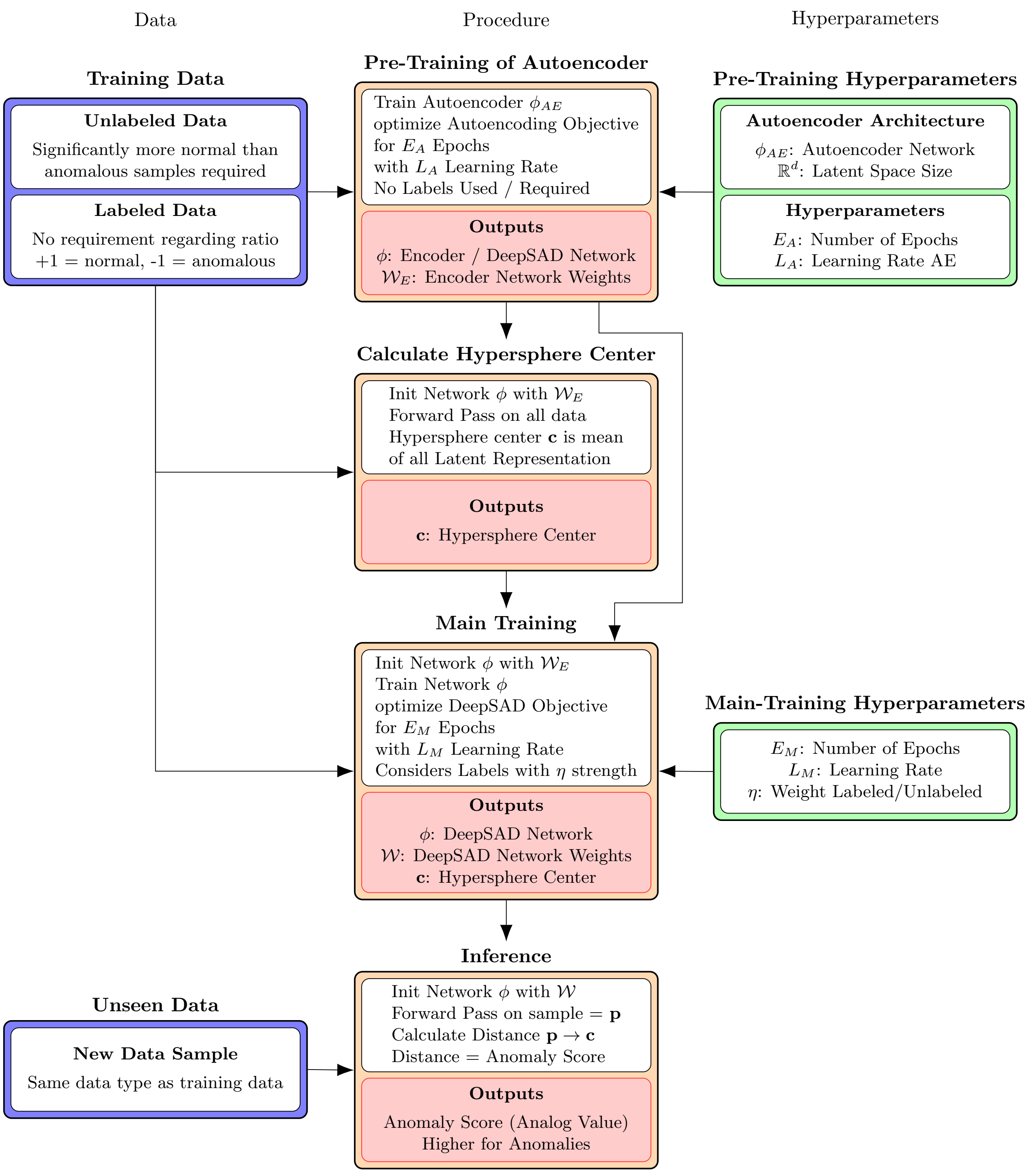

Data
Procedure
Hyperparameters
Training Data
Unlabeled Data
Significantly more normal than anomalous samples required
Labeled Data
No requirement regarding ratio
+1 = normal, -1 = anomalous
Pre-Training of Autoencoder
Train Autoencoder ϕ_{AE}
optimize Autoencoding Objective
for E_A Epochs
with L_A Learning Rate
No Labels Used / Required
Outputs
ϕ: Encoder / DeepSAD Network
$\mathcal{W}_E$: Encoder Network Weights
Pre-Training Hyperparameters
Autoencoder Architecture
ϕ_{AE}: Autoencoder Network
$\mathbb{R}^d$: Latent Space Size
Hyperparameters
E_A: Number of Epochs
L_A: Learning Rate AE
Calculate Hypersphere Center
Init Network ϕ with $\mathcal{W}_E$
Forward Pass on all data
Hypersphere center $\mathbf{c}$ is mean of all Latent Representation
Outputs
$\mathbf{c}$: Hypersphere Center
Main Training
Init Network ϕ with $\mathcal{W}_E$
Train Network ϕ
optimize DeepSAD Objective
for E_M Epochs
with L_M Learning Rate
Considers Labels with η strength
Outputs
ϕ: DeepSAD Network
$\mathcal{W}$: DeepSAD Network Weights
$\mathbf{c}$: Hypersphere Center
Main-Training Hyperparameters
E_M: Number of Epochs
L_M: Learning Rate
η: Weight Labeled/Unlabeled
Inference
Init Network ϕ with $\mathcal{W}$
Forward Pass on sample = $\mathbf{p}$
Calculate Distance $\mathbf{p} \to \mathbf{c}$
Distance = Anomaly Score
Outputs
Anomaly Score (Analog Value)
Higher for Anomalies
Unseen Data
New Data Sample
Same data type as training data

Figure 3.2: Overview of the DeepSAD workflow. Training starts with unlabeled data and optional labeled samples, which are used to pretrain an autoencoder, compute the hypersphere center, and then perform main training with adjustable weighting of labeled versus unlabeled data. During inference, new samples are encoded and their distance to the hypersphere center is used as an anomaly score, with larger distances indicating stronger anomalies.

3.2 Algorithm Details and Hyperparameters

Since DeepSAD is heavily based on its predecessor Deep SVDD [16], it is helpful to first understand Deep SVDD's optimization objective, so we start by explaining it here. For input space $\mathcal{X} \subseteq \mathbb{R}^D$, output space $\mathcal{Z} \subseteq \mathbb{R}^d$, and a neural network $\phi(\cdot; \mathcal{W}) : \mathcal{X} \to \mathcal{Z}$, where $\mathcal{W}$ depicts the neural network's weights with L layers $\{\mathbf{W}_1, \dots, \mathbf{W}_L\}$, n the number of unlabeled training samples $\{\mathbf{x}_1, \dots, \mathbf{x}_n\}$, $\mathbf{c}$ the center of the hypersphere in the latent space, Deep SVDD teaches the neural network to cluster normal data closely together in the latent space by defining its optimization objective as follows.

$$\min_{\mathcal{W}} \quad \frac{1}{n}\sum_{i=1}^{n} \|\phi(\mathbf{x}_i;\mathcal{W}) - \mathbf{c}\|^2 + \frac{\lambda}{2}\sum_{\ell=1}^{L} \|\mathbf{W}^{\ell}\|_F^2. \tag{3.1}$$

Deep SVDD is an unsupervised method that does not rely on labeled data to train the network to differentiate between normal and anomalous data. The first term of its optimization objective depicts the shrinking of the data-encompassing hypersphere around the given center $\mathbf{c}$. For each data sample $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, its geometric distance to $\mathbf{c}$ in the latent space produced by the neural network $\phi(\,\cdot\,;\mathcal{W})$ is minimized proportionally to the number of data samples n. The second term is a standard L2 regularization term, which prevents overfitting with hyperparameter $\lambda > 0$ and $\|\cdot\|_F$ denoting the Frobenius norm.

Ruff et al. argue that the pretraining step employing an autoencoder—originally introduced in Deep SVDD—not only allows a geometric interpretation of the method as minimum volume estimation, i.e., the shrinking of the data encompassing hypersphere, but also a probabilistic one as entropy minimization over the latent distribution. The autoencoding objective during pretraining implicitly maximizes the mutual information between the data and its latent representation, aligning the approach with the Infomax principle while encouraging a latent space with minimal entropy. This insight enabled Ruff et al. to introduce an additional term in DeepSAD's objective, beyond that of its predecessor Deep SVDD, which incorporates labeled data to better capture the characteristics of normal and anomalous data. They demonstrate that DeepSAD's objective effectively models the latent distribution of normal data as having low entropy, while that of anomalous data is characterized by higher entropy. In this framework, anomalies are interpreted as being generated from an infinite mixture of distributions that differ from the normal data distribution. The introduction of this aforementioned term in DeepSAD's objective allows it to learn in a semi-supervised way, which helps the model better position known normal samples near the hypersphere center and push known anomalies farther away, thereby enhancing its ability to differentiate between normal and anomalous data.

From Equation 3.1, it is easy to understand DeepSAD's optimization objective, seen in Equation 3.2, which additionally uses m number of labeled data samples $\{(\tilde{\mathbf{x}}_1, \tilde{y}_1), \ldots, (\tilde{\mathbf{x}}_m, \tilde{y}_1)\} \in \mathcal{X} \times \mathcal{Y}$ and $\mathcal{Y} = \{-1, +1\}$ for which $\tilde{y} = +1$ denotes normal and $\tilde{y} = -1$ anomalous samples, as well as a new hyperparameter $\eta > 0$, which can be used to balance the strength with which labeled and unlabeled samples contribute to the training.

The objective is

$$\min_{\mathcal{W}} \quad \frac{1}{n+m}\sum_{i=1}^{n} \|\phi(\mathbf{x}_i;\mathcal{W}) - \mathbf{c}\|^2 + \frac{\eta}{n+m}\sum_{j=1}^{m} \left(\|\phi(\tilde{\mathbf{x}}_j;\mathcal{W}) - \mathbf{c}\|^2\right)^{\tilde{y}_j} + \frac{\lambda}{2}\sum_{\ell=1}^{L} \|\mathbf{W}^{\ell}\|_F^2. \tag{3.2}$$

The first term of Equation 3.2 stays almost the same, differing only in its consideration of the introduced m labeled data samples for its proportionality. The second term is newly introduced to incorporate the labeled data samples with hyperparameter η's strength, by either minimizing or maximizing the distance between each sample's latent representation and $\mathbf{c}$, depending on that sample's label $\tilde{y}$. The standard L2 regularization is kept identical to Deep SVDD's optimization objective. It can also be observed that in the case of $m = 0$ labeled samples, DeepSAD falls back to Deep SVDD's optimization objective and can therefore be used in a completely unsupervised fashion as well.

Hyperparameters DeepSAD relies on several tunable hyperparameters that influence different stages of the algorithm. The most relevant ones are summarized and discussed below.

- **Network architecture ϕ**
  The encoder architecture determines the representational capacity of the model. Because DeepSAD builds on a pretraining autoencoder, the architecture must be expressive enough to reconstruct input data during pretraining, but also compact enough to support separation of normal and anomalous samples in the latent space. The choice of architecture is therefore data-dependent: convolutional encoders are often used for images, while fully connected encoders or other architectures may be more suitable for various data modalities. The architecture directly constrains which patterns the network can learn and thus strongly shapes the latent space structure.

- **Latent space dimensionality $\mathbb{R}^d$**
  The size of the latent bottleneck is a critical parameter. If $\mathbb{R}^d$ is too small, the network cannot encode all relevant information, leading to information loss and weak representations. If $\mathbb{R}^d$ is too large, the network risks overfitting by encoding irrelevant detail, while also increasing computational cost. These insights stem from autoencoder literature [28], but it is unclear whether they apply directly to DeepSAD: here, the autoencoder serves only for pretraining, and the encoder is subsequently fine-tuned with a different objective. Thus, the optimal choice of $\mathbb{R}^d$ may not coincide with the value that would be ideal for autoencoder reconstruction alone.

- **Label weighting η**
  The parameter η controls the relative contribution of labeled versus unlabeled data in the DeepSAD objective. With $\eta = 1$, both groups contribute equally (normalized by their sample counts). Larger values of η emphasize the labeled data, pulling labeled normal data closer to the center and pushing labeled anomalies further away. Smaller values emphasize the unlabeled data, effectively reducing the influence of labels. Its impact depends not only on its numerical value but also on the quantity and quality of available labels.

- **Learning rates L_A and L_M**
  Two learning rates are defined: L_A for the autoencoder pretraining and L_M for the main DeepSAD training. The learning rate sets the step size used during gradient descent updates and thereby controls the stability and speed of training. If it is too high, the optimization may diverge or oscillate; if too low, convergence becomes excessively slow and may get stuck in poor local minima. Schemes with adaptive learning rates, such as ADAM, may be applied to prevent poor choices.

- **Number of epochs E_A and E_M**
  The number of training epochs specifies how many full passes over the dataset are made in pretraining (E_A) and in the main DeepSAD training (E_M). More epochs allow the model to fit more closely to the training data, but also increase the risk of overfitting to noise or mislabeled samples. In practice, the effective number of epochs depends on dataset size, network architecture, and whether early stopping is applied.

- **Regularization rate λ**
  The rate of regularization, where λ has to be larger than 0 for regularization to take effect. A higher value decreases the chance of overfitting at the cost of model complexity.

4 Data and Preprocessing

Situations such as earthquakes, structural failures, and other emergencies that require rescue robots are fortunately rare. When these operations do occur, the primary focus is on the rapid and safe rescue of survivors rather than on data collection. Consequently, there is a scarcity of publicly available data from such scenarios. To improve any method, however, a large, diverse, and high-quality dataset is essential for comprehensive evaluation. This challenge is further compounded in our work, as we evaluate a training-based approach that imposes even higher demands on the data, especially requiring a great deal of diverse training samples, making it difficult to find a suitable dataset.

In this chapter, we outline the specific requirements we established for the data, describe the dataset selected for this task—including key statistics and notable features—and explain the preprocessing steps applied for training and evaluating the methods.

4.1 Data Requirements and Challenges

To ensure our chosen dataset meets the needs of reliable degradation quantification in subterranean rescue scenarios, we imposed the following requirements:

1. **Data Modalities:**
   The dataset must include LiDAR sensor data, since we decided to train and evaluate our method on what should be the most universally used sensor type in the given domain. To keep our method as generalized as possible, we chose to only require range-based point cloud data and neglect sensor-specific data such as intensity or reflectivity, though it may be of interest for future work. It is also desirable to have complementary visual data, such as camera images, for better context, manual verification, and understanding of the data.

2. **Context & Collection Method:**
   To mirror the real-world conditions of autonomous rescue robots, the data should originate from locations such as subterranean environments (tunnels, caves, collapsed structures), which closely reflect what would be encountered during rescue missions. Ideally, it should be captured from a ground-based, self-driving robot platform in motion instead of aerial, handheld, or stationary collection, to ensure similar circumstances to the target domain.

3. **Degradation Characteristics:**
   Because our goal is to quantify the degradation of LiDAR data encountered by rescue robots, the dataset must exhibit significant degradation of LiDAR returns from aerosols (i.e., dust or smoke particles), which should be the most frequent and challenging degradation encountered. This requirement is key to evaluating how well our method detects and measures the severity of such challenging conditions.

4. **Volume & Class Balance:**
   The dataset must be large enough to train deep learning models effectively. Since our semi-supervised approach depends on learning a robust model of "normal" data, the majority

of samples should be high-quality, degradation-free scans. Simultaneously, there must be a sufficient number of degraded (anomalous) scans to permit a comprehensive evaluation of quantification performance.

5. **Ground-Truth Labels:**
   Finally, to evaluate and tune our method, we need some form of ground truth indicating which scans are degraded. Obtaining reliable labels in this domain is challenging. Manual annotation of degradation levels is laborious and somewhat subjective. We address these challenges and issues of labeling data for evaluation next.

Quantitative benchmarking of degradation quantification requires a degradation label for every scan. Ideally, that label would be a continuous degradation score, although a binary label would still enable meaningful comparison. As the rest of this section shows, producing any reliable label is already challenging, and assigning meaningful analog scores may not be feasible at all. Compounding the problem, no public search-and-rescue (SAR) LiDAR data set offers such ground truth as far as we know. To understand the challenges around labeling LiDAR data degradation, we will look at what constitutes degradation in this context.

In Section 2.4, we discussed some internal and environmental error causes of LiDAR sensors, such as multi-return ambiguities or atmospheric scattering, respectively. While we are aware of research into singular failure modes [29] or research trying to model the totality of error sources occurring in other domains [30], there appears to be no such model for the search and rescue domain and its unique environmental circumstances. Although scientific consensus appears to be that airborne particles are the biggest contributor to degradation in SAR [31], we think that a more versatile definition is required to ensure confidence during critical SAR missions, which are often of a volatile nature. We are left with an ambiguous definition of what constitutes LiDAR point cloud degradation in the SAR domain.

We considered which types of objective measurements may be available to produce ground-truth labels, such as particulate matter sensors, LiDAR point clouds' inherent properties such as range-dropout rate and others, but we fear that using purely objective measures to label the data, would limit our learning based method to imitating the labels' sources instead of differentiating all possible degradation patterns from high quality data. Due to the incomplete error model in this domain, there may be novel or compound error sources that would not be captured using such an approach. As an example, we did observe dense smoke reflecting enough rays to produce phantom objects, which may fool SLAM algorithms. Such a case may even be labeled incorrectly as normal by one of the aforementioned objective measurement labeling options, if the surroundings do not already exhibit enough dispersed smoke particles.

To mitigate the aforementioned risks, we adopt a human-centric, binary labelling strategy. We judged analog and multi-level discrete rating scales to be too subjective for human consideration, which only left us with the simplistic, but hopefully more reliable, binary choice. We used two labeling approaches, producing two evaluation sets, whose motivation and details will be discussed in more detail in Section 4.3. Rationale for the exact labeling procedures requires knowledge of the actual dataset we ended up choosing, which we will present in the next section.

4.2 Dataset

Based on the previously discussed requirements and the challenges of obtaining reliable labels, we selected the "Multimodal Dataset from Harsh Sub-Terranean Environment with Aerosol Particles for Frontier Exploration" [32] for training and evaluation. This dataset comprises multimodal sensor data collected from a robotic platform navigating tunnels and rooms in a subterranean environment, an underground tunnel in Luleå, Sweden. Notably, some experiments incorporated an artificial smoke machine to simulate heavy degradation from aerosol particles, making the dataset particularly well-suited to our use case. A Pioneer 3-AT2 robotic platform, which can be seen in Figure 4.1(a), was used to mount a multitude of sensors that are described in Table 4.1 and whose mounting locations are depicted in Figure 4.1(b).

Table 4.1: On–board sensors recorded in the “Multimodal Dataset from Harsh Sub-Terranean Environment with Aerosol Particles for Frontier Exploration” dataset. Numbers match the labels in Fig. 4.1; only the most salient details are shown for quick reference.

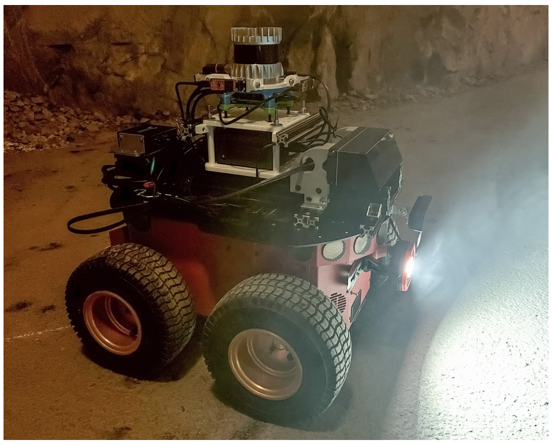
#	Sensor	Recorded Data	Key Specs
1	Spinning 3-D LiDAR *Ouster OS1-32*	3-D cloud, reflectivity	10 Hz, 32 ch, 360° × 42.4°, ≤ 120 m
2	mm-wave RADAR (×4) *TI IWR6843AoP*	4 × 60° RADAR point clouds	30 Hz, 60 GHz, 9 m max, 0.05 m res.
3	Solid-state LiDAR *Velodyne Velarray M1600*	Forward LiDAR cloud	10 Hz, 160 ch, 120° × 32°, 0.1–30 m
4	RGB-D / stereo cam *Luxonis OAK-D Pro*	Stereo b/w images, depth map	15 fps, 75 mm baseline, active IR 930 nm
5	LED flood-light *RS PRO WL28R*	Illumination for stereo camera	7 W, 650 lm (no data stream)
6	IMU *Pixhawk 2.1 Cube Orange*	Accel, gyro, mag, baro	190 Hz, 9-DoF, vibration-damped
7	On-board PC *Intel NUC i7*	Time-synced logging	Quad-core i7, 16 GB RAM, 500 GB SSD

We use data from the *Ouster OS1-32* LiDAR sensor, which was configured to capture 10 frames per second with a resolution of 32 vertical channels and 2048 measurements per channel. These settings yield equiangular measurements across a vertical field of view of 42.4° and a complete 360° horizontal field of view. Consequently, every LiDAR scan can generate up to 65,536 points. Each point contains the X, Y, and Z coordinates (in meters, with the sensor location as the origin) along with values for *range*, *intensity*, and *reflectivity*—typical metrics measured by LiDAR sensors. The datasets' point clouds are saved in a dense format, meaning each of the 65,536 measurements is present in the data, although fields for missing measurements contain zeroes.

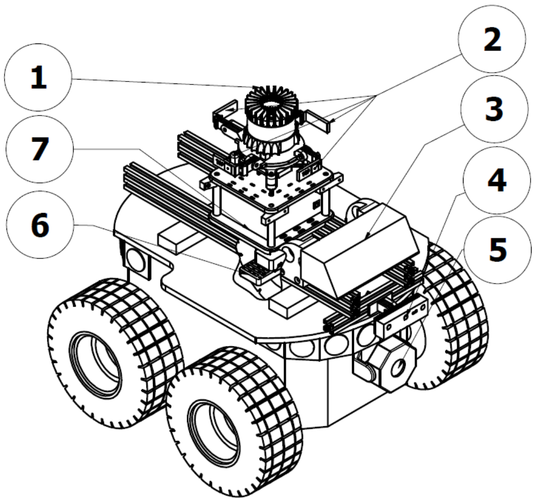
During the measurement campaign, a total of 14 experiments were conducted—10 prior to operating the artificial smoke machine (hereafter referred to as normal experiments) and 4 after it had already been running for some time (anomalous experiments). In 13 of these experiments, the sensor platform was in near-constant motion (either translating at roughly 1m/s or rotating), with only one anomalous experiment conducted while the platform remained stationary. Although this means we do not have two stationary experiments from the same exact position for a direct comparison between normal and anomalous conditions, the overall experiments are similar enough to allow for meaningful comparisons. In addition to the presence of water vapor from the smoke machine, the experiments vary in illumination conditions, the presence of humans on the measurement grounds, and additional static artifacts. For our purposes, only the artificial smoke is relevant; differences in lighting or incidental static objects do not affect our analysis. Regardless of illumination, the LiDAR sensor consistently produces comparable point clouds, and the presence of static objects does not influence our quantification of point cloud degradation.

In the anomalous experiments, the artificial smoke machine appears to have been running for some time before data collection began, as evidenced by both camera images and LiDAR data showing an even distribution of water vapor around the machine. The stationary experiment is particularly unique: the smoke machine was positioned very close to the sensor platform and was actively generating new, dense smoke, to the extent that the LiDAR registered the surface of the fresh water vapor as if it were a solid object.

Figures 4.2 and 4.3 show a representative depiction of the environment of the experiments as a camera image of the IR camera and the point cloud created by the OS1 LiDAR sensor at



(a) Pioneer 3-AT2 mobile base carrying the sensor tower. The four-wheel, skid-steered platform supports up to 30 kg payload and can negotiate rough terrain—providing the mobility required for subterranean data collection.



(b) Sensor layout and numbering. Components: 1 OS1-32 LiDAR, 2 mm-wave RADARs, 3 M1600 LiDAR, 4 OAK-D Pro camera, 5 LED flood-light, 6 IMU, 7 Intel NUC. See Table 4.1 for detailed specifications.

Figure 4.1: Robotic platform and sensor configuration used to record the dataset.

practically the same time.

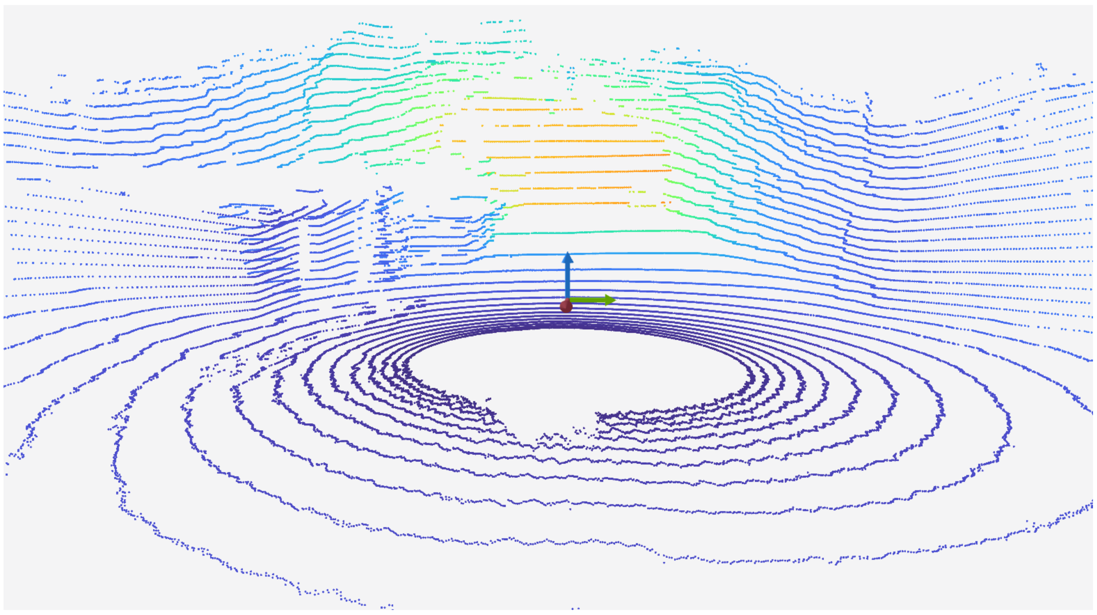


Figure 4.2: Screenshot of 3D rendering of an experiment's point cloud produced by the OS1-32 LiDAR sensor without smoke and with illumination (same frame and roughly same alignment as Figure 4.3). The point color corresponds to the measurement range, and the axis in the center of the figure marks the LiDAR's position.

Regarding the dataset volume, the 10 normal experiments ranged from 88.7 to 363.1 seconds, with an average duration of 157.65 seconds. At a capture rate of 10 frames per second, these experiments yield 15,765 non-degraded point clouds. In contrast, the 4 anomalous experiments, including one stationary experiment lasting 11.7 seconds and another extending to 62.1 seconds, averaged 47.33 seconds, resulting in 1,893 degraded point clouds. In total, the dataset comprises

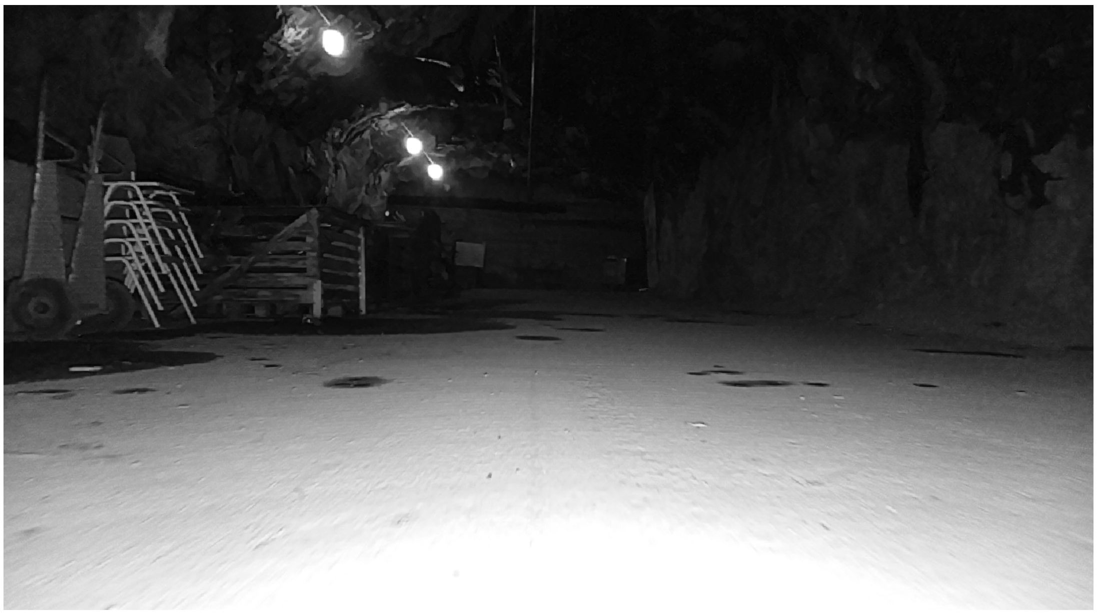


Figure 4.3: Screenshot of IR camera output of an experiment without smoke and with illumination (same frame and roughly same alignment as Figure 4.2).

17,658 point clouds, with approximately 89.28% classified as non-degraded (normal) and 10.72% as degraded (anomalous). The distribution of experimental data is visualized in Figure 4.4.

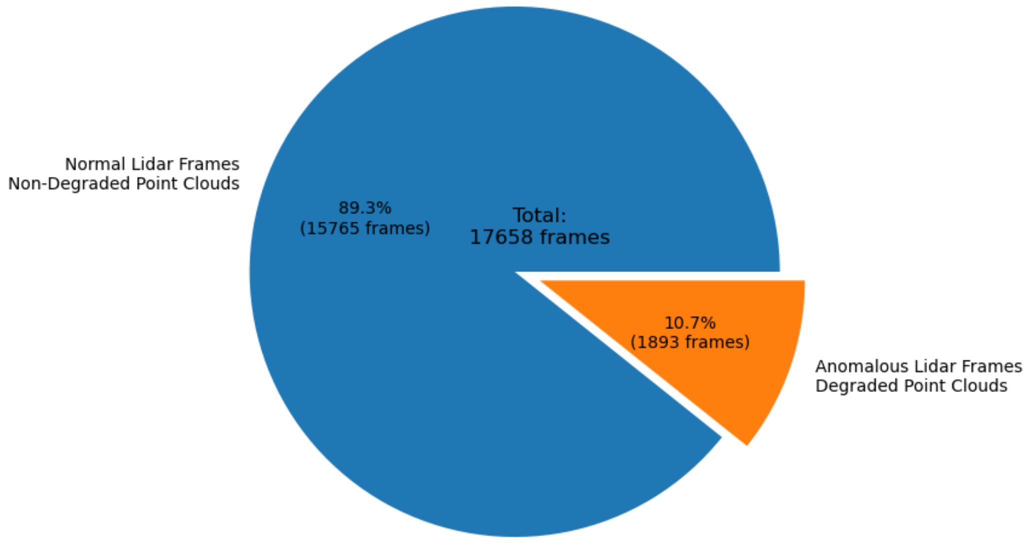


Figure 4.4: Pie chart visualizing the amount and distribution of normal and anomalous LiDAR frames (i.e., point clouds) in [32].

The artificial smoke introduces measurable changes that clearly separate the *anomalous* runs from the *normal* baseline. One change is a larger share of missing points per scan: smoke particles scatter or absorb the laser beam before it reaches a solid target, so the sensor reports an error instead of a distance. Figure 4.5 shows the resulting right–shift of the missing-point histogram, a known effect for LiDAR sensors in aerosol-filled environments. Another demonstrative effect is the appearance of many spurious returns very close to the sensor; these near-field points arise when back-scatter from the aerosol itself is mistaken for a surface echo. The box plot in Figure 4.6 confirms a pronounced increase in sub-50 cm hits under smoke, a range at which we do not expect any non-erroneous measurements. Both effects are consistent with the behaviour reported in [33].

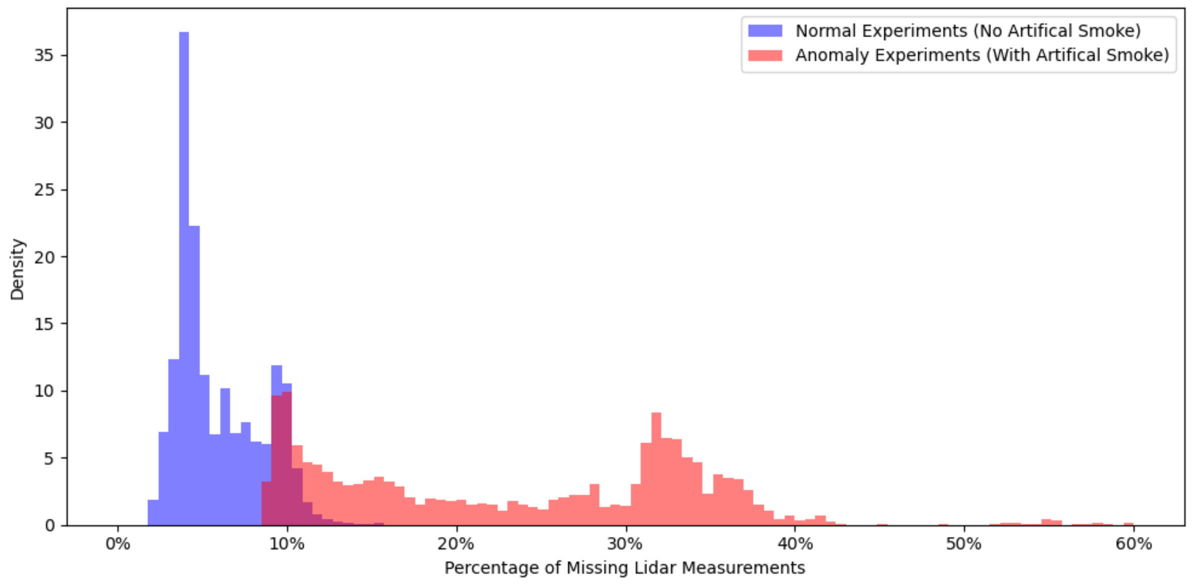


Figure 4.5: Density histogram showing the percentage of missing measurements per scan for normal experiments without degradation and anomalous experiments with artificial smoke introduced as degradation.

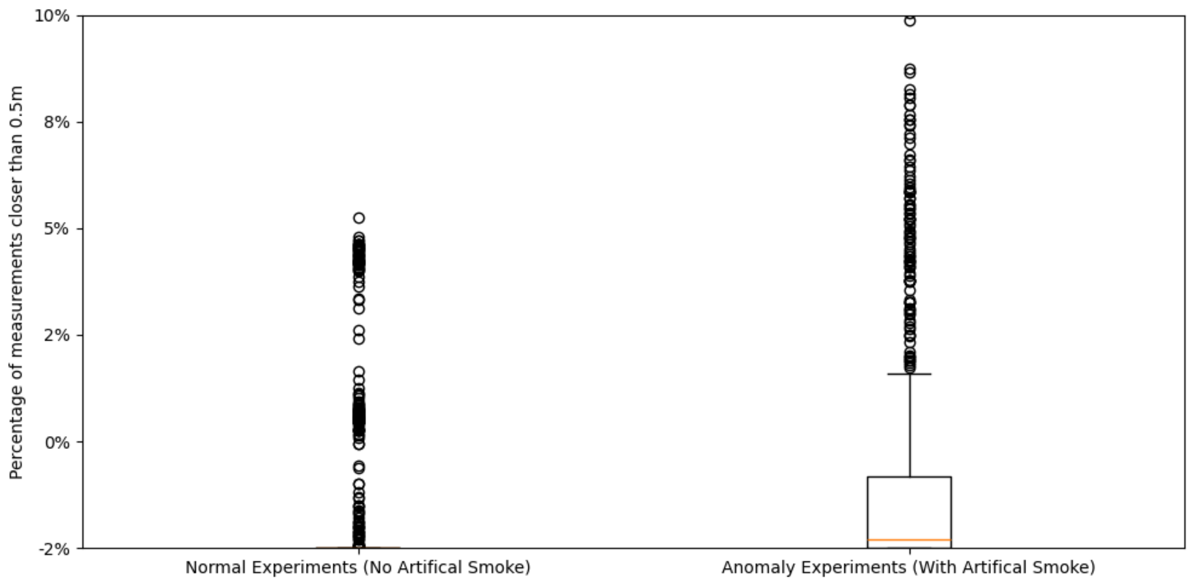


Figure 4.6: Box diagram depicting the percentage of measurements closer than 50 centimeters to the sensor for normal and anomalous experiments.

Taken together, the percentage of missing points and the proportion of near-sensor returns provide a concise indication of how strongly the smoke degrades our scans—capturing the two most prominent aerosol effects, drop-outs and back-scatter spikes. They do not, however, reveal the full error landscape discussed earlier (compound errors, temperature drift, multipath, . . .), so they should be read as an easily computed synopsis rather than an exhaustive measure of LiDAR quality. Next, we will discuss how the LiDAR scans were preprocessed before use and how we actually assigned ground-truth labels to each scan, so that we could train and evaluate our quantification degradation methods.

4.3 Preprocessing Steps and Labeling

As described in Section 3.1, the method under evaluation is data type agnostic and can be adapted to work with any kind of data by choosing a suitable autoencoder architecture. In our case, the input data are point clouds produced by a LiDAR sensor. Each point cloud contains up to 65,536 points, with each point represented by its X, Y, and Z coordinates. To tailor the DeepSAD architecture to this specific data type, we would need to design an autoencoder suitable for processing three-dimensional point clouds. Although autoencoders can be developed for various data types, [34] observed that over 60% of recent research on autoencoders focuses on two-dimensional image classification and reconstruction. Consequently, there is a more established understanding of autoencoder architectures for images compared to those for three-dimensional point clouds.

For this reason and to simplify the architecture, we converted the point clouds into two-dimensional grayscale images using a spherical projection. This approach—proven successful in related work [27]—encodes each LiDAR measurement as a single pixel, where the pixel's grayscale value is determined by the reciprocal range, calculated as $v = \frac{1}{\sqrt{X^2+Y^2+Z^2}}$. Given the LiDAR sensor's configuration, the resulting images have a resolution of 2048 pixels in width and 32 pixels in height. Missing measurements in the point cloud are mapped to pixels with a brightness value of $v = 0$.

To create this mapping, we leveraged the available measurement indices and channel information inherent in the dense point clouds, which are ordered from 0 to 65,535 in a horizontally ascending, channel-by-channel manner. For sparse point clouds without such indices, one would need to rely on the pitch and yaw angles relative to the sensor's origin to correctly map each point to its corresponding pixel, although this often leads to ambiguous mappings due to numerical errors in angle estimation.

Figure 4.7 displays two examples of LiDAR point cloud projections to aid in the reader's understanding. Although the original point clouds were converted into grayscale images with a resolution of 2048×32 pixels, these raw images can be challenging to interpret. To enhance human readability, we applied the viridis colormap and vertically stretched the images so that each measurement occupies multiple pixels in height. The projection in (a) is derived from a scan without artificial smoke—and therefore minimal degradation—while the projection in (b) comes from an experiment where artificial smoke introduced significant degradation.

The remaining challenge was labeling a large enough portion of the dataset in a reasonably accurate manner, whose difficulties and general approach we described in Section 4.1. Since, to our knowledge, neither our chosen dataset nor any other publicly available one provides objective labels for LiDAR data degradation in the SAR domain, we had to define our own labeling approach. With objective measures of degradation unavailable, we explored alternative labeling methods—such as using the statistical properties like the number of missing measurements per point cloud or the higher incidence of erroneous measurements near the sensor we described in Section 4.2. Ultimately, we were concerned that these statistical approaches might lead the method to simply mimic the statistical evaluation rather than to quantify degradation in a generalized and robust manner. After considering these options, we decided to label all point clouds from experiments with artificial smoke as anomalies, while point clouds from experiments without smoke were labeled as normal data. This labeling strategy—based on the presence or absence of smoke—is fundamentally an environmental indicator, independent of the intrinsic data properties recorded during the experiments.

The simplicity of this labeling approach has both advantages and disadvantages. On the pos-

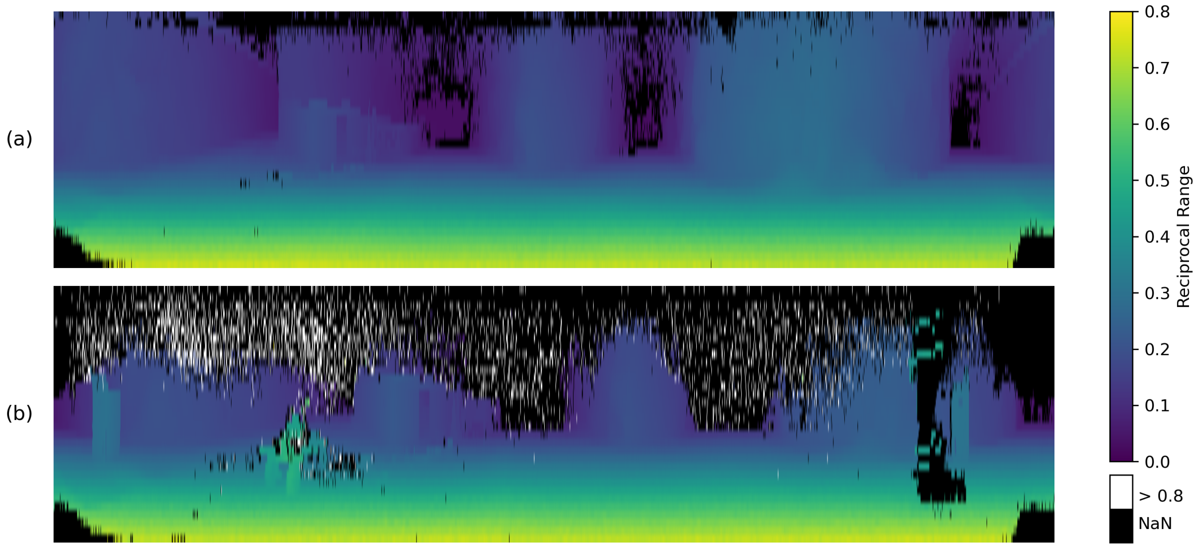


Figure 4.7: Two-dimensional projections of two point clouds, (a) from an experiment without degradation and (b) from an experiment with artificial smoke as degradation. To aid the reader's perception, the images are vertically stretched, and a colormap has been applied to the pixels' reciprocal range values, while the actual training data is grayscale.

itive side, it is easy to implement and creates a clear distinction between normal and anomalous data. However, its simplicity is also its drawback: some point clouds from experiments with artificial smoke do not exhibit perceptible degradation, yet they are still labeled as anomalies. The reason for this is that during the three non-static anomalous experiments, the sensor platform starts recording in a tunnel roughly 20 meters from the smoke machine's location. It starts by approaching the smoke machine, navigates close to the machine for some time, and then leaves its perimeter once again. Since the artificical smoke's density is far larger near the machine it originates from, the time the sensor platform spent close to it produced highly degraded point clouds, whereas the beginnings and ends of the anomalous experiments capture point clouds that are subjectively not degraded and appear similar to those from the normal experiments. This effect is clearly illustrated by the degradation indicators which we talked about earlier–the proportion of missing points and the amount of erroneous points close to the sensor per point cloud–as can be seen in Figure 4.8.

Afraid that the incorrectly labeled data may negatively impact DeepSAD's semi-supervised training, we chose to manually remove the anomalous labels from the beginning and end of the anomalous experiments, for training purposes. This refinement gave us more confidence in the training signal but reduced the number of labeled anomalies. For evaluation, we therefore report results under both schemes:

1. **Experiment-based labels:** All scans from anomalous experiments are marked anomalous, including border cases. This yields conservative performance metrics that reflect real-world label noise.

2. **Manually-defined labels:** Only unequivocally degraded scans are marked anomalous, producing near-ideal separation in a lot of cases.

Under both evaluation schemes, all frames from normal experiments were marked as normal, since they appear to have produced high-quality data throughout. A visualization of how the two evaluation schemes measure up in terms of the number of samples per class can be seen in Figure 4.9.

By evaluating and comparing both approaches, we hope to demonstrate a more thorough performance investigation than with only one of the two labeling schemes.

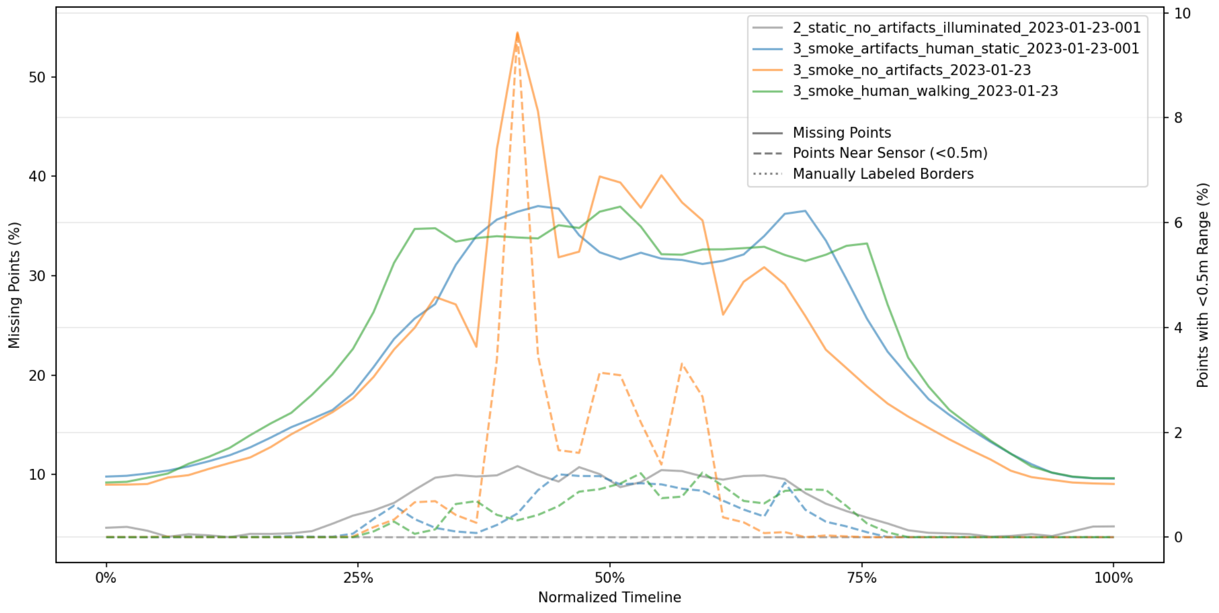


Figure 4.8: Missing points and points with a measured range smaller than 50cm per point cloud over a normalized timeline of the individual experiments. This illustrates the rise, plateau, and fall of degradation intensity during the anomalous experiments, owing to the spatial proximity between the LiDAR sensor and the degradation source (smoke machine). One of the normal experiments (without artificial smoke) is included as a baseline in gray.

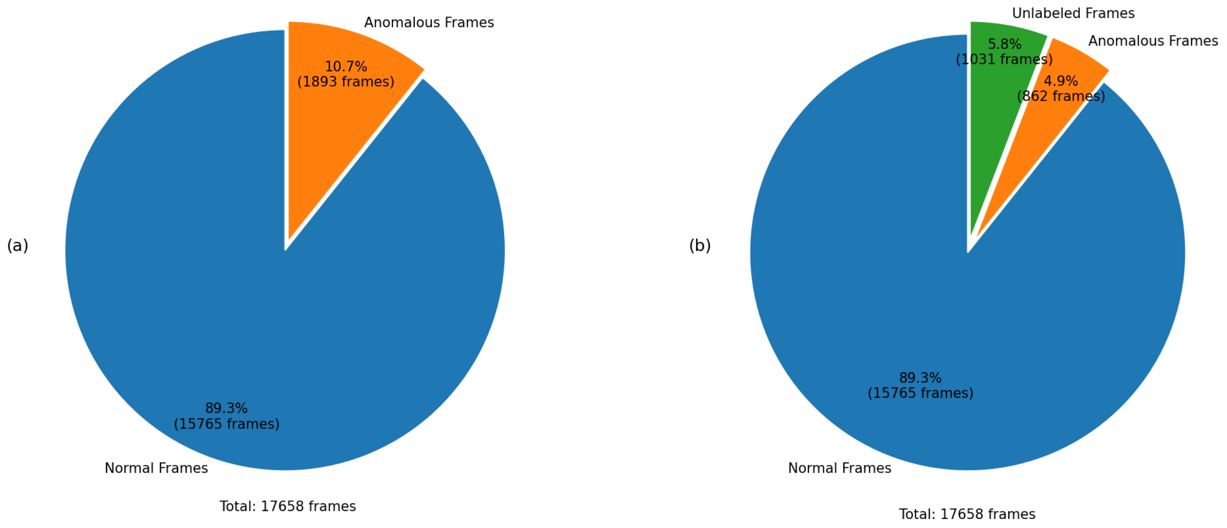


Figure 4.9: Pie charts visualizing the number of normal and anomalous labels applied to the dataset for (a) experiment-based labeling scheme and (b) manually-defined labeling scheme. A large part of the experiment-based anomalous labels had to be removed for the manually-defined scheme, since, subjectively, they were either clearly or possibly not degraded.

5 Experimental Setup

We built our experiments starting from the official DeepSAD PyTorch implementation and evaluation framework, available at `https://github.com/lukasruff/Deep-SAD-PyTorch`. This codebase provides routines for loading standard datasets, training DeepSAD and several baseline models, and evaluating their performance.

In the following sections, we detail our adaptations to this framework:

- Data integration: preprocessing and loading the dataset introduced in Chapter 4.
- Model architecture: configuring DeepSAD's encoder to match our point cloud input format, contrasting two distinct neural network architectures to investigate their impact on the method's output.
- Training & evaluation: training DeepSAD alongside two classical baselines—Isolation Forest and One-class SVM (OCSVM)—and comparing their degradation-quantification performance.
- Experimental environment: the hardware and software stack used, with typical training and inference runtimes.

Together, these components define the full experimental pipeline, from data loading, preprocessing, method training, to the evaluation and comparison of methods.

5.1 Framework & Data Preparation

DeepSAD's PyTorch implementation—our starting point—includes implementations for training on standardized datasets such as MNIST, CIFAR-10, and datasets from *ODDS Library* [35]. The framework can train and test DeepSAD as well as a number of baseline algorithms, namely SSAD, OCSVM, Isolation Forest, KDE, and SemiDGM, with the loaded data and evaluate their performance by calculating the Receiver Operating Characteristic (ROC) and its Area Under the Curve (AUC) for all given algorithms. We adapted this implementation, which was originally developed for Python 3.7, to work with Python 3.12 and changed or added functionality. We allowed loading data from our chosen dataset, added DeepSAD models that work with the LiDAR projections datatype, added more evaluation methods, and an inference module.

The raw SubTER dataset is provided as one ROS bag file per experiment, each containing a dense 3D point cloud from the Ouster OS1-32 LiDAR. To streamline training and avoid repeated heavy computation, we project these point clouds offline into 2D "range images" as described in Section 4.3 and export them to files as NumPy arrays. Storing precomputed projections allows rapid data loading during training and evaluation. Many modern LiDARs can be configured to output range images directly, which would bypass the need for post-hoc projection. When available, such native range-image streams can further simplify preprocessing or even allow skipping this step completely.

We extended the DeepSAD framework's PyTorch `DataLoader` by implementing a custom `Dataset` class that ingests our precomputed NumPy range-image files and attaches appropriate evaluation labels. Each experiment's frames are stored as individual `.npy` files with the numpy array shape (Number of Frames, H, W), containing the point clouds' reciprocal range values. Our `Dataset` initializer scans a directory of these files, loads the NumPy arrays from file into memory, transforms them into PyTorch tensors, and assigns evaluation and training labels accordingly.

The first labeling scheme, called *experiment-based labels*, assigns

$$y_{\text{exp}} = \begin{cases} -1 & \text{if the filename contains “smoke”, signifying anomalous/degraded data,} \\ +1 & \text{otherwise, signifying normal data.} \end{cases}$$

At load time, any file with "smoke" in its name is treated as anomalous (label -1), and all others (normal experiments) are labeled $+1$.

To obtain a second source of ground truth, we also support *manually-defined labels.* A companion JSON file specifies a start and end frame index for each of the four smoke experiments—defining the interval of unequivocal degradation. During loading, the second label y_{man} is assigned as follows:

$$y_{\text{man}} = \begin{cases} -1 & \text{Frames within the manually selected window from smoke experiments} \\ +1 & \text{All frames from non-smoke experiments} \\ 0 & \text{Frames outside the manually selected window from smoke experiments} \end{cases}$$

We pass instances of this `Dataset` to PyTorch's `DataLoader`, enabling batch sampling, shuffling, and multi-worker loading. The dataloader returns the preprocessed LiDAR projection, both evaluation labels, and a semi-supervised training label.

To control the supervision of DeepSAD's training, our custom PyTorch `Dataset` accepts two integer parameters, `num_labelled_normal` and `num_labelled_anomalous`, which specify how many samples of each class should retain their labels during training. We begin with the manually-defined evaluation labels, so as not to use mislabeled anomalous frames for the semi-supervision. Then, we randomly un-label (set to 0) enough samples of each class until exactly `num_labelled_normal` normals and `num_labelled_anomalous` anomalies remain labeled.

To obtain robust performance estimates on our relatively small dataset, we implement k-fold cross-validation. A single integer parameter, `num_folds`, controls the number of splits. We use scikit-learn's `KFold` (from `sklearn.model_selection`) with `shuffle=True` and a fixed random seed to partition each experiment's frames into `num_folds` disjoint folds. Training then proceeds across k rounds, each time training on $(k-1)/k$ of the data and evaluating on the remaining $1/k$. In our experiments, we set `num_folds=5`, yielding an 80/20 train/evaluation split per fold.

For inference (i.e., model validation on held-out experiments), we provide a second `Dataset` class that loads a single experiment's NumPy file (no k-fold splitting), does not assign any labels to the frames, nor does it shuffle frames, preserving temporal order. This setup enables seamless, frame-by-frame scoring of complete runs—crucial for analyzing degradation dynamics over an entire experiment.

5.2 Model Configuration

Since the neural network architecture trained in the DeepSAD method is not fixed as described in Section 3.2 but rather chosen based on the input data, we also had to choose an autoencoder architecture befitting our preprocessed LiDAR data projections. Since [27] reported success in training DeepSAD on similar data, we first adapted the network architecture utilized by them for our use case, which is based on the simple and well-understood LeNet architecture [36]. Additionally, we were interested in evaluating the importance and impact of a well-suited network architecture for DeepSAD's performance and therefore designed a second network architecture, henceforth referred to as "efficient architecture", to incorporate a few modern techniques, befitting our use case.

The LeNet-inspired autoencoder can be split into an encoder network (Figure 5.1) and a decoder network (Figure 5.2) with a latent space in between the two parts. Such an arrangement is typical for autoencoder architectures, as we discussed in Section 2.3. The encoder network is simultaneously DeepSAD's main training architecture, which is used to infer the degradation quantification in our use case, once trained.

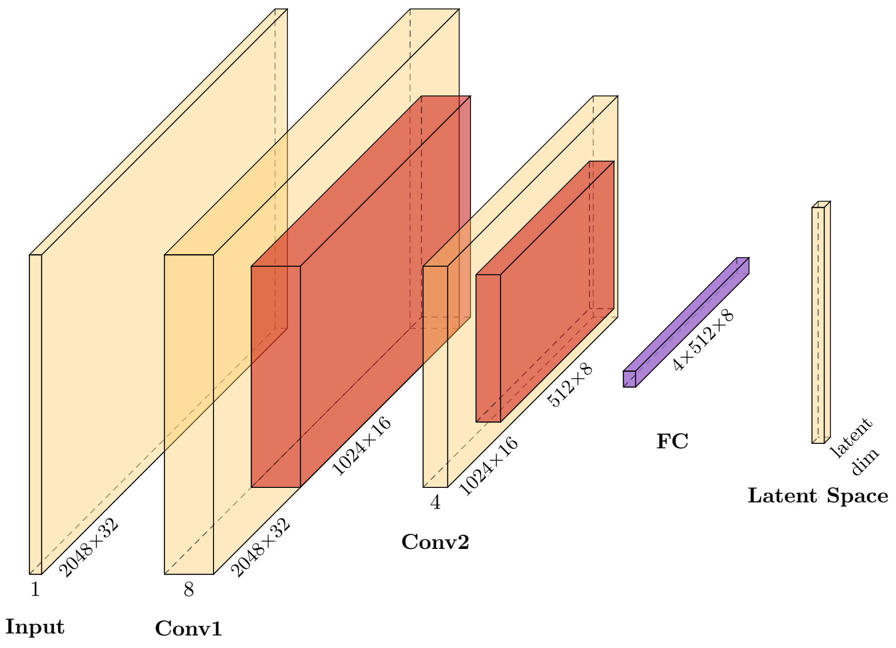


Figure 5.1: Architecture of the LeNet-inspired encoder. The input is a LiDAR range image of size $1 \times 2048 \times 32$ *(channels × width × height). The first block (Conv1) applies a* 5×5 *convolution with 8 output channels, followed by batch normalization, LeakyReLU activation, and* 2×2 *max pooling, resulting in a feature map of size* $8 \times 1024 \times 16$*. The second block (Conv2) applies another* 5×5 *convolution with 4 output channels, again followed by normalization, activation, and* 2×2 *max pooling, producing* $4 \times 512 \times 8$*. This feature map is flattened and passed through a fully connected (FC) layer of size* $4 \cdot 512 \cdot 8 = 16384$*, which maps to the latent space of dimensionality* d*, where* d *is a tunable hyperparameter* ($32 \leq d \leq 1024$ *in our experiments). The latent space serves as the compact representation used by DeepSAD for anomaly detection.*

The LeNet-inspired encoder network (see Figure 5.1) is a compact convolutional neural network that reduces image data into a lower-dimensional latent space. It consists of two stages of convolution, normalization, non-linear activation, and pooling, followed by a dense layer that defines the latent representation. Conceptually, the convolutional layers learn small filters that detect visual patterns in the input (such as edges or textures). Batch normalization ensures that these learned signals remain numerically stable during training, while a LeakyReLU activation introduces non-linearity, allowing the network to capture more complex relationships. Pooling operations then downsample the feature maps, which reduces the spatial size of the data and emphasizes the most important features. Finally, a dense layer transforms the extracted feature maps into the latent space.

The decoder network (see Figure 5.2) mirrors the encoder and reconstructs the input from its latent representation. A dense layer first expands the latent vector into a feature map of shape $4 \times 512 \times 8$, which is then upsampled and refined in two successive stages. Each stage consists of an interpolation step that doubles the spatial resolution, followed by a transpose convolution that learns how to add structural detail. The first stage operates on 4 channels, and the second on 8 channels, with the final transpose convolution reducing the output to a single channel. The

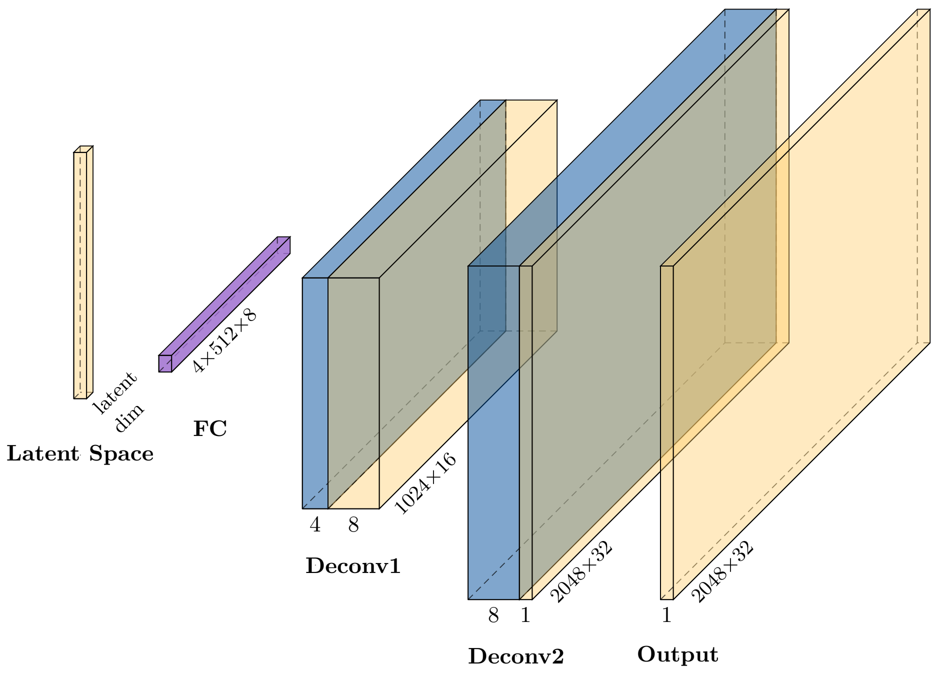


Figure 5.2: Architecture of the LeNet-inspired decoder. The input is a latent vector of dimension d, where d is the tunable latent space size ($32 \leq d \leq 1024$ in our experiments). A fully connected (FC) layer first expands this vector into a feature map of size $4 \times 512 \times 8$ (channels × width × height). The first upsampling stage applies interpolation with scale factor 2, followed by a transpose convolution with 8 output channels, batch normalization, and LeakyReLU activation, yielding $8 \times 1024 \times 16$. The second stage again upsamples by a factor of 2 and applies a transpose convolution, reducing the channels to 1. This produces the reconstructed output of size $1 \times 2048 \times 32$, which matches the original input dimensionality required for the autoencoding objective.

result is a reconstructed output of size $1 \times 2048 \times 32$, matching the original input dimensionality required for the autoencoding objective.

Even though the LeNet-inspired encoder proved capable of achieving our degradation quantification objective in initial experiments, we identified several shortcomings that motivated the design of a second, more efficient architecture. The most important issue concerns the shape of the CNN's receptive field (RF), which describes the region of the input that influences a single output activation. Its size and aspect ratio determine which structures the network can effectively capture: if the RF is too small, larger patterns cannot be detected, while an excessively large RF may hinder the network from learning to recognize fine details. For standard image data, the RF is often expressed as a symmetric $n \times n$ region, but in principle it can be computed independently per axis.

The RF shape's issue arises from the fact that spinning multi-beam LiDAR oftentimes produce point clouds possessing dense horizontal but limited vertical resolution. In our case, this results in a pixel-per-degree resolution of approximately $5.69\,{}^{pixel}/_{deg}$ vertically and $1.01\,{}^{pixel}/_{deg}$ horizontally. Consequently, the LeNet-inspired encoder's calculated receptive field of 16×16 pixels translates to an angular size of $15.88° \times 2.81°$, which is highly rectangular in angular space. Such a mismatch risks limiting the network's ability to capture degradation patterns that extend differently across the two axes.

To adjust for this, we decided to modify the network architecture and included further modifications to improve the method's performance. The encoder (see Figure 5.3) follows the same general idea as the LeNet-inspired encoder, but incorporates the following modifications:

- **Non-square convolution kernels.** Depthwise-separable convolutions with kernel size

3×17 are used instead of square kernels, resulting in an RF of 10×52 pixels, corresponding to 9.93° × 9.14°, substantially more balanced than the LeNet-inspired network's RF.

- **Circular padding along azimuth.** The horizontal axis is circularly padded to respect the wrap-around of 360° LiDAR data, preventing artificial seams at the image boundaries.
- **Aggressive horizontal pooling.** A 1×4 pooling operation is applied early in the network, which reduces the over-sampled horizontal resolution (2048 px to 512 px) while keeping vertical detail intact.
- **Depthwise-separable convolutions with channel shuffle.** Inspired by MobileNet [37] and ShuffleNet [38], this reduces the number of parameters and computations while retaining representational capacity, making the network more suitable for embedded platforms, while simultaneously allowing more learnable channels without increasing computational demand.
- **Max pooling.** Standard max pooling is used instead of average pooling, since it preserves sharp activations that are often indicative of localized degradation.
- **Channel compression before latent mapping.** After feature extraction, a 1×1 convolution reduces the number of channels before flattening, which lowers the parameter count of the final fully connected layer without sacrificing feature richness.

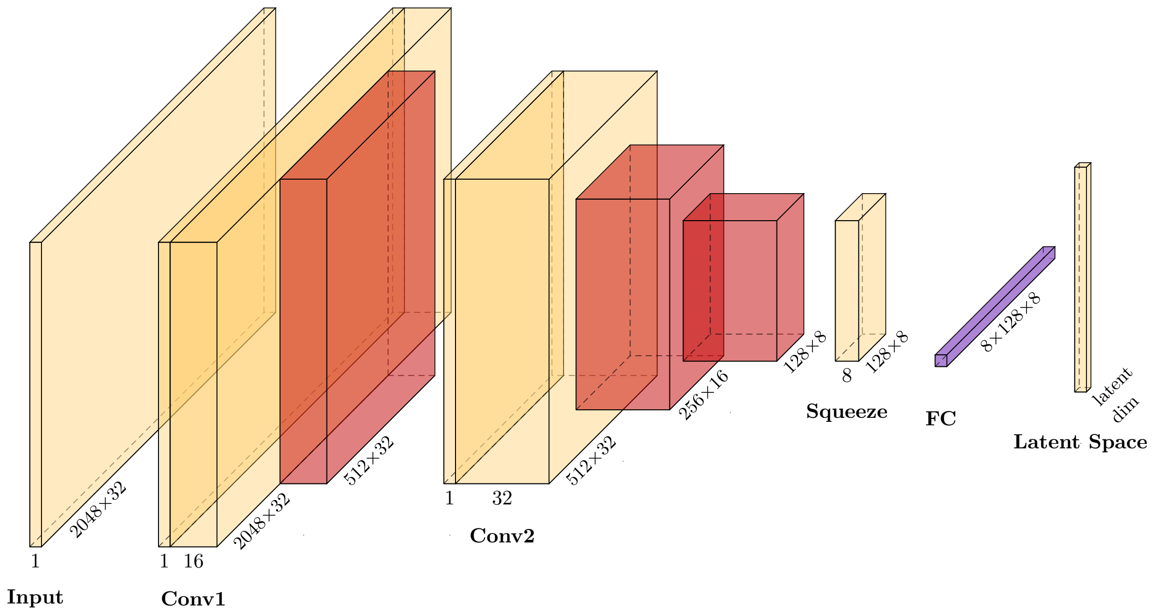


*Figure 5.3: Architecture of the Efficient encoder. The input is a LiDAR range image of size $1 \times 2048 \times 32$ (channels × width × height). The first block (**Conv1**) applies a depthwise–separable 3×17 convolution with circular padding along the azimuth, followed by batch normalization, LeakyReLU, and an aggressive horizontal pooling step, producing an intermediate representation of $16 \times 512 \times 32$. The second block (**Conv2**) applies another depthwise–separable convolution with channel shuffle, followed by two stages of 2×2 max pooling, yielding $32 \times 128 \times 8$. A 1×1 convolution (**Squeeze**) then reduces the channel dimension to 8, producing $8 \times 128 \times 8$. Finally, a fully connected layer (**FC**) flattens this feature map and projects it into the latent space of size d, where d is a tunable hyperparameter ($32 \leq d \leq 1024$ in our experiments).*

The decoder (see Figure 5.4) mirrors the encoder's structure but introduces changes to improve reconstruction stability:

- **Nearest-neighbor upsampling followed by convolution.** Instead of relying solely on transposed convolutions, each upsampling stage first enlarges the feature map using parameter-free nearest-neighbor interpolation, followed by a depthwise-separable convolution. This strategy reduces the risk of checkerboard artifacts while still allowing the network to learn fine detail.
- **Asymmetric upsampling schedule.** Horizontal resolution is restored more aggressively (e.g., scale factor 1×4) to reflect the anisotropic downsampling performed in the encoder.
- **Final convolution with circular padding.** The output is generated using a (3×17) convolution with circular padding along the azimuth, similar to the new encoder, ensuring consistent treatment of the 360° LiDAR input.

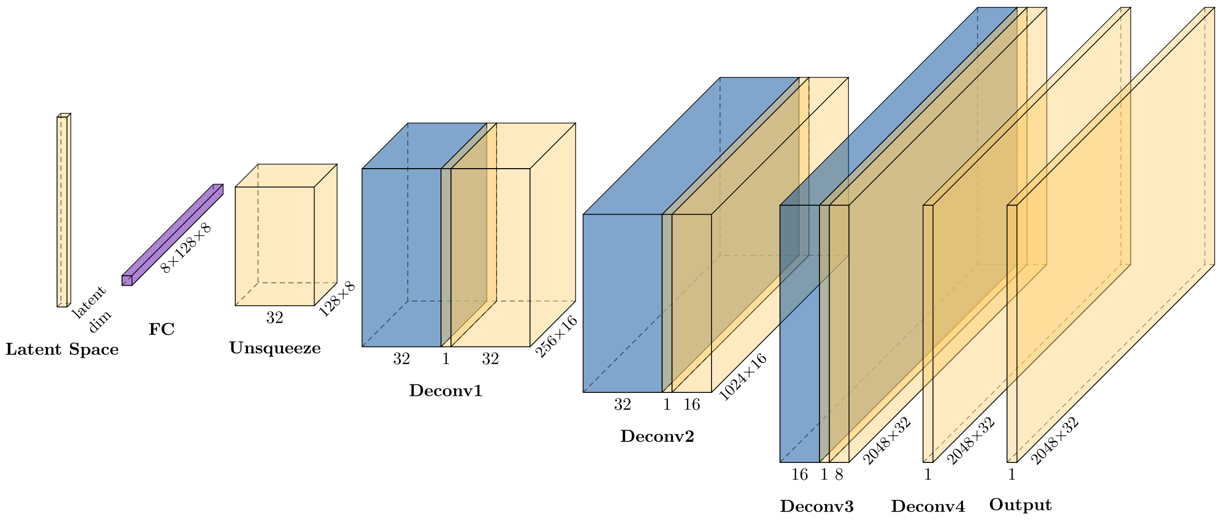


*Figure 5.4: Architecture of the Efficient decoder. The input is a latent vector of dimension d. A fully connected layer first expands this into a feature map of size $8 \times 128 \times 8$, followed by a 1×1 convolution (**Unsqueeze**) to increase the channel count to 32. The following three blocks (**Deconv1–3**) each consist of nearest-neighbor upsampling and a depthwise–separable convolution: **Deconv1** doubles both axes ($32 \times 256 \times 16$), **Deconv2** restores horizontal resolution more aggressively with a 1×4 upsampling ($16 \times 1024 \times 16$), and **Deconv3** doubles both axes again to $8 \times 2048 \times 32$. The final block (**Deconv4**) applies a (3×17) convolution with circular padding along the azimuth, reducing the channels to 1 and producing the reconstructed output of size $1 \times 2048 \times 32$, which matches the original input dimensionality.*

To compare the computational efficiency of the two architectures, we show the number of trainable parameters and the number of multiply–accumulate operations (MACs) for different latent space sizes used in our experiments in Table 5.1. Even though the efficient architecture employs more layers and channels, which allows the network to learn to recognize more types of patterns when compared to the LeNet-inspired one, the encoders' MACs are quite similar. The more complex decoder design of the efficient network appears to contribute a lot more MACs, which leads to longer pretraining times, which we report in Section 5.4.

Encoders (DeepSAD Networks)				
	Parameters		MACs	
Latent $\mathbb{R}^d$	**LeNet**	**Efficient**	**LeNet**	**Efficient**
32	0.53	0.26	27.92	29.82
64	1.05	0.53	28.44	30.08
128	2.10	1.05	29.49	30.61
256	4.20	2.10	31.59	31.65
512	8.39	4.20	35.78	33.75
768	12.58	6.29	39.98	35.85
1024	16.78	8.39	44.17	37.95
Autoencoders (Encoder plus Decoder)				
	Parameters		**MACs**	
Latent $\mathbb{R}^d$	**LeNet**	**Efficient**	**LeNet**	**Efficient**
32	1.05	0.53	54.95	168.49
64	2.10	1.06	56.00	169.02
128	4.20	2.11	58.10	170.07
256	8.39	4.20	62.29	172.16
512	16.78	8.40	70.68	176.36
768	25.17	12.59	79.07	180.55
1024	33.56	16.79	87.46	184.75

Table 5.1: Comparison of parameter count and MACs (repored in millions) for DeepSAD LeNet-inspired and DeepSAD Efficient encoder and pretraining autoencoder (encoder plus decoder) networks across different latent space sizes.

5.3 Baseline Methods & Evaluation Metrics

To contextualize the performance of DeepSAD, we compare against two widely used baselines: Isolation Forest and OCSVM. Both are included in the original DeepSAD codebase and associated paper, and they represent well-understood yet conceptually distinct families of anomaly detection. Together, these baselines provide complementary perspectives: raw input tree-based partitioning (Isolation Forest) and dimensionality-reduced kernel-based boundary learning (OCSVM), offering a broad and well-established basis for comparison.

Isolation Forest is an ensemble method for anomaly detection that builds on the principle that anomalies are easier to separate from the rest of the data. It constructs many binary decision trees, each by recursively splitting the data at randomly chosen features and thresholds. In this process, the "training" step consists of building the forest of trees: each tree captures different random partitions of the input space, and together they form a diverse set of perspectives on how easily individual samples can be isolated.

Once trained, the method assigns an anomaly score to new samples by measuring their average path length through the trees. Normal samples, being surrounded by other similar samples, typically require many recursive splits and thus end up deep in the trees. Anomalies, by contrast, stand out in one or more features, which means they can be separated much earlier and end up closer to the root. The shorter the average path length, the more anomalous the sample is considered. This makes Isolation Forest highly scalable and robust: training is efficient, and the resulting model is fast to apply to new data. In our setup, we apply Isolation Forest directly to the LiDAR input representation, providing a strong non-neural baseline for comparison against DeepSAD.

OCSVM takes a very different approach by learning a flexible boundary around normal samples. It assumes all training data to be normal, with the goal of enclosing the majority of these samples in such a way that new points lying outside this boundary can be identified as anomalies. The boundary itself is learned using the support vector machine framework. In essence, OCSVM looks for a hyperplane in some feature space that maximizes the separation between the bulk of the data and the origin. To make this possible, even when the normal data has a complex, curved shape, OCSVM uses a kernel function such as the radial basis function (RBF). The kernel implicitly maps the input data into a higher-dimensional space, where the cluster of normal samples becomes easier to separate with a simple hyperplane. When this separation is mapped back to the original input space, it corresponds to a flexible, nonlinear boundary that can adapt to the structure of the data.

During training, the algorithm balances two competing objectives: capturing as many of the normal samples as possible inside the boundary, while keeping the region compact enough to exclude potential outliers. Once this boundary is established, applying OCSVM is straightforward — any new data point is checked against the learned boundary, with points inside considered normal and those outside flagged as anomalous.

In our setting, the raw input dimensionality (2048×32 per frame) is too high for a direct OCSVM fit, so we reuse the autoencoder's encoder from DeepSAD's pretraining as a learned dimensionality reduction (to the same latent size as DeepSAD) to allow OCSVM training on this latent space. The dimensionality reduction step is always performed with the corresponding DeepSAD encoder and its autoencoder pretraining weights that match the evaluated setting (i.e., same latent size and backbone).

We adapted the baseline implementations to our data loader and input format and added support for multiple evaluation targets per frame (two labels per data point), reporting both results per experiment. Both baselines, like DeepSAD, output continuous anomaly scores, which allows us to evaluate them directly without committing to a fixed threshold.

Evaluation Metrics As discussed in Section 4.3, evaluating model performance in our setup is challenging due to the absence of analog ground truth. Instead, we rely on binary labels that are additionally noisy and subjective. All models under consideration produce continuous anomaly scores: DeepSAD outputs a positive-valued distance to the center of a hypersphere, Isolation Forest measures deviation from the mean tree depth (which can be negative), and OCSVM returns a signed distance to the decision boundary. Because these scores differ in scale and sign—and due to the lack of a reliable degradation threshold—it is not appropriate to evaluate performance using metrics such as accuracy or F1 score, both of which require classification at a fixed threshold.

Instead, we adopt threshold-independent evaluation curves that illustrate model behavior across the full range of possible thresholds. The most commonly used of these is the Receiver Operating Characteristic (ROC) [39] curve, along with its scalar summary metric, ROC AUC. ROC curves plot the true positive rate (TPR) against the false positive rate (FPR), providing insight into how well a model separates the two classes. However, as noted in [40], [41] and confirmed in our own testing, ROC AUC can be misleading under strong class imbalance—a common condition in anomaly detection.

To address this, we instead rely on Precision–Recall Curves (PRC) [42], which better capture model behavior on the minority class. PRC plots precision—the fraction of predicted anomalies that are correct—against recall—the fraction of true anomalies that are detected. As the decision threshold is lowered, recall increases but typically at the cost of precision, since more false positives are included. This tradeoff is captured across all thresholds. The metric definitions are as follows:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \quad \text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}.$$

In our evaluation, this distinction proved practically significant. As illustrated in Figure 5.5, ROC AUC values in (a) appear similarly strong for both, Isolation Forest and DeepSAD (0.693 vs. 0.782), suggesting comparable performance. However, the PRC in (b) reveals a clear divergence: while DeepSAD maintains high precision across recall levels, Isolation Forest suffers a steep decline in precision as recall increases, due to a high number of false positives. The resulting Average Precision (AP)—the area under the PRC—is much lower for Isolation Forest (0.207 vs. 0.633), offering a more realistic account of its performance under imbalance.

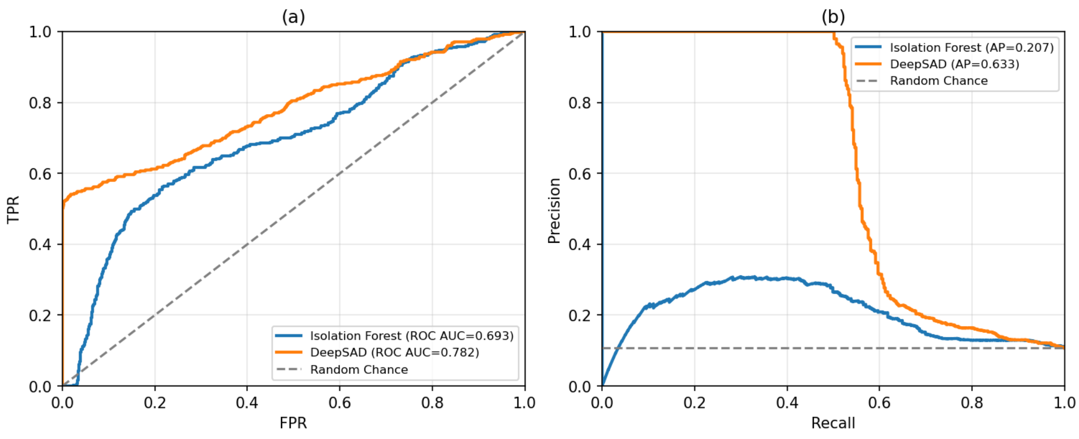


Figure 5.5: Comparison of ROC (a) and PRC (b) for the same evaluation run. ROC fails to reflect the poor performance of Isolation Forest, which misclassifies many normal samples as anomalous at lower thresholds. The PRC exposes this effect, resulting in a substantially lower AP for Isolation Forest than for DeepSAD.

In addition to cross-validated performance comparisons, we also apply the trained models to previously unseen, temporally ordered experiments to simulate inference in realistic conditions. Since each method produces scores on a different scale—with different signs and ranges—raw scores are not directly comparable. To enable comparison, we compute a z-score [43] per frame, defined as the number of standard deviations a score deviates from the mean of the normal data. To perform the normalization, we compute the mean and standard deviation of anomaly scores on a clean reference experiment. These values are then used to normalize scores from degraded experiments, making it easy to see how much each method's output deviates from its own baseline under degradation. It also facilitates a unified view across methods, even though their outputs are otherwise heterogeneous. In this way, z-score normalization supports threshold-free interpretation and enables consistent model comparison during inference.

In conclusion, the combination of unreliable thresholds and pronounced class imbalance renders traditional threshold-based metrics unsuitable for our setting. PRC and AP provide a more faithful representation of model behavior across thresholds, while z-score normalization extends this by enabling direct comparison of inference-time outputs across methods and conditions.

5.4 Experiment Overview & Computational Environment

Across all experiments, we vary three factors: (i) latent space dimensionality, (ii) encoder architecture (LeNet-inspired vs. Efficient), and (iii) the amount of semi-supervision (labeling regime). To keep results comparable, we fix the remaining training hyperparameters: all autoencoders are pretrained for $E_A = 50$ epochs with ADAM as an optimizer at a starting learning rate of $L_A = 1 \cdot 10^{-5}$; all DeepSAD models are then trained for $E_M = 150$ epochs with the same optimizer and starting learning rate ($L_M = 1 \cdot 10^{-5}$). The DeepSAD label-weighting parameter is kept at $\eta = 1$ and the regularization rate at $\lambda = 1 \cdot 10^{-6}$ for all runs. Every configuration is evaluated with 5-fold cross-validation, and we report fold means.

We first search over the latent bottleneck size by pretraining autoencoders only. For both encoder backbones, we evaluate latent sizes $32, 64, 128, 256, 512, 768$, and 1024. The goal is to identify compact yet expressive representations and to compare the autoencoding performance between the two network architectures, LeNet-inspired and Efficient. Additionally, we are interested in finding possible correlations between the autoencoder performance and the DeepSAD anomaly detection performance.

Using the same latent sizes and backbones, we train full DeepSAD models initialized from the pretrained encoders. We study three supervision regimes, from unsupervised to strongly supervised (see Table 5.2 for proportions within the training folds):

- **Unsupervised:** $(0, 0)$ labeled (normal, anomalous) samples.
- **Low supervision:** $(50, 10)$ labeled samples.
- **High supervision:** $(500, 100)$ labeled samples.

Percentages in Table 5.2 are computed relative to the training split of each fold (80% of the data) from the experiment-based labeling scheme. Importantly, for semi-supervised labels, we *only* use hand-selected, unambiguous smoke intervals from the manually-defined evaluation scheme to avoid injecting mislabeled data into training.

Table 5.2: Proportion of labeled samples in the training folds for each labeling regime. Percentages are computed relative to the available training data after 5-fold splitting (80% of the dataset per fold).

Regime	**% normals labeled**	**% anomalies labeled**	**% overall labeled**	**% overall unlabeled**
(0,0)	0.00%	0.00%	0.00%	100.00%
(50,10)	0.40%	1.46%	0.42%	99.58%
(500,100)	3.96%	14.57%	4.25%	95.75%

Combining 7 latent sizes, 2 architectures, and 3 labeling regimes yields $7 \times 2 \times 3 = 42$ DeepSAD configurations per evaluation protocol. Table 5.3 summarizes the grid.

These experiments were run on a computational environment for which we summarize the hardware and software stack in Table 5.4.

Pretraining runtimes for the autoencoders are reported in Table 5.5. These values are averaged across folds and labeling regimes, since the pretraining step itself does not make use of labels.

The full DeepSAD training times are shown in Table 5.6, alongside the two classical baselines, Isolation Forest and OCSVM. Here, the contrast between methods is clear: while DeepSAD requires on the order of 15–20 minutes of GPU training per configuration and fold, both baselines complete training in seconds on CPU. The OCSVM training can only be this fast due to the reduced input dimensionality from utilizing DeepSAD's pretraining encoder as a preprocessing step, although other dimensionality reduction methods may also be used, which could require less computational resources for this step.

Inference latency per sample is presented in Table 5.7. These measurements highlight an important property: once trained, all methods are extremely fast at inference, with DeepSAD operating in the sub-millisecond range and the classical baselines being even faster. This confirms

Table 5.3: Parameter space for the DeepSAD grid search. Each latent size is tested for both architectures and all labeling regimes.

	Latent sizes	**Architectures**	**Labeling regimes (normal, anomalous)**
Levels	32 64 128 256 512 768 1024	LeNet-inspired Efficient	(0,0) (50,10) (500,100)
Count	7	2	3
Total combinations		$7 \times 2 \times 3 = \mathbf{42}$	

Table 5.4: Computational Environment (Hardware & Software)

System	
Operating System	`Ubuntu 22.04.5 LTS`
Kernel	`6.5.0-44-generic`
Architecture	`x86_64`
CPU Model	`AMD Ryzen Threadripper 3970X 32-Core Processor`
CPU Cores (physical)	`32`
CPU Threads (logical)	`64`
CPU Base Frequency	`2200 MHz`
CPU Max Frequency	`3700 MHz`
Total RAM	`94.14 GiB`
GPU	
GPU Name	`NVIDIA GeForce RTX 4090`
GPU Memory	`23.99 GiB`
GPU Compute Capability	`8.9`
NVIDIA Driver Version	`535.161.07`
CUDA (Driver) Version	`12.2`
Software Environment	
Python	`3.12.11`
PyTorch	`2.7.1+cu126`
PyTorch Built CUDA	`12.6`
cuDNN (PyTorch build)	`90501`
scikit-learn	`1.7.0`
NumPy	`2.3.0`
SciPy	`1.15.3`

Table 5.5: Autoencoder pretraining runtime (seconds): mean ± std across folds and across semi-supervised lableling regimes.

Latent Dim.	Autoencoder Efficient	Autoencoder LeNet
32	1175.03 ± 35.87 s	384.90 ± 34.59 s
64	1212.53 ± 35.76 s	398.22 ± 41.25 s
128	1240.86 ± 11.51 s	397.98 ± 33.43 s
256	1169.72 ± 33.26 s	399.40 ± 38.20 s
512	1173.34 ± 34.99 s	430.31 ± 38.02 s
768	1204.45 ± 37.52 s	436.49 ± 37.13 s
1024	1216.79 ± 34.82 s	411.69 ± 34.82 s

Table 5.6: Training runtime: total seconds (mean ± std).

	DeepSAD		Baselines	
	LeNet	Efficient	IsoForest	OCSVM
Latent Dim.				
32	765 ± 91 s	1026 ± 84 s	0.55 ± 0.02 s	1.07 ± 00.29 s
64	815 ± 93 s	1124 ± 60 s	0.55 ± 0.02 s	1.98 ± 01.57 s
128	828 ± 63 s	1164 ± 02 s	0.55 ± 0.02 s	3.17 ± 02.63 s
256	794 ± 97 s	986 ± 82 s	0.55 ± 0.02 s	12.81 ± 14.19 s
512	806 ± 99 s	998 ± 80 s	0.55 ± 0.02 s	22.76 ± 23.52 s
768	818 ± 86 s	1053 ± 78 s	0.55 ± 0.02 s	14.24 ± 01.21 s
1024	770 ± 86 s	1054 ± 87 s	0.55 ± 0.02 s	28.20 ± 24.04 s

that, despite higher training costs, DeepSAD can be deployed in real-time systems without inference becoming a bottleneck.

Table 5.7: Inference latency (ms/sample): mean ± std across folds; baselines collapsed across networks and semi-labeling.

Latent Dim.	DeepSAD LeNet	DeepSAD Efficient	IsoForest	OCSVM
32	0.31 ± 0.04 ms	0.36 ± 0.05 ms	0.02 ± 0.00 ms	0.07 ± 0.02 ms
64	0.33 ± 0.06 ms	0.43 ± 0.04 ms	0.02 ± 0.00 ms	0.10 ± 0.06 ms
128	0.31 ± 0.04 ms	0.45 ± 0.02 ms	0.02 ± 0.00 ms	0.16 ± 0.09 ms
256	0.30 ± 0.04 ms	0.33 ± 0.02 ms	0.02 ± 0.00 ms	0.30 ± 0.21 ms
512	0.32 ± 0.04 ms	0.33 ± 0.02 ms	0.02 ± 0.00 ms	0.63 ± 0.65 ms
768	0.33 ± 0.03 ms	0.41 ± 0.06 ms	0.02 ± 0.00 ms	0.39 ± 0.07 ms
1024	0.27 ± 0.02 ms	0.39 ± 0.05 ms	0.02 ± 0.00 ms	0.94 ± 0.98 ms

Together, these results provide a comprehensive overview of the computational requirements of our experimental setup. They show that while our deep semi-supervised approach is significantly more demanding during training than classical baselines, it remains highly efficient at inference, which is the decisive factor for deployment in time-critical domains such as rescue robotics.

6 Results and Discussion

In this chapter, we present the evaluation experiments, based on the experimental setup described in Chapter 5. We begin in Section 6.1 with the pretraining stage, where the two autoencoder architectures were trained across multiple latent space dimensionalities. These results provide insight into the representational capacity of each architecture. In Section 6.2, we turn to the main experiments: training DeepSAD models and benchmarking them against baseline algorithms (Isolation Forest and OCSVM). Finally, in Section 6.3, we present inference results on data that were held out during training. These plots illustrate how the algorithms behave when applied sequentially to unseen data, offering a more practical perspective on their potential for real-world rescue robotics applications.

6.1 Autoencoder Pretraining Results

The results of pretraining the two autoencoder architectures are summarized in Table 6.1. Reconstruction performance is reported as mean squared error (MSE), with trends visualized in Figure 6.1. The results show that the modified Efficient architecture consistently outperforms the LeNet-inspired baseline across all latent space dimensionalities. The improvement is most pronounced at lower-dimensional bottlenecks (e.g., 32 or 64 dimensions) but remains observable up to 1024 dimensions, although the gap narrows.

Table 6.1: Autoencoder pretraining MSE losses across latent dimensions. Left: overall loss; Right: anomaly-only loss. The mean across folds is reported. Maximum observed standard deviation across all cells (not shown): 0.0067.

	Overall loss (MSE)		Anomaly loss (MSE)	
Latent Dim.	LeNet	Efficient	LeNet	Efficient
32	0.0223	**0.0136**	0.0701	**0.0554**
64	0.0168	**0.0117**	0.0613	**0.0518**
128	0.0140	**0.0110**	0.0564	**0.0506**
256	0.0121	**0.0106**	0.0529	**0.0498**
512	0.0112	**0.0103**	0.0514	**0.0491**
768	0.0109	**0.0102**	0.0505	**0.0490**
1024	0.0106	**0.0101**	0.0500	**0.0489**

Because the overall reconstruction loss might obscure how well encoders represent anomalous samples, we additionally evaluate reconstruction errors only on degraded samples from manually-defined smoke segments (Figure 6.2). As expected, reconstruction losses are higher on these challenging samples than in the overall evaluation. However, the relative advantage of the Efficient architecture remains, suggesting that its improvements extend to anomalous inputs as well.

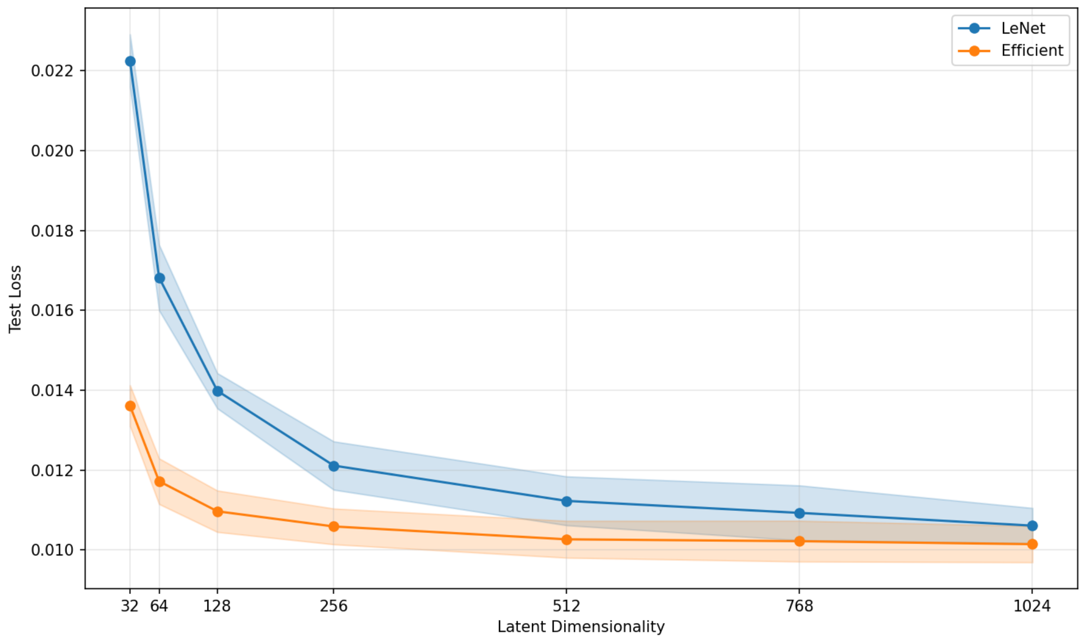


Figure 6.1: Reconstruction loss (MSE) across latent dimensions for LeNet-inspired and Efficient autoencoder architectures.

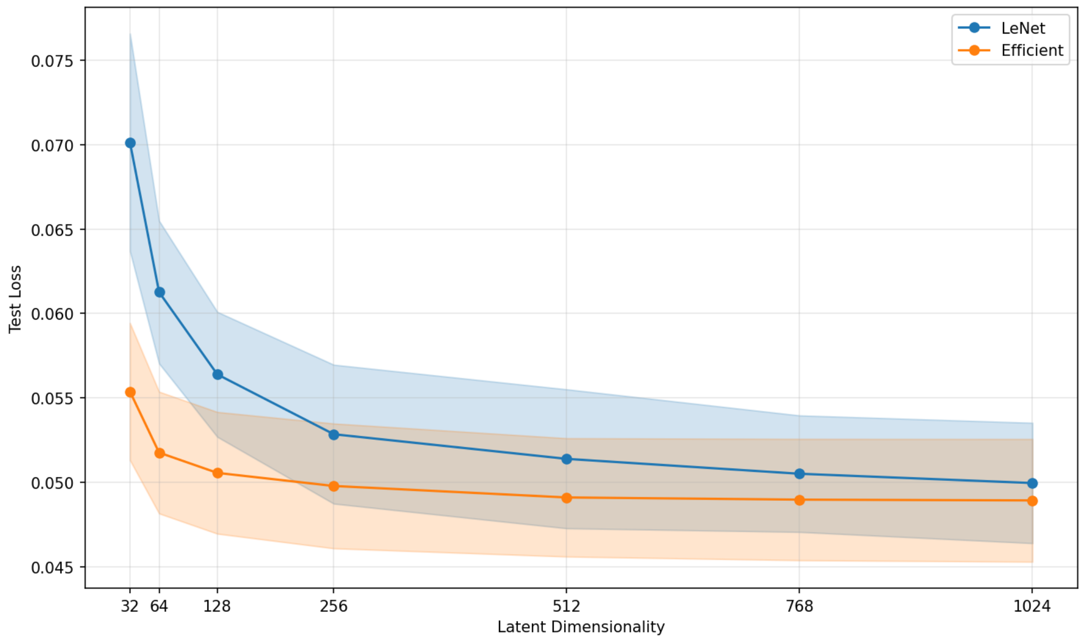


Figure 6.2: Reconstruction loss (MSE) across latent dimensions for LeNet-inspired and Efficient autoencoder architectures, evaluated only on degraded data from manually-defined smoke experiments.

6.2 DeepSAD Detection Performance

Due to the challenges of ground truth quality, evaluation results must be interpreted with care. As introduced earlier, we consider two complementary evaluation schemes:

- **Experiment-based labels:** An objective way to assign anomaly labels to all frames from degraded runs. However, this also marks many near-normal frames at the start and end of runs as anomalous. These knowingly false labels lower the maximum achievable AP, because even an ideal model would be forced to classify some normal frames as anomalous.

- **Manually-defined labels:** A cleaner ground truth, containing only clearly degraded frames. This removes mislabeled intervals and allows nearly perfect separation. However, it also simplifies the task too much, because borderline cases are excluded.

Table 6.2 summarizes average precision (AP) across latent dimensions, labeling regimes, and methods. Under experiment-based evaluation, both DeepSAD variants consistently outperform the baselines, reaching AP values around 0.60–0.66 compared to 0.21 for the Isolation Forest and 0.31–0.49 for OCSVM. Under manually-defined evaluation, DeepSAD achieves nearly perfect AP in all settings, while the baselines remain much lower. This contrast shows that the lower AP under experiment-based evaluation is not a weakness of DeepSAD itself, but a direct result of mislabeled samples in the evaluation data. Therefore, the manually-defined scheme confirms that DeepSAD separates clearly normal from clearly degraded frames very well, while also highlighting that label noise must be kept in mind when interpreting the experiment-based results.

The precision–recall curves for experiment-based evaluation (Figure 6.3) illustrate these effects more clearly. For DeepSAD, precision stays close to 1 until about 0.5 recall, after which it drops off sharply. This plateau corresponds to the fraction of truly degraded frames in the anomalous set. Once recall moves beyond this point, the evaluation demands that the model also "find" the mislabeled anomalies near the run boundaries. To do so, the decision threshold must be lowered so far that many normal frames are also flagged, which causes precision to collapse. The baselines behave differently: OCSVM shows a smooth but weaker decline without a strong high-precision plateau, while Isolation Forest collapses to near-random performance. These operational differences are hidden in a single AP number but are important for judging how the methods would behave in deployment.

Taken together, the two evaluation schemes provide complementary insights. The experiment-based labels offer a noisy but realistic setting that shows how methods cope with ambiguous data, while the manually-defined labels confirm that DeepSAD can achieve nearly perfect separation when the ground truth is clean. The combination of both evaluations makes clear that (i) DeepSAD is stronger than the baselines under both conditions, (ii) the apparent performance limits under experiment-based labels are mainly due to label noise, and (iii) interpreting results requires care, since performance drops in the curves often reflect mislabeled samples rather than model failures. At the same time, both schemes remain binary classifications and therefore cannot directly evaluate the central question of whether anomaly scores can serve as a continuous measure of degradation. For this reason, we extend the analysis in Section 6.3, where inference on entire unseen experiments is used to provide a more intuitive demonstration of the methods' potential for quantifying LiDAR degradation in practice.

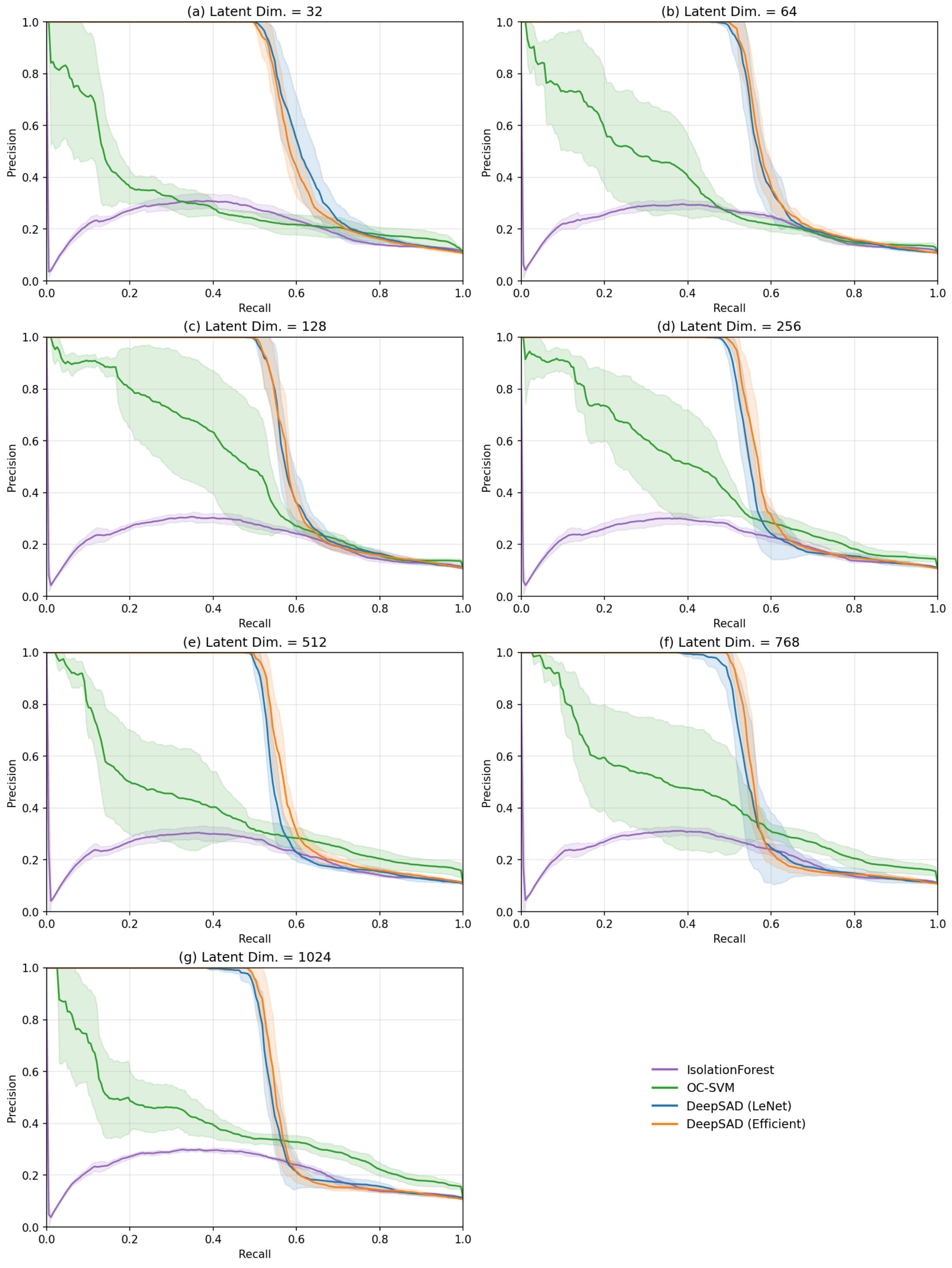


Figure 6.3: Representative precision–recall curves (a) - (g) over all latent dimensionalities 32 - 1024 for semi-labeling regime 0/0 from experiment-based evaluation labels. DeepSAD maintains a large high-precision operating region before collapsing; OCSVM declines smoother but exhibits high standard deviation between folds; IsoForest collapses quickly and remains flat. DeepSAD's fall-off is at least partly due to known mislabeled evaluation targets.

Table 6.2: AP means across 5 folds for both evaluations, grouped by labeling regime. Mean observed standard deviation per method: DeepSAD (LeNet) 0.015; DeepSAD (Efficient) 0.012; IsoForest 0.011; OCSVM 0.091.

	Experiment-based eval.				Manually-defined eval.			
Latent Dim.	DeepSAD (LeNet)	DeepSAD (Efficient)	IsoForest	OCSVM	DeepSAD (LeNet)	DeepSAD (Efficient)	IsoForest	OCSVM
Labeling regime: 0/0 *(normal/anomalous samples labeled)*								
32	**0.664**	0.650	0.217	0.315	**1.000**	**1.000**	0.241	0.426
64	0.635	**0.643**	0.215	0.371	**1.000**	**1.000**	0.233	0.531
128	**0.642**	**0.642**	0.218	0.486	**1.000**	**1.000**	0.241	0.729
256	0.615	**0.631**	0.214	0.452	0.999	**1.000**	0.236	0.664
512	0.613	**0.635**	0.216	0.397	**1.000**	**1.000**	0.241	0.550
768	0.609	**0.617**	0.219	0.439	0.997	**1.000**	0.244	0.624
1024	0.607	**0.612**	0.215	0.394	0.997	**1.000**	0.235	0.529
Labeling regime: 50/10 *(normal/anomalous samples labeled)*								
32	0.569	**0.582**	0.217	0.315	0.933	**0.976**	0.241	0.426
64	0.590	**0.592**	0.215	0.371	0.970	**0.986**	0.233	0.531
128	0.566	**0.588**	0.218	0.486	0.926	**0.983**	0.241	0.729
256	**0.598**	0.587	0.214	0.452	0.978	**0.984**	0.236	0.664
512	0.550	**0.587**	0.216	0.397	0.863	**0.978**	0.241	0.550
768	**0.596**	0.577	0.219	0.439	**0.992**	0.974	0.244	0.624
1024	**0.601**	0.568	0.215	0.394	**0.990**	0.966	0.235	0.529
Labeling regime: 500/100 *(normal/anomalous samples labeled)*								
32	**0.625**	0.621	0.217	0.315	**0.999**	0.997	0.241	0.426
64	0.611	**0.621**	0.215	0.371	0.996	**0.998**	0.233	0.531
128	0.607	**0.615**	0.218	0.486	0.996	**0.998**	0.241	0.729
256	0.604	**0.612**	0.214	0.452	0.984	**0.998**	0.236	0.664
512	0.578	**0.608**	0.216	0.397	0.916	**0.998**	0.241	0.550
768	0.597	**0.598**	0.219	0.439	0.994	**0.995**	0.244	0.624
1024	**0.601**	0.591	0.215	0.394	0.990	**0.993**	0.235	0.529

Effect of latent space dimensionality. During autoencoder pretraining, we observed that reconstruction loss decreased monotonically with larger latent spaces, as expected: a bigger bottleneck allows the encoder–decoder to retain more information. If autoencoder performance were directly predictive of DeepSAD performance, we would therefore expect average precision to improve with larger latent dimensions. The actual results, however, show the opposite trend (Figure 6.4): compact latent spaces (32–128) achieve the highest AP, while performance declines as the latent size grows. This inverse correlation is most clearly visible in the unsupervised case. Part of this effect can be attributed to evaluation label noise, which larger spaces amplify. More importantly, it shows that autoencoder performance does not translate directly into DeepSAD performance. Pretraining losses can still help compare different architectures for robustness and performance, but they cannot be used to tune the latent dimensionality: the dimensionality that minimizes reconstruction loss in pretraining is not necessarily the one that maximizes anomaly detection performance in DeepSAD.

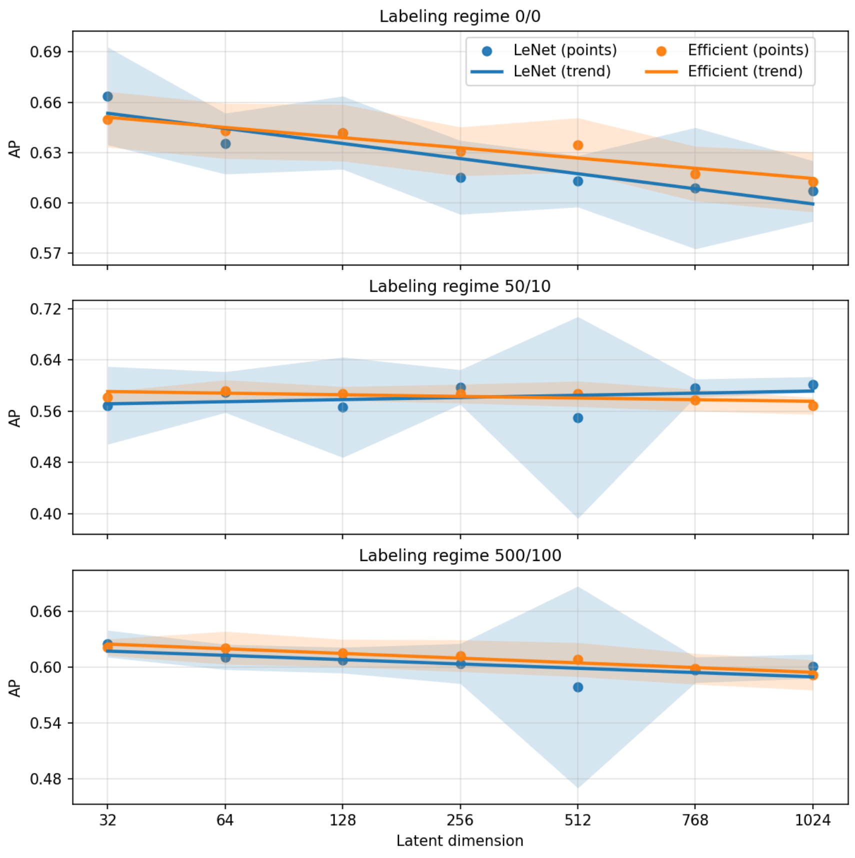


Figure 6.4: AP as a function of latent dimension (experiment-based evaluation). DeepSAD shows an inverse correlation between AP and latent space size.

Effect of semi-supervised labeling. Table 6.2 shows that the unsupervised regime (0/0) achieves the best AP, while the lightly supervised regime (50/10) performs worst. With many labels (500/100), performance improves again but remains slightly below the unsupervised case. This pattern also appears under the manually-defined evaluation, which excludes mislabeled frames. Consequently, the drop with light supervision cannot be explained by noisy evaluation targets, but must stem from the training process itself.

The precision–recall curves in Figure 6.5 show that the overall curve shapes are similar across regimes, but shifted relative to one another in line with the AP ordering (0/0) > (500/100) > (50/10). We attribute these shifts to overfitting: when only a few anomalies are labeled, the model fits them too strongly, and if those examples differ too much from other anomalies, generalization suffers. This explains why lightly supervised training performs even worse than unsupervised training, which avoids this bias.

The LeNet variant illustrates this effect most clearly, showing unusually high variance across folds in the lightly supervised case. In several folds, precision drops untypically early, which supports the idea that the model has overfit to a poorly chosen subset of labeled anomalies. The Efficient variant is less affected, maintaining more stable precision plateaus, which suggests it is more robust to such overfitting, which we observe consistently for nearly all latent dimensionalities.

With many labels (500/100), the results become more stable again, and the PRC curves closely resemble the unsupervised case, only shifted slightly left. A larger and more diverse set of labeled anomalies reduces the risk of unlucky sampling and improves generalization, but

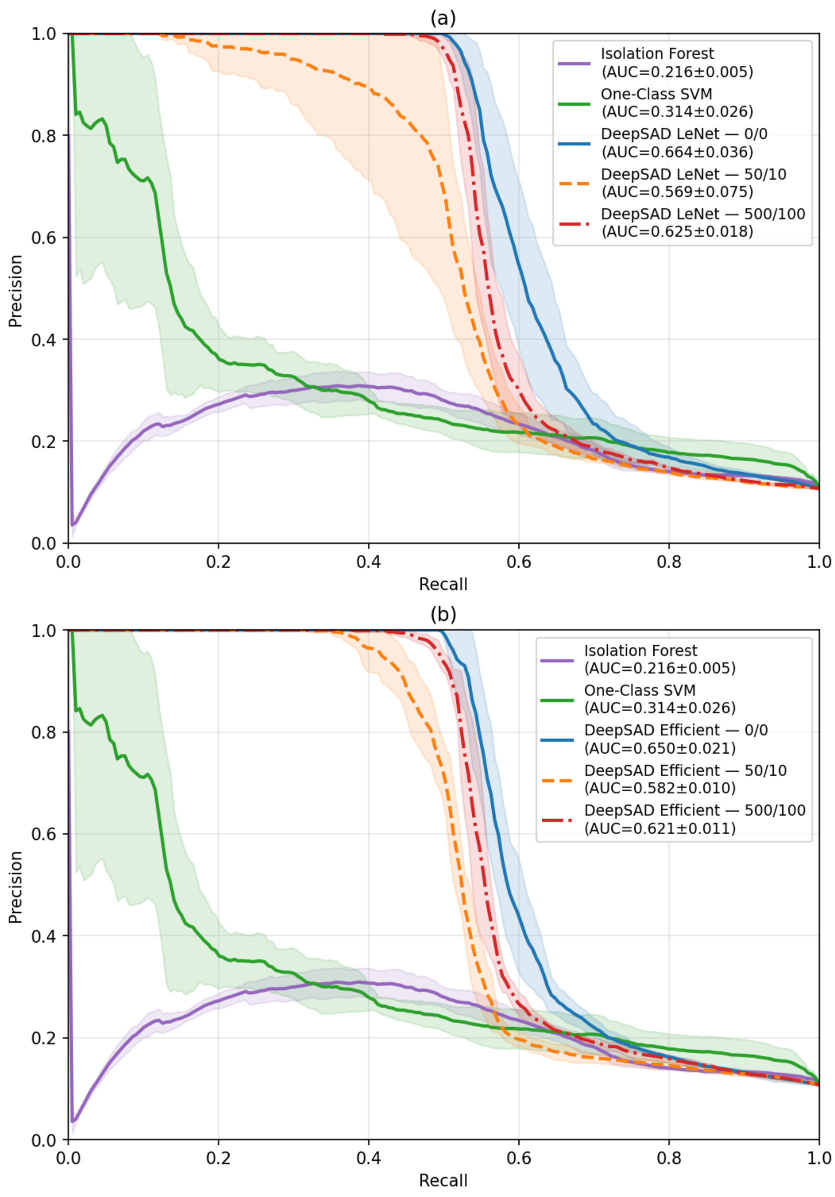


Figure 6.5: PRCs from experiment-based evaluation for all three labeling regimes (unsupervised, lightly supervised, heavily supervised), shown separately for the LeNet-inspired (a) and Efficient (b) encoders. Baseline methods are included for comparison. Latent dimension 32 is shown as it achieved the best overall AP and is representative of the typical PRC shapes across dimensions.

it still cannot fully match the unsupervised regime, where no overfitting to a specific labeled subset occurs. The only exception is an outlier at latent dimension 512 for LeNet, where the curve again resembles the lightly supervised case, likely due to label sampling effects amplified by higher latent capacity.

In summary, three consistent patterns emerge: (i) a very small number of labels can hurt performance by causing overfitting to specific examples, (ii) many labels reduce this problem but still do not surpass unsupervised generalization, and (iii) encoder architecture strongly affects robustness, with the LeNet-inspired encoder being more sensitive to unstable behavior than the Efficient encoder.

6.3 Inference on Held-Out Experiments

In addition to the evaluation of PRC and AP obtained from k-fold cross-validation with varying hyperparameters, we also examine the behavior of the fully trained methods when applied to previously unseen, held-out experiments. While the prior analysis provided valuable insights into the classification capabilities of the methods, it was limited by two factors: first, the binary ground-truth labels were of uneven quality due to the aforementioned mislabeling of frames, and second, the binary formulation does not reflect our overarching goal of quantifying sensor degradation on a continuous scale. To provide a more intuitive understanding of how the methods might perform in real-world applications, we therefore present results from running inference sequentially on entire experiments. These frame-by-frame time-axis plots simulate online inference and illustrate how anomaly scores evolve as data is captured, thereby serving as a candidate metric for quantifying the degree of LiDAR degradation during operation.

As discussed in Section 5.3, we apply z-score normalization to enable comparison of the different methods during inference. After normalization, the resulting time series were still highly noisy, which motivated the application of exponential moving average (EMA) smoothing. EMA was chosen because it is causal (does not rely on future data) and thus suitable for real-time inference. Although it introduces a small time delay, this delay is shorter than for other smoothing techniques, such as running averages.

The plots in Figure 6.6 highlight important differences in how well the tested methods distinguish between normal and degraded sensor conditions. The plots show how strongly the method's scores deviate from their clean-data baseline and include statistical indicators (missing points and near-sensor particle hits) in blue and green.

Among the four approaches, the strongest separation is achieved by DeepSAD Efficient (b), followed by DeepSAD LeNet (a), then OCSVM (c). For Isolation Forest (d), the anomaly scores are already elevated in the clean experiment, which prevents reliable differentiation between normal and degraded runs and makes the method unsuitable in this context.

When comparing the methods to the statistical indicators, some similarities in shape may suggest that the methods partly capture these statistics, although such interpretations should be made with caution. The anomaly detection models are expected to have learned additional patterns that are not directly observable from simple statistics, and these may also contribute to their ability to separate degraded from clean data.

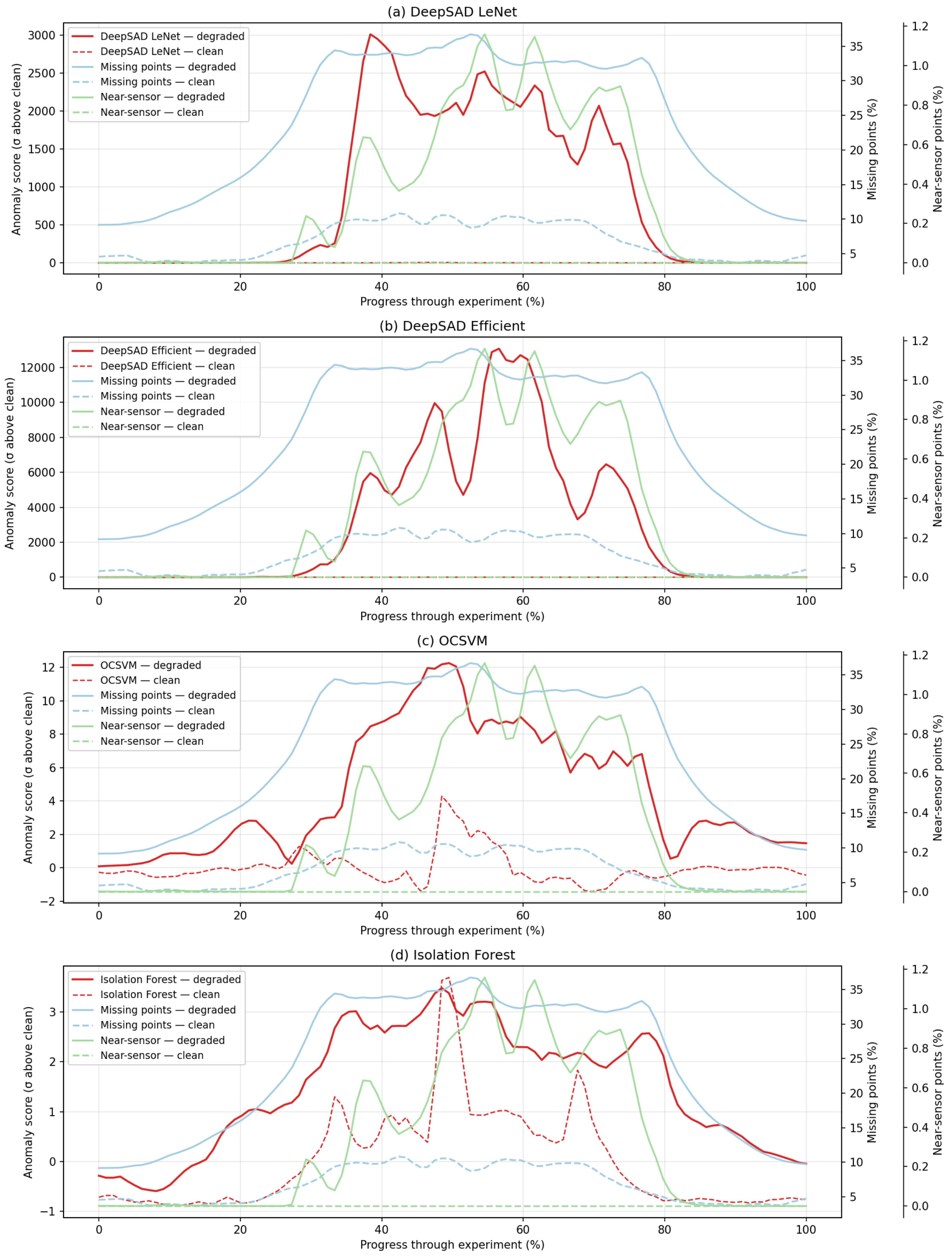


Figure 6.6: Comparison of inference on unseen data for clean (dashed) vs. degraded (solid) experiments. Each subplot, (a) - (d), compares one method's anomaly score deviation from its clean baseline in red to statistical indicators in blue and green, which indicate the percentage of missing LiDAR points and near-sensor particle hits, respectively. Latent dimension: 32; training regime: 0 normal, 0 anomalous samples. Smoothed with EMA $\alpha = 0.1$.

7 Conclusion and Future Work

This thesis set out to answer the research question stated in Chapter 1:

> Can autonomous robots quantify the reliability of LiDAR sensor data in hazardous environments to make more informed decisions?

Our results indicate a qualified “yes.” Using anomaly detection (AD)—in particular DeepSAD—we can obtain scores that (i) separate clearly normal from clearly degraded scans and (ii) track degradation trends over time on held-out experiments (see Sections 6.2 and 6.3). At the same time, the absence of robust ground truth limits how confidently we can assess *continuous* quantification quality and complicates cross-method comparisons. The remainder of this chapter summarizes what we contributed, what we learned, and what is still missing.

Main contributions.

- **Empirical comparison for LiDAR degradation.** A systematic evaluation of DeepSAD against Isolation Forest and OCSVM across latent sizes and labeling regimes, showing that DeepSAD consistently outperforms the baselines under both evaluation schemes (Section 6.2).
- **Latent dimensionality insight.** Autoencoder pretraining loss decreases with larger latent spaces, but DeepSAD performance shows the opposite trend: compact bottlenecks (32–128) achieve the highest average precision (AP). This contrast demonstrates that pretraining performance does not directly predict DeepSAD performance—latent dimensionality cannot be tuned via autoencoder loss alone, even though it remains useful for comparing architectures.
- **Semi-supervision insight.** In our data, *unsupervised* DeepSAD performed best; *light* labeling (50/10) performed worst; *many* labels (500/100) partially recovered performance but did not surpass the unsupervised approach. Evidence from precision–recall curve (PRC) shapes and fold variance points to *training-side overfitting to a small labeled set*, an effect that persists even under clean manually-defined evaluation (Table 6.2, Figure 6.5).
- **Encoder architecture matters.** The Efficient encoder, specifically tailored to the application at hand, outperformed the LeNet-inspired variant in pretraining and downstream AD, indicating that representation quality substantially affects DeepSAD performance (Section 6.1, Section 6.2).
- **Temporal inference recipe.** For deployment-oriented analysis, we propose z-score normalization based on clean data and causal EMA smoothing to obtain interpretable time-series anomaly scores on full experiments (Section 6.3).

Practical recommendations. For settings similar to ours, we recommend: (i) use PRC and AP for model selection and reporting, since ROC and its AUC can give overly optimistic results under strong class imbalance; (ii) prefer compact latent spaces (e.g., 32–128) and determine the smallest dimensionality that still preserves task-relevant information; (iii) evaluate multiple encoder architectures, as design choices strongly affect performance and robustness; (iv) avoid very small labeled sets, which can cause overfitting to narrow anomaly exemplars. If labels are used, collect many and diverse examples—though unsupervised training may still generalize best.

We now turn to the main limiting factor that emerged throughout this work: the lack of robust, expressive ground truth for LiDAR degradation and its downstream impact.

7.1 Missing Ground Truth as an Obstacle

The most significant obstacle identified in this work is the absence of a robust and comprehensive ground truth for LiDAR degradation. As discussed in Chapter 4, it is not trivial to define what "degradation" precisely means in practice. Although error models for LiDAR and theoretical descriptions of how airborne particles affect laser returns exist, these models typically quantify errors at the level of individual points (e.g., missing returns, spurious near-range hits). Such metrics, however, may not be sufficient to assess the impact of degraded data on downstream perception. For example, a point cloud with relatively few but highly localized errors—such as those caused by a dense smoke cloud—may cause a SLAM algorithm to misinterpret the region as a solid obstacle. In contrast, a point cloud with a greater number of dispersed errors might be easier to filter and thus cause little or no disruption in mapping. Consequently, the notion of "degradation" must extend beyond point-level error statistics to include how different error patterns propagate to downstream modules.

To our knowledge, no public datasets with explicit ground truth for LiDAR degradation exist. Even if such data were collected, for example, with additional smoke sensors, it is unclear whether this would provide a usable ground truth. A smoke sensor measures only at a single point in space, while LiDAR observes many points across the environment from a distance, so the two do not directly translate. In our dataset, we relied on the fact that clean and degraded experiments were clearly separated: data from degraded runs was collected only after artificial smoke had been released. However, the degree of degradation varied strongly within each run. Because the smoke originated from a single machine in the middle of the sensor platform's traversal path, early and late frames were often nearly as clear as those from clean experiments. This led to mislabeled frames at the run boundaries and limited the reliability of experiment-based evaluation. As shown in Section 6.2, this effect capped achievable AP scores even for strong models. The underlying difficulty is not only label noise, but also the challenge of collecting labeled subsets that are representative of the full range of anomalies.

One promising direction is to evaluate degradation not directly on raw LiDAR frames but via its downstream impact. For example, future work could assess degradation based on discrepancies between a previously mapped 3D environment and the output of a SLAM algorithm operating under degraded conditions. In such a setup, subjective labeling may still be required in special cases (e.g., dense smoke clouds treated as solid obstacles by SLAM), but it would anchor evaluation more closely to the ultimate users of the data.

Finally, the binary ground truth employed here is insufficient for the quantification goal. As shown in Section 6.3, DeepSAD's anomaly scores appear suitable not only for classification but also for expressing intermediate levels of degradation. Analog evaluation targets would therefore be highly valuable, as they would allow testing whether anomaly scores increase consistently with degradation severity, rather than only separating "normal" from "degraded."

7.2 Insights into DeepSAD and AD for Degradation Quantification

This work has shown that the DeepSAD principle is applicable to LiDAR degradation in hazardous environments and yields promising detection performance as well as runtime feasibility (see Sections 6.2 and 5.4). Compared to simpler baselines such as Isolation Forest and OCSVM, DeepSAD achieved much stronger separation between clean and degraded data. While OCSVM showed smoother but weaker separation, and Isolation Forest produced high false positives even in clean runs, both DeepSAD variants maintained large high-precision regions before collapsing under mislabeled evaluation targets.

However, the semi-supervised component of DeepSAD did not improve results in our setting. In fact, adding a small number of labels often reduced performance due to overfitting to narrow subsets of anomalies. While larger labeled sets stabilized training, they still did not surpass the unsupervised regime (see Section 6.2). This suggests that without representative and diverse labeled anomalies, unsupervised training remains the safer choice.

We also observed that the choices of encoder architecture and latent dimensionality are critical. The Efficient encoder consistently outperformed the LeNet-inspired baseline, producing more stable precision–recall curves and stronger overall results. Similarly, compact latent spaces (32–128 dimensions) yielded the best performance and proved more robust under noisy evaluation conditions, while larger latent spaces amplified the impact of mislabeled samples and caused sharper precision collapses. These findings underline the importance of representation design for robust anomaly detection.

Finally, inference experiments showed that DeepSAD's anomaly scores can track degradation trends over time when normalized and smoothed, suggesting potential for real-world quantification. Future work could explore per-sample weighting of semi-supervised targets, especially if analog ground truth becomes available, allowing DeepSAD to capture varying degrees of degradation as a graded rather than binary signal.

7.3 Open Questions and Future Work

Several promising avenues remain open for future exploration:

- **Temporal modeling:** Instead of treating frames independently, future methods could directly model the difference between temporally consecutive frames to capture dynamic aspects of degradation.

- **LiDAR intensity:** LiDAR typically save an intensity value per point, indicating the strength of the reflected optical signal, which could be incorporated to improve degradation quantification.

- **Sensor fusion:** Combining LiDAR with complementary sensors (e.g., ultrasonic sensors that penetrate dense clouds) could mitigate blind spots inherent to single-sensor evaluation.

- **Input segmentation:** The DeepSAD architecture tested here processed full 360° LiDAR scans. This may obscure localized degradations. Segmenting point clouds into angular sectors and computing anomaly scores per sector could provide more fine-grained quantification. Preliminary tests in this direction were promising, but were not pursued further in this thesis.

- **Cross-sensor generalization:** Current experiments assume identical sensor resolution. Extending the method to work across different LiDAR types, including those with varying angular resolutions, remains an open question and would enhance applicability in heterogeneous robotic fleets and allow the incorporation of more datasets during training.

In summary, while this thesis demonstrates the feasibility of using anomaly detection for LiDAR degradation quantification, significant challenges remain. Chief among them are the definition and collection of ground truth, the development of analog evaluation targets, and architectural adaptations for more complex real-world scenarios. Addressing these challenges will be critical for moving from proof-of-concept to practical deployment in rescue robotics and beyond.

Bibliography

[1] F. E. and, “Xli. on discordant observations,” *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 23, no. 143, pp. 364–375, 1887. DOI: 10.1080/14786448708628471. eprint: https://doi.org/10.1080/14786448708628471. [Online]. Available: https://doi.org/10.1080/14786448708628471.

[2] Q. Wei, Y. Ren, R. Hou, B. Shi, J. Y. Lo, and L. Carin, “Anomaly detection for medical images based on a one-class classification,” in *Medical Imaging 2018: Computer-Aided Diagnosis*, N. Petrick and K. Mori, Eds., ser. Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, vol. 10575, Feb. 2018, 105751M, p. 105751M. DOI: 10.1117/12.2293408.

[3] M. Ul Hassan, M. H. Rehmani, and J. Chen, “Anomaly detection in blockchain networks: A comprehensive survey,” *IEEE Communications Surveys & Tutorials*, vol. 25, no. 1, pp. 289–318, 2023. DOI: 10.1109/COMST.2022.3205643.

[4] D. Y. Oh and I. D. Yun, “Residual error based anomaly detection using auto-encoder in smd machine sound,” *Sensors*, vol. 18, no. 5, 2018, ISSN: 1424-8220. DOI: 10.3390/s18051308. [Online]. Available: https://www.mdpi.com/1424-8220/18/5/1308.

[5] V. Chandola, A. Banerjee, and V. Kumar, “Anomaly detection: A survey,” *ACM Comput. Surv.*, vol. 41, no. 3, Jul. 2009, ISSN: 0360-0300. DOI: 10.1145/1541880.1541882. [Online]. Available: https://doi.org/10.1145/1541880.1541882.

[6] C. Cortes and V. Vapnik, “Support-vector networks,” *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[7] S. Lloyd, “Least squares quantization in pcm,” *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, 1982. DOI: 10.1109/TIT.1982.1056489.

[8] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, “A density-based algorithm for discovering clusters in large spatial databases with noise.,” in *KDD*, E. Simoudis, J. Han, and U. M. Fayyad, Eds., AAAI Press, 1996, pp. 226–231, ISBN: 1-57735-004-9. [Online]. Available: http://dblp.uni-trier.de/db/conf/kdd/kdd96.html#EsterKSX96.

[9] K. P. F.R.S., “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*, vol. 2, no. 11, pp. 559–572, 1901. DOI: 10.1080/14786440109462720.

[10] L. Ruff, R. A. Vandermeulen, N. Görnitz, A. Binder, E. Müller, K. Mü ller, and M. Kloft, “Deep semi-supervised anomaly detection,” *CoRR*, vol. abs/1906.02694, 2019. arXiv: 1906.02694. [Online]. Available: http://arxiv.org/abs/1906.02694.

[11] P. Bergmann and D. Sattlegger, “Anomaly detection in 3d point clouds using deep geometric descriptors,” in *2023 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, IEEE, Jan. 2023, pp. 2612–2622. DOI: 10.1109/wacv56688.2023.00264. [Online]. Available: http://dx.doi.org/10.1109/WACV56688.2023.00264.

[12] B. Rodríguez-Cuenca, S. García-Cortés, C. Ordóñez, and M. Alonso, “Automatic detection and classification of pole-like objects in urban point cloud data using an anomaly detection algorithm,” *Remote Sensing*, vol. 7, no. 10, pp. 12 680–12 703, Sep. 2015, ISSN: 2072-4292. DOI: 10.3390/rs71012680. [Online]. Available: http://dx.doi.org/10.3390/rs71012680.

[13] A. L. Samuel, “Some studies in machine learning using the game of checkers,” *IBM Journal of Research and Development*, vol. 3, no. 3, pp. 210–229, Jul. 1959, ISSN: 0018-8646. DOI: 10.1147/rd.33.0210. [Online]. Available: http://dx.doi.org/10.1147/rd.33.0210.

[14] J. Morimoto and F. Ponton, “Virtual reality in biology: Could we become virtual naturalists?” *Evolution: Education and Outreach*, vol. 14, no. 1, May 2021, ISSN: 1936-6434. DOI: 10.1186/s12052-021-00147-x. [Online]. Available: http://dx.doi.org/10.1186/s12052-021-00147-x.

[15] M. E. Villa-Pérez, M. Á. Álvarez-Carmona, O. Loyola-González, M. A. Medina-Pérez, J. C. Velazco-Rossell, and K.-K. R. Choo, “Semi-supervised anomaly detection algorithms: A comparative summary and future research directions,” *Knowledge-Based Systems*, vol. 218, p. 106 878, Apr. 2021, ISSN: 0950-7051. DOI: 10.1016/j.knosys.2021.106878. [Online]. Available: http://dx.doi.org/10.1016/j.knosys.2021.106878.

[16] L. Ruff, R. Vandermeulen, N. Goernitz, L. Deecke, S. A. Siddiqui, A. Binder, E. Müller, and M. Kloft, “Deep one-class classification,” in *Proceedings of the 35th International Conference on Machine Learning*, J. Dy and A. Krause, Eds., ser. Proceedings of Machine Learning Research, vol. 80, PMLR, Oct. 2018, pp. 4393–4402. [Online]. Available: https://proceedings.mlr.press/v80/ruff18a.html.

[17] L. Weng, “From autoencoder to beta-vae,” *lilianweng.github.io*, 2018. [Online]. Available: https://lilianweng.github.io/posts/2018-08-12-vae/.

[18] R. Linsker, “Self-organization in a perceptual network,” *Computer*, vol. 21, no. 3, pp. 105–117, 1988. DOI: 10.1109/2.36.

[19] J. Chen, S. Sathe, C. Aggarwal, and D. Turaga, “Outlier detection with autoencoder ensembles,” in *Proceedings of the 2017 SIAM International Conference on Data Mining.* Society for Industrial and Applied Mathematics, Jun. 2017, pp. 90–98, ISBN: 9781611974973. DOI: 10.1137/1.9781611974973.11. [Online]. Available: http://dx.doi.org/10.1137/1.9781611974973.11.

[20] D. Gong, L. Liu, V. Le, B. Saha, M. R. Mansour, S. Venkatesh, and A. Van Den Hengel, “Memorizing normality to detect anomaly: Memory-augmented deep autoencoder for unsupervised anomaly detection,” in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, IEEE, Oct. 2019, pp. 1705–1714. DOI: 10.1109/iccv.2019.00179. [Online]. Available: http://dx.doi.org/10.1109/ICCV.2019.00179.

[21] F. H. Nahhas, H. Z. M. Shafri, M. I. Sameen, B. Pradhan, and S. Mansor, “Deep learning approach for building detection using lidar–orthophoto fusion,” *Journal of Sensors*, vol. 2018, pp. 1–12, Aug. 2018, ISSN: 1687-7268. DOI: 10.1155/2018/7212307. [Online]. Available: http://dx.doi.org/10.1155/2018/7212307.

[22] N. Druml, I. Maksymova, T. Thurner, D. Lierop, M. Hennecke, and A. Foroutan, “1d mems micro-scanning lidar,” English, Sep. 2018.

[23] R. C. Smith and P. Cheeseman, “On the representation and estimation of spatial uncertainty,” *The International Journal of Robotics Research*, vol. 5, no. 4, pp. 56–68, Dec. 1986, ISSN: 1741-3176. DOI: 10.1177/027836498600500404. [Online]. Available: http://dx.doi.org/10.1177/027836498600500404.

[24] J.-I. Park, S. Jo, H.-T. Seo, and J. Park, “Lidar denoising methods in adverse environments: A review,” *IEEE Sensors Journal*, vol. 25, no. 5, pp. 7916–7932, Mar. 2025, ISSN: 2379-9153. DOI: 10.1109/jsen.2025.3526175. [Online]. Available: http://dx.doi.org/10.1109/JSEN.2025.3526175.

[25] T. Parsons, J. Seo, B. Kim, H. Lee, J.-C. Kim, and M. Cha, “Dust de-filtering in lidar applications with conventional and cnn filtering methods,” *IEEE Access*, vol. 12, pp. 22 032–22 042, 2024, ISSN: 2169-3536. DOI: 10.1109/access.2024.3362804. [Online]. Available: http://dx.doi.org/10.1109/ACCESS.2024.3362804.

[26] A. Kyuroson, A. Koval, and G. Nikolakopoulos, “Efficient real-time smoke filtration with 3d lidar for search and rescue with autonomous heterogeneous robotic systems,” in *IECON 2023- 49th Annual Conference of the IEEE Industrial Electronics Society*, IEEE, Oct. 2023, pp. 1–7. DOI: 10.1109/iecon51785.2023.10312303. [Online]. Available: http://dx.doi.org/10.1109/IECON51785.2023.10312303.

[27] C. Zhang, Z. Huang, M. H. Ang, and D. Rus, “Lidar degradation quantification for autonomous driving in rain,” in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021, pp. 3458–3464. DOI: 10.1109/IROS51168.2021.9636694.

[28] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[29] H.-B. Mokrane, P. De Souza, P. Nordqvist, and N. Roy, “Modelling of lidar sensor disturbances by solid airborne particles,” May 2021.

[30] F. Matos, J. Bernardino, J. Durães, and J. Cunha, “A survey on sensor failures in autonomous vehicles: Challenges and solutions,” *Sensors*, vol. 24, no. 16, p. 5108, Aug. 2024, ISSN: 1424-8220. DOI: 10.3390/s24165108. [Online]. Available: http://dx.doi.org/10.3390/s24165108.

[31] K. Ebadi, L. Bernreiter, H. Biggie, G. Catt, Y. Chang, A. Chatterjee, C. E. Denniston, S.-P. Deschênes, K. Harlow, S. Khattak, L. Nogueira, M. Palieri, P. Petráček, M. Petrlík, A. Reinke, V. Krátký, S. Zhao, A.-a. Agha-mohammadi, K. Alexis, C. Heckman, K. Khosoussi, N. Kottege, B. Morrell, M. Hutter, F. Pauling, F. o. Pomerleau, M. Saska, S. Scherer, R. Siegwart, J. L. Williams, and L. Carlone, “Present and future of slam in extreme environments: The darpa subt challenge,” *IEEE Transactions on Robotics*, vol. 40, pp. 936–959, 2024, ISSN: 1941-0468. DOI: 10.1109/tro.2023.3323938. [Online]. Available: http://dx.doi.org/10.1109/TRO.2023.3323938.

[32] A. Kyuroson, N. Dahlquist, N. Stathoulopoulos, V. K. Viswanathan, A. Koval, and G. Nikolakopoulos, “Multimodal dataset from harsh sub-terranean environment with aerosol particles for frontier exploration,” in *2023 31st Mediterranean Conference on Control and Automation (MED)*, IEEE, Jun. 2023, pp. 716–721. DOI: 10.1109/med59994.2023.10185906. [Online]. Available: http://dx.doi.org/10.1109/MED59994.2023.10185906.

[33] T. G. Phillips, N. Guenther, and P. R. McAree, “When the dust settles: The four behaviors of lidar in the presence of fine airborne particulates,” *Journal of Field Robotics*, vol. 34, no. 5, pp. 985–1009, Feb. 2017, ISSN: 1556-4967. DOI: 10.1002/rob.21701. [Online]. Available: http://dx.doi.org/10.1002/rob.21701.

[34] P. Li, Y. Pei, and J. Li, “A comprehensive survey on design and application of autoencoder in deep learning,” *Applied Soft Computing*, vol. 138, p. 110 176, 2023, ISSN: 1568-4946. DOI: https://doi.org/10.1016/j.asoc.2023.110176. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1568494623001941.

[35] S. Rayana, *Odds library*, 2016. [Online]. Available: https://odds.cs.stonybrook.edu.

[36] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998, ISSN: 0018-9219. DOI: 10.1109/5.726791. [Online]. Available: http://dx.doi.org/10.1109/5.726791.

[37] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, *Mobilenets: Efficient convolutional neural networks for mobile vision applications*, 2017. DOI: 10.48550/ARXIV.1704.04861. [Online]. Available: https://arxiv.org/abs/1704.04861.

[38] X. Zhang, X. Zhou, M. Lin, and J. Sun, “Shufflenet: An extremely efficient convolutional neural network for mobile devices,” in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, IEEE, Jun. 2018. DOI: 10.1109/cvpr.2018.00716. [Online]. Available: http://dx.doi.org/10.1109/CVPR.2018.00716.

[39] C. E. Metz, “Basic principles of roc analysis,” in *Seminars in nuclear medicine*, Elsevier, vol. 8, 1978, pp. 283–298.

[40] E. Calikus, S. Nowaczyk, and O. Dikmen, “Context discovery for anomaly detection,” *International Journal of Data Science and Analytics*, vol. 19, no. 1, pp. 99–113, Jun. 2024, ISSN: 2364-4168. DOI: 10.1007/s41060-024-00586-x. [Online]. Available: http://dx.doi.org/10.1007/s41060-024-00586-x.

[41] G. O. Campos, A. Zimek, J. Sander, R. J. G. B. Campello, B. Micenková, E. Schubert, I. Assent, and M. E. Houle, “On the evaluation of unsupervised outlier detection: Measures, datasets, and an empirical study,” *Data Mining and Knowledge Discovery*, vol. 30, no. 4, pp. 891–927, Jan. 2016, ISSN: 1573-756X. DOI: 10.1007/s10618-015-0444-8. [Online]. Available: http://dx.doi.org/10.1007/s10618-015-0444-8.

[42] V. Raghavan, P. Bollmann, and G. S. Jung, “A critical investigation of recall and precision as measures of retrieval system performance,” *ACM Transactions on Information Systems*, vol. 7, no. 3, pp. 205–229, Jul. 1989, ISSN: 1558-2868. DOI: 10.1145/65943.65945. [Online]. Available: http://dx.doi.org/10.1145/65943.65945.

[43] E. Kreyszig, K. Stroud, and G. Stephenson, “Advanced engineering mathematics,” *Integration*, vol. 9, no. 4, p. 1014, 2008.

List of Figures

List of Tables