אוניברסיטת בן-גוריון בנגב

Ben-Gurion University of the Negev

הפקולטה למדעי ההנדסה

המחלקה להנדסת חשמל ומחשבים

Faculty of Engineering Science

Dept. of Electrical and Computer Engineering

Fourth Year Engineering Project

Final Report

Intrusion Detection System in CAN-BUS

Project Number:	**p-2025-118**
Student #1:	Name: Omri Asulin ID: 316225036
Student #2:	Name: Saar Tsadik ID: 312420201
Supervisors:	Prof. Michael Segal
Submitting Date:	27.07.2025

Abstract

I

Modern vehicles rely heavily on the CAN-BUS protocol for communication between Electronic Control Units (ECUs), yet this protocol lacks encryption and authentication, making it vulnerable to various cyber-attacks such as Denial-of-Service (DoS), spoofing, and fuzzing. These attacks can be launched through both wired and wireless interfaces, threatening the safety and reliability of in-vehicle systems. While traditional machine learning approaches have been applied to intrusion detection, they often require large labeled datasets and struggle to detect previously unseen (zero-day) attacks. To address these limitations, our approach focuses on learning the normal patterns of CAN traffic in an unsupervised manner using a compact and efficient neural architecture. By modeling temporal and spatial dependencies within the CAN messages, the system can highlight irregularities that deviate from expected behavior, even when attack types are previously unknown or unlabeled. A structured latent space enables the system to capture key distinctions in message flow, supporting robust detection of both known and novel threats. The model's lightweight design allows for fast inference and low computational overhead, making it practical for real-time deployment in embedded automotive environments.

Keywords: CAN-BUS, Intrusion Detection System (IDS), Cybersecurity, Unsupervised Learning, Neural Architecture, Zero-Day Attacks, Automotive Security, Real-Time Detection, Deep Learning, Adversarial Autoencoder, ECUs, Embedded Systems.

II

רכבים מודרניים מסתמכים במידה רבה על פרוטוקול CAN-BUS לצורך תקשורת בין יחידות הבקרה האלקטרוניות (ECUs), אך פרוטוקול זה חסר מנגנוני הצפנה ואימות, מה שהופך אותו לפגיע למתקפות סייבר שונות כגון מתקפות מניעת שירות (DoS), זיוף (spoofing) התקפת טשטוש (fuzzing). מתקפות אלו יכולות להתבצע הן דרך ממשקים חוטיים והן דרך ממשקים אלחוטיים, ועלולות לאיים על הבטיחות והאמינות של מערכות הרכב. למרות שנעשו ניסיונות ליישם שיטות למידת מכונה מסורתיות לזיהוי חדירות, שיטות אלו דורשות לרוב מערכי נתונים גדולים ומתויגים, ומתקשות בזיהוי מתקפות חדשות ולא מוכרות.כדי להתמודד עם מגבלות אלו, הגישה שלנו מתמקדת בלמידה בלתי מפוקחת של דפוסי התעבורה התקינה ברשת CAN תוך שימוש בארכיטקטורה נוירונית קומפקטית ויעילה. באמצעות מידול של קשרים מרחביים וזמניים בין ההודעות, המערכת מסוגלת לזהות חריגות אשר סוטות מההתנהגות הצפויה גם כאשר סוגי המתקפות אינם מוכרים או מסומנים מראש. מרחב סמוי מובנה מאפשר למערכת לזהות הבדלים מהותיים בזרימת ההודעות, ולתמוך בזיהוי חזק של איומים ידועים וחדשים כאחד. העיצוב הקל של המודל מאפשר הסקה מהירה ודרישות חישוביות נמוכות, מה שהופך אותו מתאים לפריסה בזמן אמת בסביבות רכב משובצות.

1. Introduction to the Project

In recent years, the growing complexity of modern vehicles has introduced both innovation and new vulnerabilities. A foundational element in vehicle communication is the Controller Area Network (CAN-BUS), a protocol that facilitates reliable and low-latency data exchange between Electronic Control Units (ECUs). Despite its efficiency and widespread adoption, CAN-BUS was developed without inherent security mechanisms, making it highly susceptible to malicious cyber-attacks such as Denial-of-Service (DoS), spoofing, and fuzzing.These vulnerabilities pose serious risks, particularly as vehicles become more interconnected and software driven. Intrusions into the CAN-BUS can affect safety-critical functions, compromising not only system integrity but also passenger safety. As automotive technology advances, the need for intelligent, adaptive cybersecurity solutions becomes increasingly urgent. Our project aims to address this challenge by designing an Intrusion Detection System (IDS) specifically tailored for CAN-BUS environments. Leveraging both statistical feature engineering and deep learning techniques, our system is capable of detecting anomalous behavior in CAN traffic without relying on labeled attack data. Instead, it learns patterns of normal vehicle behavior in an unsupervised manner, enabling it to flag deviations that may indicate the presence of cyber threats.

1.1. Related Work

- **Statistical Characteristics-Based Intrusion Detection System (SC-IDS):**
  This method detects anomalies by calculating statistical baselines over sliding windows of CAN traffic. Significant deviations are flagged as suspicious. While SC-IDS performs well across various attacks, its accuracy may drop under dynamic driving conditions.

- **Hamming Distance and Interpacket Timing (Taylor et al.):**

  This approach uses Hamming distance and timing analysis to detect spoofing and injection attacks. It performs well but needs large training data and may produce false positives with noisy inputs.

- **Clock-Based Intrusion Detection System (CIDS)**:
  CIDS detects ECU impersonation by analyzing clock skew as a digital fingerprint. It’s effective but relies on stable hardware timing, which may vary in different conditions.

- **Entropy-Based Intrusion Detection System:**
  This method tracks the entropy of CAN ID distributions over time, which remains stable under normal conditions. Large deviations may signal attacks like flooding or

spoofing. While lightweight and effective for high-volume anomalies, it struggles with stealthy or low-rate attacks.

These methods have significantly contributed to the understanding of CAN-BUS vulnerabilities, yet most depend on fixed thresholds, supervised training, or extensive calibration. Furthermore, their performance may degrade when exposed to novel or unseen attacks.

1.2. Advantages of Our Method Compared to Existing Methods

Our solution introduces several technical innovations and architectural choices that offer clear advantages over existing CAN-BUS intrusion detection methods By leveraging an Adversarial Autoencoder (AAE) and transforming CAN frames into 2-channel images.

- **Unsupervised Learning Using Only Normal Data:**
  Unlike many existing systems that require labeled attack data, our approach trains the model exclusively on normal CAN traffic. This enables the system to detect previously unseen attacks based on deviations from learned behavior patterns, making it more resilient to emerging or zero-day threats and practical for real-world deployment.

- **2-Channel Image Encoding of CAN Messages :**
  Each CAN frame is transformed into a 2-channel image that encodes both the identifier (ID) bits and data bytes. This novel representation allows convolutional networks to extract both spatial and semantic relationships within the message structure something traditional flat feature vectors cannot achieve.

- **Dual Discriminator Framework for Latent Space Regularization :**
  The AAE architecture integrates two discriminators: one focused on aligning the latent representation with a predefined prior distribution, and another on maintaining the fidelity of the reconstructed outputs. This dual mechanism improves the model's ability to distinguish anomalies from subtle variations in normal traffic, without being overly sensitive to noise.

- **Designed for Scalability and Real-World Integration:**
  By eliminating the dependence on labeled data and adopting compact representations, the system is well-suited for on-board automotive deployment. Its lightweight processing and modular structure support integration with existing electronic control units (ECUs), making it scalable across diverse vehicle platforms and network conditions.

- **Using Only the Last Payload in the Sliding Window**

  In our preprocessing pipeline, each 29-message window is transformed into a 32×32×2 image by encoding all CAN IDs into the first channel and only the final message's 64-bit payload into the second. This design reflects a core assumption in real-time embedded systems: the last message often represents the system's response to prior inputs, capturing the causal structure where earlier messages establish context and the last encodes output. Including all 29 payloads would increase dimensionality and disrupt spatial structure, reducing learning efficiency. Empirically, attack patterns such as fuzzy or malfunction injections tend to manifest strongly in the final message. Thus, using only the last payload yields a compact, semantically aligned, and statistically meaningful representation ideal for convolutional learning.

1.3. Project Goal

The primary goal of this project is to enhance the detection of message injection attacks in in-vehicle Controller Area Network (CAN) systems using a lightweight and efficient deep learning approach. The CAN protocol, lacking built-in security, is vulnerable to cyberattacks such as spoofing, DoS, and fuzzy attacks, which pose serious safety risks to modern autonomous and connected vehicles. Success for this project is measured by the ability to accurately detect both known and unknown attacks in real-time, even with limited labeled data. This will be achieved by developing a semi-supervised Convolutional Adversarial Autoencoder (CAAE) model that learns distinguishing features of malicious behavior from raw CAN IDs. Evaluation will focus on achieving a high F1 score and low error rate across varied attack types

1.4 CAN Bus Architecture and Vulnerabilities

The Controller Area Network (CAN) bus is the core communication backbone between the various Electronic Control Units (ECUs) within a modern vehicle. Designed for real-time broadcast-based messaging, it provides efficient, flexible, and scalable data exchange, where

each message is identified by a unique CAN ID. Due to the absence of built-in encryption and authentication, the CAN protocol is inherently vulnerable to various types of cyberattacks. Among the most critical are:

Denial-of-Service (DoS): Exploits the message prioritization mechanism by flooding the bus with high-priority messages, effectively blocking legitimate traffic.

Spoofing Attacks: The attacker injects malicious messages using valid CAN IDs to impersonate trusted ECUs, manipulating vehicle behavior.

Fuzzy Attacks: Random or malformed messages are injected into the network, often causing unpredictable behavior or system malfunctions.

These attack types, combined with easy physical or wireless access to the in-vehicle network, highlight the urgency of robust intrusion detection systems (IDS) tailored to automotive environments.

2. Data Understanding and Preprocessing

In this stage, we focused on understanding the structure of raw CAN bus data and transforming it into a format suitable for deep learning. This involved extracting meaningful features, cleaning and normalizing diverse datasets, and converting message sequences into structured image-like inputs using a sliding window approach.

2.1. Structure and Characteristics of CAN Bus Data

The Controller Area Network (CAN) bus is a communication protocol used in modern vehicles to enable electronic control units (ECUs) to communicate without a central host. Each CAN message consists of a timestamp, CAN ID, data length code (DLC), and up to 8 bytes of data. These messages are broadcast in a sequential, high-frequency stream.

Key structural elements include:

CAN ID: A hexadecimal identifier that can be up to 29 bits (extended format), representing the source or type of message.

Timestamp: The time at which the message was sent, used for timing analysis.

DLC: Specifies the number of bytes in the message (0–8).

DATA[0]–DATA[7]: The actual payload of the message, each byte typically in hex.

In the raw datasets, each row represents a single CAN message. Attack types (e.g., Fuzzy) are not necessarily distinguishable from normal messages based on any one field, making preprocessing and pattern recognition essential.

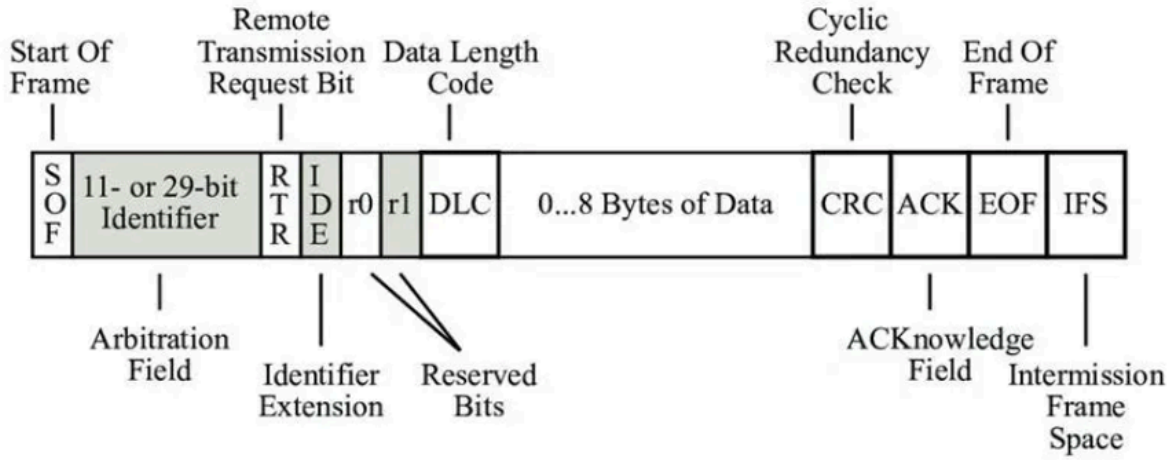


Fig. 1. CAN data frame format

2.2 Feature Extraction: IDs, Bytes, and Timing

To prepare the data for learning, multiple feature extraction strategies were employed:

- **CAN ID Bits**: The 29-bit binary representation of the CAN ID is extracted and used as a compact, structured feature. This helps models learn structural differences in IDs under attack conditions.

- **Byte-Level Features**: Each of the 8 data bytes is converted from hexadecimal to integer format, then optionally unpacked into bit vectors (e.g., 8x8 matrix) to increase feature granularity.

- **Flags/Labels**: The label field is converted to binary 1 for attack, 0 for normal, sometimes requiring the use of fallback heuristics in dirty datasets.

- **Timestamp/Timing**: Though timing features are critical in some IDS models (e.g., Delta T), this model primarily uses static frames, and time is implicitly encoded in the sliding window construction.

2.3 Data Cleaning and Normalization

The datasets came from various sources, vehicle models, and attack types often with inconsistent formatting or missing fields. Cleaning steps included:

Parsing each row and standardizing column names across datasets.

Handling missing values: coercing numeric fields like timestamps and DLC into valid formats, and dropping rows with missing critical fields.

Normalizing label values to binary form using consistent heuristics.

Converting all byte fields to integer form, and optionally to bit-wise representations.

Sorting each dataset chronologically by time stamp to preserve temporal coherence.

This preprocessing was implemented in a robust pre-process function.

2.4 Sliding Window Approach and Image Representation

To capture temporal context and message correlations, the data was processed into fixed-length sliding windows:

Each window spans **29 consecutive CAN messages**, aligning with the 29-bit CAN ID.

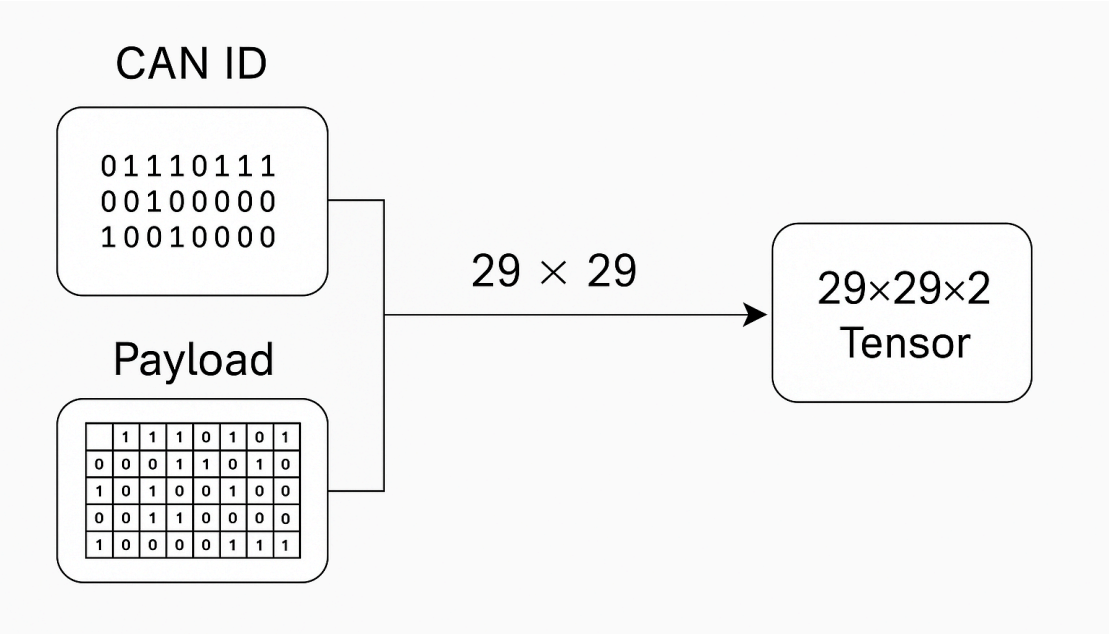
The **CAN ID bit matrix** (29×29) is resized to a 32×32 grayscale image using interpolation.

The **last byte vector** of the window is unpacked into an 8×8 bit matrix and similarly resized to 32×32.

These two matrices are stacked as two channels **final input shape: 32×32×2**.

This innovative representation transforms time-series CAN data into image-like tensors suitable for convolutional neural networks. It allows the encoder to learn spatiotemporal patterns that indicate normal vs. malicious activity.

Fig. 2. Two Channel Vector Input Pipeline



2.5 TFRecord Pipeline: Efficient Data Handling

To handle the large volume and variety of datasets, we transformed the processed data into TFRecord format. This provided multiple advantages:

High throughput with parallel loading, shuffling, and prefetching via the Tf.data API

Low memory footprint, as data is streamed instead of loaded entirely

Reusability and scalability, especially in multi-phase training (train/val/test) and cloud-based setups

Furthermore, time-consuming operations such as binary conversion, reshaping, and resizing were executed once during preprocessing. This significantly accelerated training and ensured consistency during evaluation.

2.6 Dataset Diversity and Realism

To ensure robustness and generalizability of our intrusion detection system, we leveraged a diverse collection of real-world CAN bus datasets originating from different vehicle manufacturers, including Chevrolet, Hyundai, and KIA. Major part of the datasets were curated from the publicly available **Car Hacking Dataset**, which includes a mix of normal driving behavior and a wide variety of cyberattack scenarios.

Dataset Composition

Each dataset contains time-stamped CAN messages consisting of CAN ID, payload bytes, DLC, and ground-truth labels. The attack scenarios include:

Denial-of-Service (DoS) – overloading the network to block legitimate traffic

Spoofing – impersonating a valid CAN ID to issue false commands

Fuzzy – injecting malformed or random messages to disrupt ECUs

Dataset Variability and Authenticity

The inclusion of data from multiple vehicle platforms helps capture protocol differences and message diversity across brands. Furthermore, the datasets reflect real operational conditions, including varying message rates, ID ranges, and payload structures offering a representative training environment for our model.

By utilizing this heterogeneous and labeled dataset, we ensure that our model is trained not only to detect attacks in controlled settings but also to perform reliably across various real-world automotive systems and traffic patterns

3. Modeling & Solution Approach

Outlier detection in automotive CAN bus systems presents unique challenges due to the complexity, high dimensionality, and temporal dependencies of message sequences. Traditional supervised learning methods are limited by the scarcity of labeled attack data, making anomaly detection inherently a semi-supervised problem. Initially, we approached the task with naive optimism, experimenting with classic algorithms like Random Forest and MLP, but quickly realized their limitations in capturing subtle behavioral deviations and unknown attack types leading us toward a more robust, semi-supervised deep learning framework.

3.1 Initial Attempts: Random Forest and MLP

To establish a performance baseline, we implemented both a Multi-Layer Perceptron (MLP) and a Random Forest (RF) for binary classification of CAN messages. These models are well-suited to structured tabular data and worked with our engineered features, which included encoded message IDs, payload bytes, aggregate stats (e.g., sum, mean, zero counts), and timing deltas.

We optimized the MLP using grid search over hidden layer size, regularization, and learning rate. While both models initially performed well especially in detecting attacks the confusion matrix revealed a high false positive rate. The MLP showed strong recall on attacks but misclassified many normal samples, particularly due to overfitting on the Fuzzy dataset. This behavior is typical in unbalanced datasets and highlighted the MLP's inability to capture sequential or contextual dependencies.

These limitations demonstrated that supervised models relying only on statistical features were insufficient. As a result, we shifted toward deep learning architectures—specifically unsupervised and semi-supervised methods—that could better model the structure of normal behavior and generalize to unseen attack types.

Multi-Layer Perceptron (MLP)

A Multi-Layer Perceptron is a feedforward neural network composed of an input layer, one or more hidden layers with nonlinear activation functions, and an output layer. In our case, the MLP took numerical features derived from each CAN message and passed them through fully connected layers. Each neuron in a layer computes a weighted sum of inputs, followed by an activation (commonly ReLU), allowing the network to approximate complex functions. While MLPs are powerful universal approximators, they are inherently limited in modeling temporal dependencies or structural hierarchies within the data.

Random Forest

Random Forest is an ensemble method that constructs multiple decision trees on bootstrapped subsets of the data and aggregates their predictions. It is robust to overfitting in many tabular tasks and interpretable due to feature importance analysis. In our case, Random Forest achieved reasonable accuracy but suffered from similar limitations to MLP: its performance was highly dependent on the distribution of known attacks and lacked the ability to generalize to sequences or rare anomalies. As it operates on individual messages without context, its capacity for anomaly detection in sequential data like CAN logs was inherently constrained.

3.2 Autoencoder Architecture

Autoencoders are unsupervised neural networks designed to learn a compact representation of input data by reconstructing it through two components: an encoder and a decoder. Given an input vector x in R^d, the encoder maps it to a latent representation $z = f_\theta(x)$ in R^k, where k is much smaller than d. The decoder then attempts to reconstruct the original input from the latent space. This process encourages the network to capture the most informative structure in the data distribution. In the context of anomaly detection, the model is trained exclusively on normal data. As a result, it learns to accurately reconstruct normal patterns and produces higher reconstruction errors on anomalous patterns that deviate from the learned manifold.

CAN bus traffic shows regular, structured patterns, while attacks produce out-of-distribution payloads or IDs. We leverage this by detecting anomalies through reconstruction error. A shallow architecture with batch normalization, dropout, and LeakyReLU ensures stable training and handles the binary, sparse CAN data effectively.

The Theory Behind Autoencoders

An autoencoder is a neural network designed to learn a low-dimensional representation of input data by training it to reconstruct the input from a compressed latent code. It consists of two main components: an encoder $f_e\theta(x)$, which maps input data $x \in \mathbb{R}^d$ into a latent vector $z \in \mathbb{R}^k$, and a decoder $g_\theta(z)$, which reconstructs the input as $\hat{x} \in \mathbb{R}^d$. The model is trained to minimize a reconstruction loss, typically the mean squared error $||x - \hat{x}||^2$, so that $\hat{x} = g(f(x))$ approximates x. Conceptually, the autoencoder learns a function that preserves the most essential information of the input through the latent space while discarding noise or redundancy.

The Objective Function

Let $f_{\theta_e}: \mathbb{R}^d \to \mathbb{R}^k$ be the encoder with parameters θ_e, and $g_{\theta_D}: \mathbb{R}^k \to \mathbb{R}^d$ be the decoder with parameters θ_D. The autoencoder aims to minimize the expected reconstruction loss:

$$L_{AE}(\theta_e, \theta_d) = E_x \sim p_{data}(x)[\| x - g_{\theta_d}(f\theta_e(x)) \|^2]$$

This loss function encourages the network to learn latent codes z = f(x) from which the original data x can be reconstructed accurately.

Universal Approximation Assumption

We assume that both encoder and decoder are neural networks with sufficient capacity (depth and width). By the Universal Approximation Theorem, for any continuous function h(x) and $\varepsilon > 0$, there exists a neural network such that:

$$sup_x \in K \| h(x) - \hat{h}(x) \| < \varepsilon$$

This implies that with enough parameters and proper training, the networks f and g can approximate the identity function on the data manifold $M \subset \mathbb{R}^d$, such that: $g(f(x)) \approx x, \ \forall x \in M$

Data Manifold Hypothesis

We assume that high-dimensional data x lie near a low-dimensional manifold M of intrinsic dimension k ≪ d. The encoder learns a mapping f: M → $\mathbb{R}^k$, effectively 'unfolding' the manifold into a linear latent space.

Regularization and Bottleneck

To avoid trivial identity mapping, a bottleneck is imposed: the latent dimension k < d. This constraint forces the network to compress the information and retain only the most relevant features for reconstruction.

Optional regularization terms may be added to the loss, such as:

- L2 regularization on weights: λ // θ // ²
- Sparsity constraints: encouraging many elements of z to be zero
- Denoising: forcing the model to reconstruct clean x from noisy input x̃

Why it Works: Information Preservation

We can interpret the encoder as learning a projection:

$$f(x) = W_e x + b_e$$

$$g(z) = W_d z + b_d$$

The reconstruction becomes:

$$\hat{x} = W_d W_e x + W_d b_e + b_d$$

If $W_d W_e \approx I_d$, then x̂ ≈ x, and the model approximates the identity mapping over the data manifold.

In the non-linear case, this approximation is learned via backpropagation.

Auto Encoders are foundational tools in representation learning, enabling unsupervised feature extraction and dimensionality reduction.

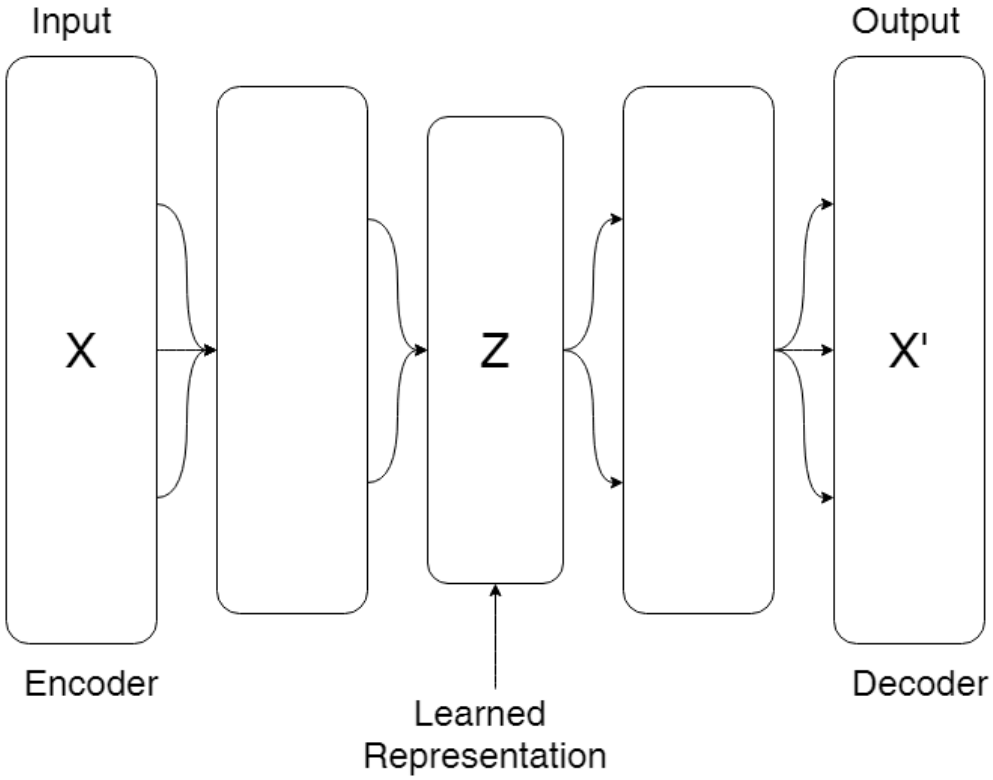


Fig. 3. CAN Auto encoder simple scheme of layers

3.3 Our Auto-Encoder Implementation

The final Simple Autoencoder was trained on balanced datasets of normal and attack samples. Optimal performance was achieved with a 256-dimensional latent space, 1024-unit dense layers, 0.1 dropout, and a learning rate of 0.001. After 25 epochs, validation loss reached 0.0028. Evaluation showed an ROC AUC of 0.9303 and an optimal threshold of 0.0035. While achieving 85% accuracy, the model misclassified 3699 normal samples and missed 2775 attacks—highlighting its sensitivity limitations. These results revealed that relying solely on reconstruction error fails to capture subtle or adversarial anomalies, prompting a shift to adversarial autoencoders for structured latent modeling and improved detection.

See (A),(B),(C) for a detailed breakdown of the model Metrices.

3.4 Adversarial Autoencoder (AAE): Architecture and Rationale

Adversarial Autoencoder (AAE) is a deep generative model that merges the functionality of a traditional autoencoder with the adversarial training process of Generative Adversarial Networks (GANs). This combination offers both robust feature learning and distribution

regularization, making it well-suited for unsupervised anomaly detection in CAN bus systems.

The AAE framework comprises an encoder, a decoder, and at least one adversarial discriminator. The encoder compresses the input data into a latent representation z. The decoder reconstructs the original input from z. During training, AAE alternates between two main phases:

1. Reconstruction Phase: Here, the encoder and decoder are jointly optimized using reconstruction loss, such as Mean Squared Error (MSE), to accurately replicate the original input.

2. Regularization Phase: This phase involves adversarial training. The encoder plays the role of a generator. It produces latent vectors z intended to follow a predefined prior distribution (e.g., standard normal distribution). A discriminator is then trained to distinguish between these generated latent vectors and true samples drawn from the prior. The encoder is subsequently trained to fool the discriminator, encouraging the latent space to conform to the prior distribution.

Mathematically, the adversarial component in AAE aims to minimize the divergence between the aggregated posterior $q(z)$ and the prior $p(z)$.

Let the encoder define the conditional distribution $p(z|x)$, and the decoder defines . The aggregated posterior $p(x|z)$ is defined as:

$$q(z) = \int_x q(z|x)p(x|z)$$

The goal is to match this aggregated posterior to the prior . To achieve this, we set up a minimax game between the encoder (acting as the generator) and the discriminator. The discriminator learns to distinguish between samples drawn from and samples produced by the encoder , while the encoder learns to fool the discriminator.

This game ensures that the encoder produces latent variables that are indistinguishable from the true prior distribution, thus regularizing the latent space. The result is a powerful autoencoder that not only reconstructs well but also structures its latent representation in a meaningful and generative way.

Formally, the GAN component is trained using the following objectives:

Discriminator loss:

$$max_D V(D) = \mathbb{E}_{x \sim p_{data(x)}}[log(D(x))] + \mathbb{E}_{z \sim p_{z(z)}}[log(1 - D(G(z)))]$$

Generator loss:

$$min_G V(G) = \mathbb{E}_{z \sim p\_z(z)} [1 - log(D(G(z)))]$$

In AAE, the encoder replaces the GAN generator, and the discriminator operates on latent vectors z. Training is performed using stochastic gradient descent in mini-batches.

To further enforce structure in the latent space, we use two discriminators:

- One on the latent variable z to match a continuous Gaussian prior.
- One on the label variable y to match a discrete categorical prior (used for semi-supervised conditioning).

The encoder is trained to produce z and y such that both appear to be drawn from their respective priors, according to the discriminators. This adversarial regularization ensures meaningful latent representations and enables detection of out-of-distribution (anomalous) data based on reconstruction error.

See Appendix D for additional GAN architecture explanation.

3.5 AAE-Based Implementation and Evaluation

Our implementation builds upon the dual-discriminator AAE framework described above. The input data is structured as 29x29x2 tensors, representing CAN ID and payload. The encoder maps this input to two outputs: a latent vector z of size 64 and a categorical label y (normal or attack).

The model is trained in three phases within each epoch:

1. Reconstruction phase using reconstruction loss only.
2. Regularization phase using adversarial loss with the latent and label discriminators.
3. Supervised classification phase with cross-entropy loss on labeled examples.

We used dropout with rate 0.15 and ReLU activations for regularization and stability. Optimization was performed using the Adam optimizer with separate learning rates for each phase. A WGAN-GP style gradient penalty was included to stabilize discriminator training and enforce the 1-Lipschitz condition.

Evaluation was conducted by freezing the encoder and decoder, and then computing the reconstruction error on test data. We selected the anomaly detection threshold using Youden's J statistic to balance recall and false positives. Final performance metrics on the test set were as follows:

These results highlight the ability of AAE to generalize to previously unseen attack types while maintaining low false alarm rates.

See Appendix D for a detailed breakdown of the model architecture.

3.6 Final Solution-Convolutional Adversarial Autoencoder 2D Channel:

The Convolutional Adversarial Autoencoder (CAAE) represents a synthesis of convolutional feature extraction and adversarial regularization, forming a robust framework for anomaly detection in sequential data such as CAN bus traffic. Building upon the Adversarial Autoencoder (AAE), the CAAE architecture is designed to address two core challenges in intrusion detection: capturing spatial-temporal patterns in data and minimizing dependence on labeled samples.

At the heart of the CAAE is a convolutional autoencoder, which replaces traditional fully connected layers with convolutional and deconvolutional layers. This design choice exploits the spatial locality of patterns in input data represented as 2D frames for instance, a matrix of stacked binary-encoded CAN IDs allowing the model to learn translation-invariant features indicative of normal and abnormal communication sequences.

3.6.1 Multi-Channel Input Representation

To maximize the CAAE's representational capacity, input frames are extended to multiple channels. Specifically, a two-channel configuration is adopted:

> Channel 1 captures a time window of 29-bit CAN ID sequences, stacked temporally and resized to a 32×32 grid.
>
> Channel 2 encodes the 64-bit payload of the most recent frame in the window, unpacked from raw bytes and similarly resized.

This dual-channel encoding enables the network to jointly analyze protocol-level structure (CAN IDs) and application-level content (payload bits) within a unified input representation. The convolutional architecture is inherently well-suited to processing such multi-dimensional data, allowing filters to discover cross-channel interactions that may signal malicious behavior.

This strategy is scalable and extensible: additional semantic channels such as timing intervals or DLC patterns could be incorporated with minimal architectural changes, further enhancing the model's ability to learn complex, domain-specific correlations.

To introduce structure into the learned latent space, the CAAE integrates two adversarial discriminators:

1. Categorical Discriminator : This network enforces that a portion of the latent representation encodes class-level semantics (e.g., normal vs. attack) by aligning it with a categorical prior distribution, typically Cat(K), where K is the number of classes.

2. Gaussian Discriminator: This discriminator regularizes the remaining latent variables to follow a continuous multivariate Gaussian prior, ensuring smooth and disentangled representations in the latent space.

Training proceeds in three coordinated phases:

Reconstruction Phase: The autoencoder learns to compress and reconstruct input frames using a reconstruction loss, typically Mean Squared Error (MSE), applied to unlabeled data.

Adversarial Regularization Phase: The encoder is trained in tandem with the two discriminators. The discriminators are optimized to distinguish true samples drawn from the prior distributions from encoded samples, while the encoder is trained to "fool" the discriminators. This adversarial process shapes the latent space into structured, prior-conforming regions.

Supervised Phase: A small set of labeled samples is used to guide the class-encoding component of the latent space. Cross-entropy loss is applied to the class vector to ensure discriminative power between normal and anomalous frames.

To stabilize adversarial training, the CAAE employs the Wasserstein GAN (WGAN) objective with a gradient penalty. This formulation replaces the Jensen–Shannon divergence with the Wasserstein distance, leading to smoother gradients and improved convergence. The gradient penalty ensures the 1-Lipschitz constraint required by WGAN by penalizing deviations in the gradient norm from unity.

This semi-supervised architecture allows the CAAE to extract meaningful latent features from unlabeled data while using a limited amount of labeled data to enforce

semantic discrimination. Its convolutional nature makes it particularly well-suited for structured, spatially correlated inputs such as frame-encoded CAN traffic. As a result, the CAAE offers a scalable and generalizable framework for real-time anomaly detection in automotive networks.

3.6.2 Wasserstein GAN with Gradient Penalty (WGAN-GP)

In our CAAE architecture, adversarial regularization is implemented using the WGAN-GP framework, which improves training stability and convergence behavior. Below is a detailed explanation of the key mathematical components used in this loss formulation.

Wasserstein Loss for Discriminator

The primary goal of the discriminator in WGAN is not to classify samples as real or fake but to measure how far the generated data distribution is from the true distribution. It does so by maximizing the difference between the expected outputs for generated (fake) samples and real samples:

$$L_{WGAN} = \mathbb{E}_{y\sim P_{fake}}[D(\hat{Y})] - \mathbb{E}_{y\sim P_{real}}[D(Y')]$$

Where:
- Y: Samples drawn from the distribution
- Y′: Fake/generated samples
- Ŷ: Interpolated samples

This loss approximates the Earth Mover (Wasserstein-1) distance between the real and generated distributions, providing a smoother and more informative gradient for training.

To compute this, interpolated samples y^\hat{y}y^ are defined as:

$$\hat{Y} = \varepsilon Y + (1 - \varepsilon)Y', \; where \; \varepsilon \sim U[0, 1]$$

Gradient Penalty (GP)

To enforce the 1-Lipschitz continuity constraint on D, a gradient penalty term is introduced. This penalty is applied to the interpolated samples $\hat{Y}$ generated above.

$$GP = \lambda \cdot \mathbb{E}_x [(\|\nabla_y D(\hat{Y})\|_2 - 1)^2]$$

Where:
- λ: Gradient penalty coefficient (typically set to 10)
- ∇_y: Gradient with respect to interpolated inputs $\hat{Y}$

Total Discriminator Loss

The total loss for the discriminator becomes the sum of the Wasserstein loss and the gradient penalty:

$$L_D = L_{WGAN} + GP$$

Generator Loss

The generator is trained to maximize the discriminator's output on generated samples, thus minimizing:

$$L_G = -\mathbb{E}_{Y'}[D(Y')]$$

This encourages the generator (or encoder in AAE/CAAE) to produce outputs that lie close to the true data distribution under the Wasserstein metric.

In our model, these loss components are applied independently to both the categorical and Gaussian latent representations. This use of WGAN-GP acts as a strong regularizer, improving the structure and quality of learned latent features.

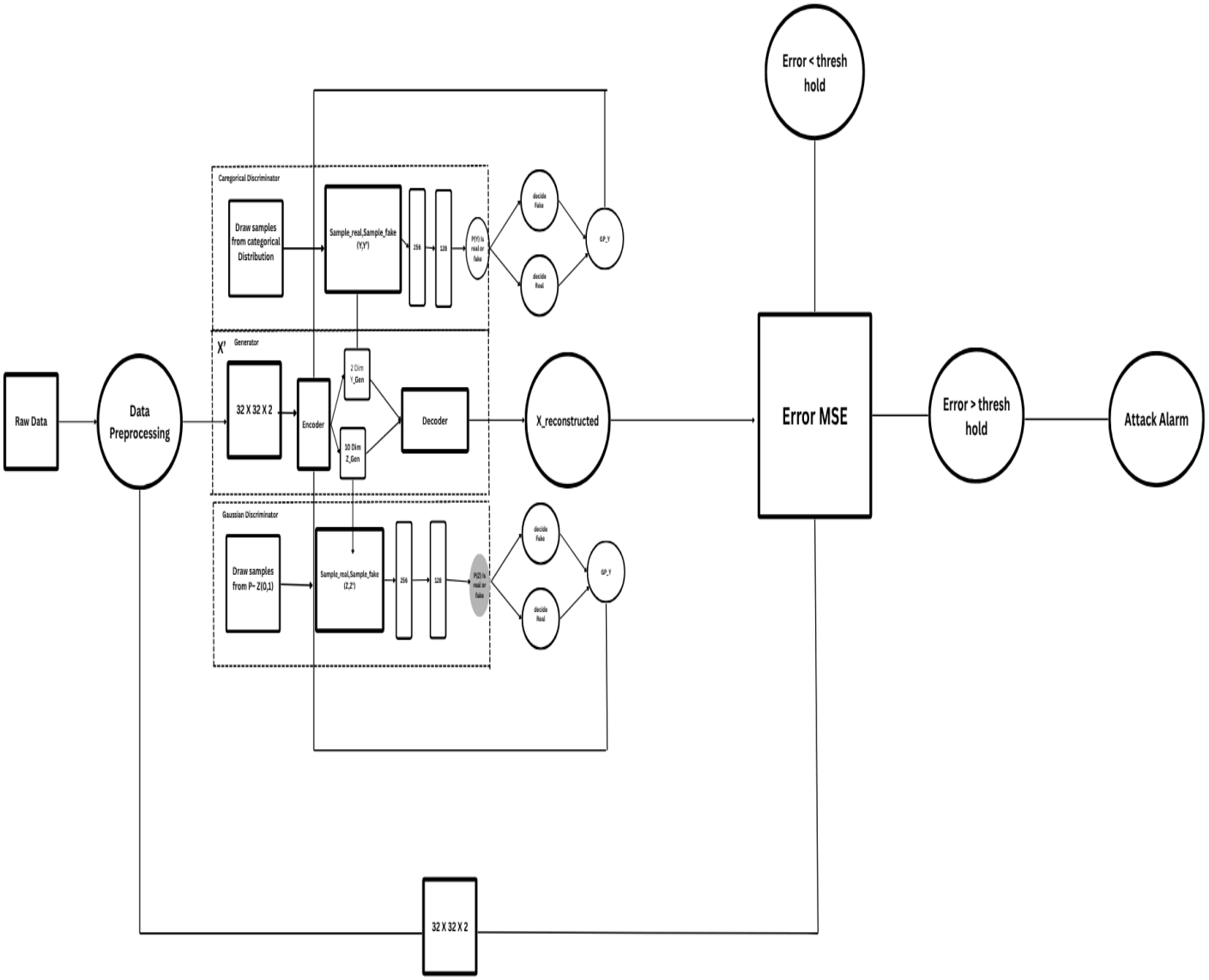


Fig 4 CAEE Architecture

4. Tools

To develop a robust pipeline for anomaly detection in CAN traffic, we employed a set of advanced software tools and machine learning frameworks. The core components of the implementation include Python, TensorFlow, and custom data preprocessing techniques. Each element played a vital role in building, training, and evaluating the Convolutional Adversarial Autoencoder (CAAE) model.

4.1. Python Integration

In our project, Python was used as the central environment for data preprocessing, model design, and evaluation. Its extensive library support enabled streamlined development and efficient handling of structured CAN data.

4.1.1. Python Libraries Used:

pandas: For parsing raw CAN CSV files and handling timestamps, identifiers, and payloads.
numpy: For reshaping arrays and performing numerical computations on binary-encoded data.
opencv (cv2): For resizing and stacking binary matrices to generate two-channel input images.
glob and os: For accessing and organizing datasets across directory structures.
matplotlib.pyplot: For visualizing loss trends, error histograms, and evaluation metrics.
scikit-learn: For computing ROC curves, AUC scores, and classification performance reports.

4.2. TensorFlow Integration

TensorFlow served as the core framework for implementing the Convolutional Adversarial Autoencoder (CAAE). It enabled precise control over the training workflow, model architecture, and GPU-accelerated execution.

4.2.1. TensorFlow Features Used:

tf.keras.Model: For defining the encoder, decoder, and adversarial discriminators as modular components.
tf.data: For building an efficient input pipeline using TFRecord files with support for shuffling, batching, and prefetching.
GradientTape: For implementing a custom training loop that separately optimizes each model component.
TensorFlow Graph Execution: For compiling training steps into optimized computation graphs.

WGAN-GP Training: For stabilizing adversarial updates using the Wasserstein loss with gradient penalty.

By leveraging Python for data manipulation and TensorFlow for deep learning model development, our project was equipped with the computational tools necessary to perform high-fidelity anomaly detection across structured CAN traffic inputs.

4.3. Data Preprocessing and Encoding

CAN traffic logs, provided in CSV format, were parsed and transformed using a custom pipeline developed in Python. This step was essential to convert the raw hexadecimal values into structured tensors suitable for convolutional input.

The preprocessing involved:

Converting hexadecimal CAN IDs into binary vectors of fixed length (29 bits).

Extracting the payload from each frame and reshaping the last message in the window into an 8×8 binary matrix.

Stacking both the ID and payload as two separate channels, resulting in a 32×32×2 image representation per sample.

The use of OpenCV ensured that each channel was properly resized and aligned, while NumPy and Pandas handled the core numerical transformations.

5. Simulation Setup

5.1 Simulation Scenarios

To validate the robustness and generalizability of our anomaly detection models, we conducted a series of six simulation scenarios using combinations of different CAN bus datasets and model configurations. These simulations test how the models respond to varying levels of data diversity and size, as well as how well they generalize across vehicle types.

5.1.1 Single-Vehicle Dataset Evaluation

Description:
The model was trained and validated on CAN data from a single vehicle type. This setup helps us verify the model's ability to learn the typical structure of benign vs. malicious traffic within a consistent environment.

Purpose:
To establish a baseline performance and assess how well the autoencoder, AAE, and CAAE can distinguish anomalies using only homogeneous data.

5.1.2 Multi-Vehicle Dataset Evaluation

Description:
This scenario extends the training to two distinct vehicle types, combining their CAN traffic patterns into one training set. Evaluation was done on both combined and individual subsets.

Purpose:
To test how well the model generalizes when exposed to varied CAN structures, simulating real-world scenarios where ECU architectures may differ slightly between vehicle models.

5.1.3 Full Dataset (Three-Vehicle) Generalization

Description:
The model was trained on a merged dataset combining three vehicle types. This configuration represents the most diverse training set in terms of message patterns, ID frequency, and data distributions.

Purpose:
To simulate deployment in environments with mixed-fleet vehicles and verify that the model still maintains clear decision boundaries between normal and attack patterns across diverse inputs.

5.1.4 Cross-Vehicle Generalization

Description:
In this simulation, the model was trained on one or two vehicle types and evaluated on a third unseen vehicle type.

Purpose:
To test the model's capacity for cross-domain generalization — whether it can detect anomalies in CAN logs from new vehicle models without retraining.

5.1.5 Large Dataset Stress Test

Description:
A large, labeled dataset from a public CAN attack dataset (e.g., Combined Car Hacking Dataset) was used to train and validate the model under high-volume conditions.

Purpose:
To push the model's learning capacity to its limit and verify convergence when exposed to dense, diverse, and long-duration CAN data, ensuring the system scales with real-world data volumes.

5.1.6 Minimal-Data Robustness Evaluation

Description:
Following the success of the large-scale training, we conducted a series of experiments where the model was trained on progressively smaller subsets of the data (down to 25% of the original).

Purpose:
To evaluate the minimum data requirements for the AAE and CAAE models to maintain reliable performance. This simulates real-world constraints where limited data may be available for certain vehicles.

By running these six simulation scenarios, we thoroughly examined the behavior of our autoencoder-based models across different levels of heterogeneity, volume, and domain shift. This simulation framework provides a comprehensive foundation for the evaluation of both in-domain and out-of-domain anomaly detection capabilities.

5.2. Noise Application

To evaluate the robustness and generalization ability of our Conv-AAE model, we inject Gaussian noise into the input data during training. Specifically, for each training sample, we apply zero-mean Gaussian noise with a small standard deviation ($\sigma = 0.01$) to the entire input feature tensor, which is a $32 \times 32 \times 2$ representation of each CAN message window.

Purpose:
This noise simulates real-world perturbations or signal corruption that may occur in practical CAN bus environments (e.g., electromagnetic interference, timestamp jitter, or sensor noise). The objective is to train the encoder to learn meaningful, noise-invariant latent representations and to encourage the decoder to reconstruct clean inputs from noisy observations.

6. Future Features

This project introduces two core features aimed at improving the robustness and adaptability of our CAAE-based CAN Intrusion Detection System (IDS). These additions mark important steps toward a more generalizable and interpretable model for automotive cybersecurity, especially in the context of connected and autonomous vehicles.

6.1. Multi-Source Training and Generalization

Our model supports training on multiple CAN datasets from different vehicle types (e.g., Chevrolet Spark, Hyundai Sonata, KIA Soul), allowing the encoder to learn attack-invariant representations across various manufacturers. This cross-domain learning improves real-world performance, where vehicle-specific patterns can vary significantly.

By combining sources during training, the model maps normal behavior into a shared latent space, while attack patterns consistently appear as outliers. Our results confirm that adversarial regularization on both latent vector z and class vector y supports stable generalization without overfitting.

6.2. Latent Expansion and Semantic Structure

The model's latent space is expanded and regularized via dual discriminators for z and y, encouraging structured representations. This allows both compact encoding and semantic clustering of traffic behaviors supporting potential classification of operational states or message categories. Additionally, the use of 32×32×2 image-like inputs enables visual traceability of anomalies, supporting explainability in safety-critical applications. This structure could support integration with explainable AI or dynamic adaptation mechanisms.

6.3. Future Work

Future developments may include: lightweight deployment on ECUs via TensorFlow Lite, unsupervised tuning on unseen vehicle types, multimodal integration (e.g., with IMU, radar), and online adversarial retraining. These directions will be vital in scaling IDS to evolving, large-scale autonomous fleets.

7 . Before the Simulations

Before training and evaluating the CAAE model, we selected key parameters expected to influence anomaly detection performance. Each was chosen based on theoretical impact and tested under controlled simulation runs.

7.1. Parameters Evaluated

Latent Dimension (z): Controls the compression capacity of the encoder. Higher dimensions may capture more features but risk overfitting.
Hypothesis: Moderate values (64–128) provide optimal balance between expressiveness and regularization.

Discriminator Strength λ_{gp}: Governs the weight of the gradient penalty in Wasserstein loss.
Hypothesis: A well-tuned penalty helps enforce latent distribution matching without destabilizing training.

Dropout Rate: Applied across encoder/decoder layers to prevent overfitting.
Hypothesis: Light dropout (e.g., 0.2) improves generalization while preserving reconstruction quality.

Training Dataset Composition: Involves using one or multiple vehicle datasets during training.
Hypothesis: Mixed-source training improves generalizability across vehicle types and unseen patterns.

Noise Injection in Input: Adds small Gaussian noise to inputs during training.
Hypothesis: Helps improve robustness and simulates real-world sensor variation.

Adversarial Loss Weighting: Balances the reconstruction loss and latent space regularization.
Hypothesis: Proper tuning improves anomaly separation in both pixel and latent domains.

7.2. Metrics for Evaluation

To assess the performance of our model, we used several key evaluation metrics:

Mean Squared Error (MSE): Measures reconstruction error between original and decoded CAN message images.

ROC AUC: Evaluates overall discrimination ability between normal and attack messages.

Optimal Threshold (Youden's J): Selected to maximize sensitivity and specificity.

Confusion Matrix: Captures true positives, false positives, true negatives, and false negatives.

Classification Report: Includes precision, recall, and F1-score for both normal and attack classes.

These metrics were calculated on the test set using the reconstruction error as the anomaly signal. For detailed numbers and visualizations (e.g., ROC curve, confusion matrix heatmap, and error distributions).

8. Experiments and Results

We evaluated our models on multiple CAN bus datasets to measure detection performance across different vehicle types and attack scenarios. Experiments included diverse attacks (DoS, Fuzzy, RPM, Gear) and were designed to test generalization under realistic driving conditions. Results are reported for both large and sparse datasets.

8.1 Results – Convolutional Adversarial Autoencoder (CAAE)

The CAAE model was evaluated on the largest available dataset, which contains a diverse range of normal and attack scenarios from the same vehicle. After 30 epochs of training, the model reached a stable validation reconstruction error of **0.0072**, indicating that the decoder could accurately reconstruct benign CAN frames from the compressed latent representation.

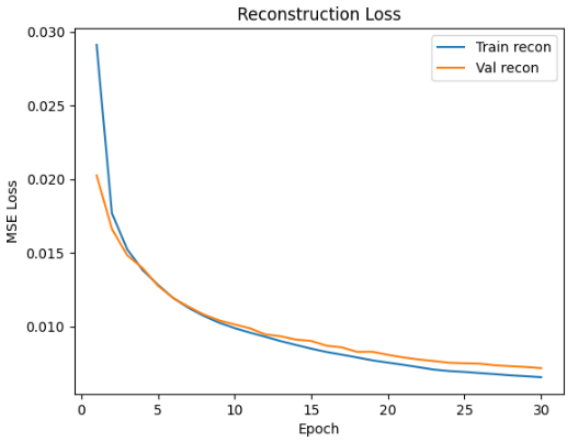
Performance evaluation was carried out on the test set, yielding an ROC AUC of 0.987, confirming the model's ability to differentiate between normal and anomalous messages with high precision. The optimal decision threshold was automatically selected at **0.0152** using Youden's J statistic.

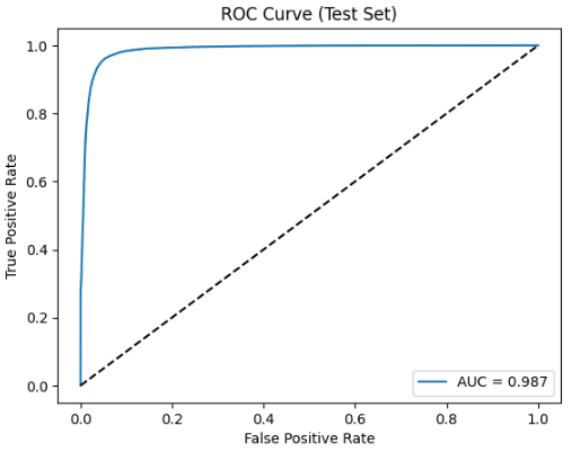
At this threshold, the True Positive Rate reached **96.2%**, while the False Positive Rate remained relatively low at **5.4%**, striking a favorable trade off between sensitivity and specificity. The confusion matrix showed that **87.4%** of all attack samples were correctly classified, with only a small number of misclassifications

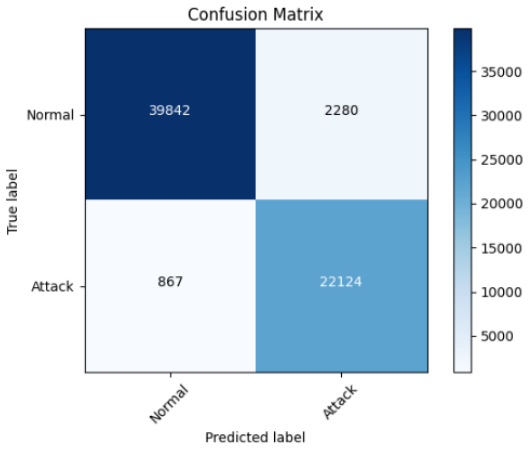
The resulting classification report further confirmed robust detection:

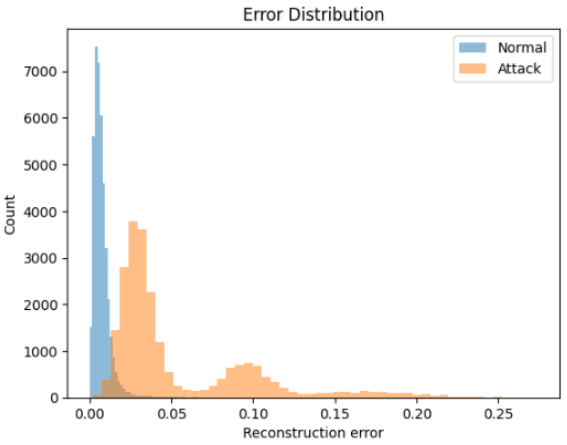
Overall accuracy across the full test set (65,113 samples) reached **95%**, and the weighted F1-score was **95%**. These metrics confirm the strength of the CAAE architecture in learning generalized latent features that separate clean and anomalous behavior without overfitting to specific message types.

Class	Precision	Recall	F1-Score
Normal	98%	95%	96%
Attack	91%	96%	93%









8.2. Simulation 2 – Cross-Vehicle Generalization (CAAE Model)

This simulation aimed to evaluate the generalization capability of our CAAE model when applied to a heterogeneous dataset combining data from three different vehicles. The primary goal was to test robustness across varying message patterns and payload structures typical of distinct ECUs and vehicle models.

Methodology:

We merged the largest datasets into one unified training corpus, ensuring a balanced representation of both normal and attack traffic. The model was trained on over 215,000 records across 30 epochs, using the same architecture and hyperparameters optimized in prior

experiments. All performance metrics were calculated using a fixed detection threshold determined from the validation set.

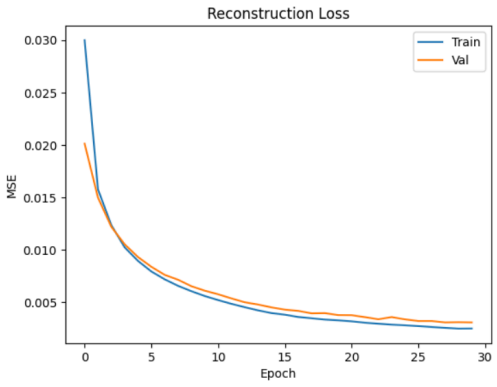
Analysis and Conclusion:

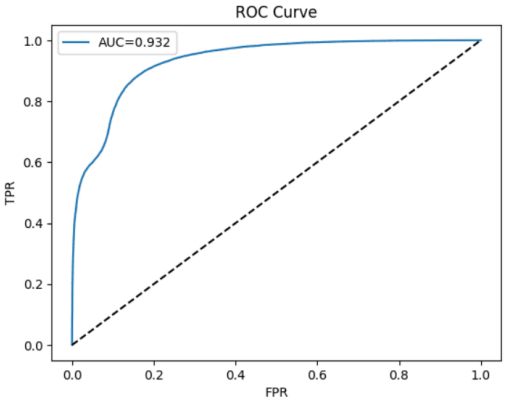
Despite the increased challenge posed by mixing data from different vehicles, the CAAE maintained a strong performance. Although the AUC and precision scores were lower compared to single-vehicle evaluations, the model successfully generalized across unseen domains, achieving high recall and a robust detection capability. This confirms that the convolutional adversarial structure helps capture shared statistical regularities across heterogeneous CAN environments, making our approach viable

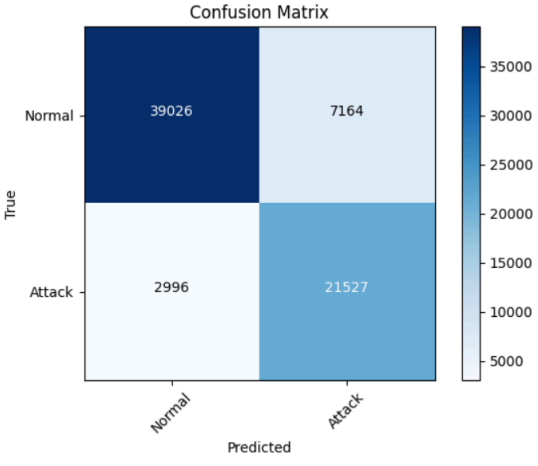
Results:

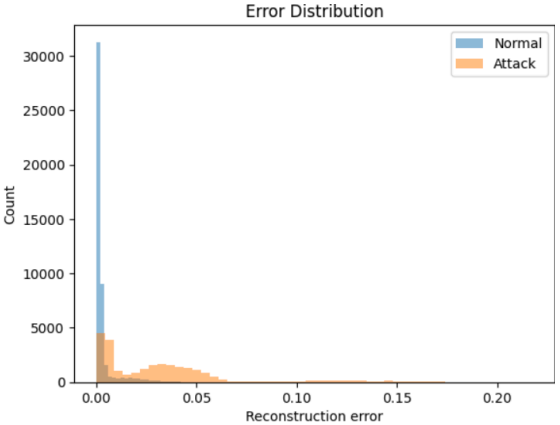
Total samples: 215,424 ROC AUC: 0.9316 Detection Threshold: 0.003435 TPR: 0.878 FPR: 0.155 Accuracy: 86%

Class	Precision	Recall	F1-Score
Normal	93%	84%	88%
Attack	75%	88%	81%









8.3. Simulation 3 – Single-Vehicle Evaluation (AAE Model)

Objective:

This simulation evaluated the performance of our AAE model when trained and tested on the largest dataset derived from a single vehicle. The aim was to measure the model's anomaly detection capability under ideal conditions, where the distribution of normal and attack traffic originates from the same CAN environment.

Methodology:

We utilized the most comprehensive dataset captured from a single vehicle, comprising over 65,000 labeled samples. The AAE model was trained using a latent adversarial structure and reconstruction-based loss over 50 epochs. A detection threshold was selected based on validation performance, and the final metrics were computed on the test set.

Analysis and Conclusion:

The AAE model demonstrated excellent detection performance when evaluated on a consistent and vehicle-specific dataset. The high AUC and F1 scores for both classes indicate that the model effectively captured temporal and structural patterns of benign and malicious CAN traffic. These results validate the strength of adversarial autoencoding in learning compact and discriminative latent representations, particularly in environments with consistent behavior. This benchmark will serve as a reference point for assessing model degradation under cross-domain and limited-data conditions in subsequent simulati**ons.**

Results:

Total samples: 65,113 ROC AUC: 0.9820 Detection Threshold: 0.015617 TPR: 0.948 FPR:0.058 Accuracy: 94%

Class	**Precision**	**Recall**	**F1-Score**
Normal	97%	94%	96%
Attack	90%	95%	92%

8.4. Simulation 3- Mid-Size Dataset Evaluation (CAAE Model)

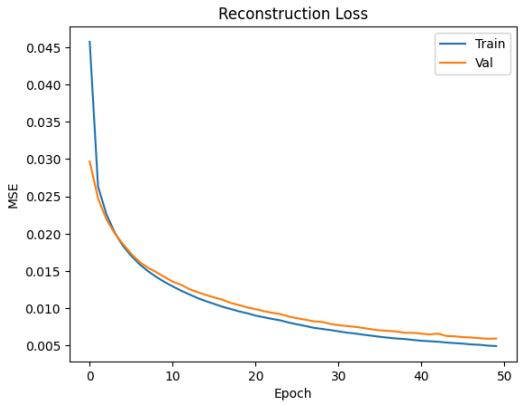
Objective:
This simulation evaluated the performance of the CAAE model on a mid-size dataset derived from a single vehicle. The aim was to assess how well the model performs in a more controlled yet realistic setting, with moderate variability in CAN traffic and attack behaviors.

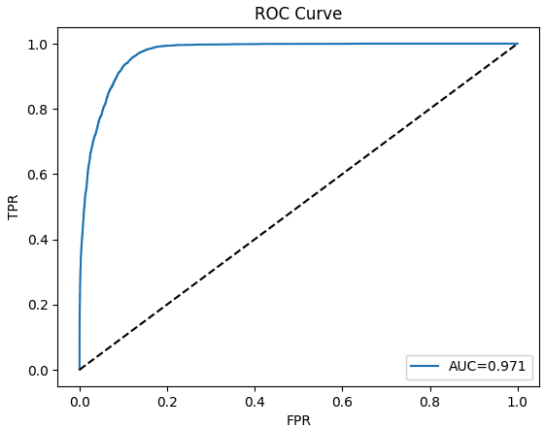
Methodology:
The CAAE was trained on approximately 22,000 samples, maintaining the same convolutional adversarial architecture used in previous simulations. The training ran for 50 epochs, and evaluation was performed on a held-out test set. A fixed threshold was chosen based on the validation set to classify normal versus anomalous frames.

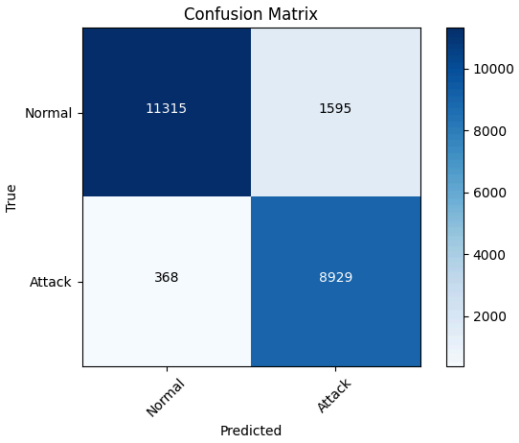
Analysis and Conclusion:
The CAAE model achieved excellent performance on the mid-size dataset, with a high AUC of 0.97 and strong balance between precision and recall. The error distribution and ROC curve indicate that the adversarial training contributed significantly to distinguishing subtle anomalies in the data. These results confirm that the CAAE remains effective when applied to a reduced but representative dataset, suggesting suitability for embedded deployment on individual vehicles with limited memory or compute.

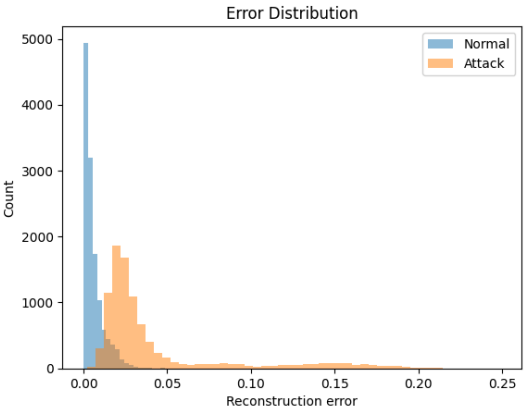
Results:

Class	Precision	Recall	F1-Score
Normal	97%	88%	92%
Attack	85%	96%	90%









8.5. Simulation 4 Mid-Sized Dataset Evaluation (AAE Model)

This simulation evaluated the performance of the standard Adversarial Autoencoder (AAE) on a mid-sized dataset. The goal was to assess the model's accuracy and robustness when trained on moderately rich data, without the convolutional enhancements used in the CAAE.

Methodology:

The dataset contained 23,199 labeled samples, composed of both normal and attack traffic. The AAE architecture remained consistent with previous experiments, utilizing dense layers for encoding and decoding, and adversarial regularization on the latent space. The model was

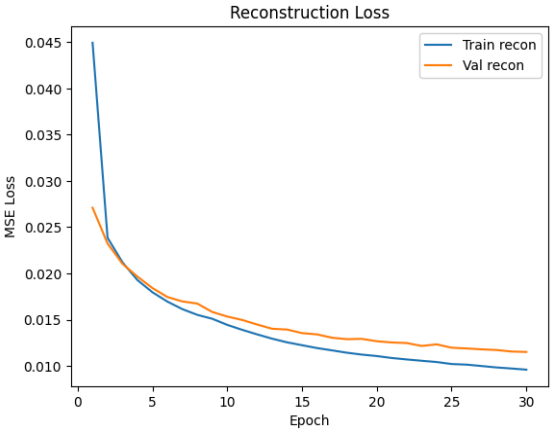
trained over 30 epochs, and the detection threshold was selected based on the optimal point on the ROC curve.

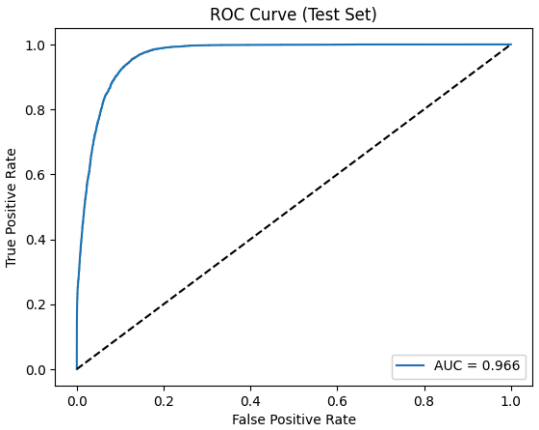
Analysis and Conclusion:

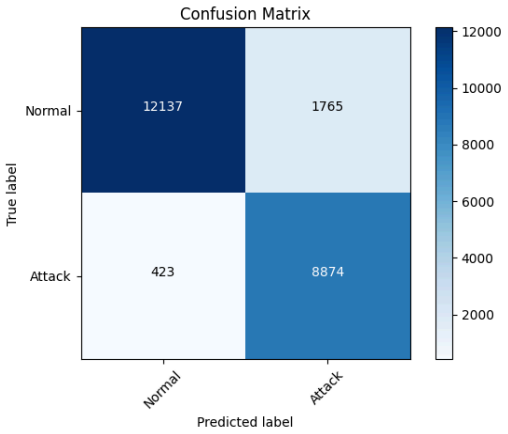
The AAE demonstrated strong performance on the mid-sized dataset, achieving a high ROC AUC and balanced precision-recall metrics. Compared to the CAAE in the same setting, the AAE showed slightly lower robustness in distinguishing normal traffic, as reflected by a higher false positive rate. Nevertheless, the AAE maintained a high true positive rate and general accuracy, validating its suitability for real-time anomaly detection in moderately sized CAN datasets. The performance gap with the CAAE, although present, remains relatively small suggesting that for resource-constrained environments, the simpler AAE can still deliver effective results.

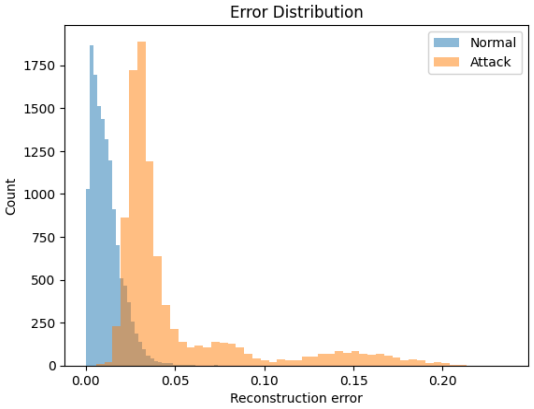
Results: Total samples: 23,199 (MSE): 0.0115 ROC AUC: 0.9661 Detection Threshold: 0.020939 True Positive Rate (TPR): 0.955 False Positive Rate (FPR): 0.127 Accuracy: 91%

Class	**Precision**	**Recall**	**F1-Score**
Normal	97%	87%	92%
Attack	83%	95%	91%









8.6. Small Dataset CAAE Model

This experiment evaluated the performance of the Convolutional Adversarial Autoencoder (CAAE) on a small-scale dataset. The focus was on understanding how well a convolutional architecture could generalize in low-data regimes, especially in distinguishing attack versus normal CAN messages.

Methodology:

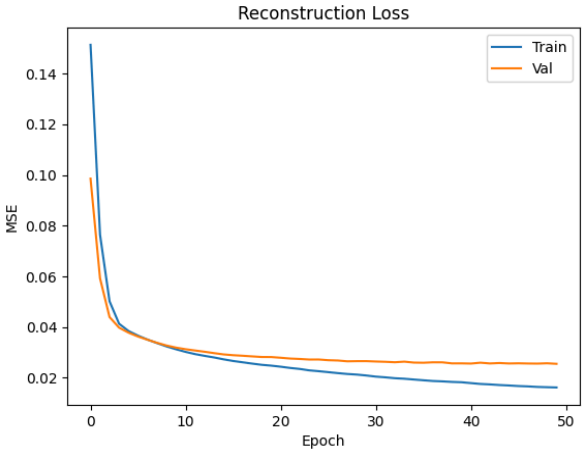
The dataset consisted of only 4,224 records, split across 33 steps per epoch. The CAAE model was built with 42 trainable layers, including convolutional and transposed convolutional blocks. Training was conducted for 50 epochs, and evaluation was performed using ROC-AUC, confusion matrix, and error histograms to determine classification thresholds based on reconstruction error**.**

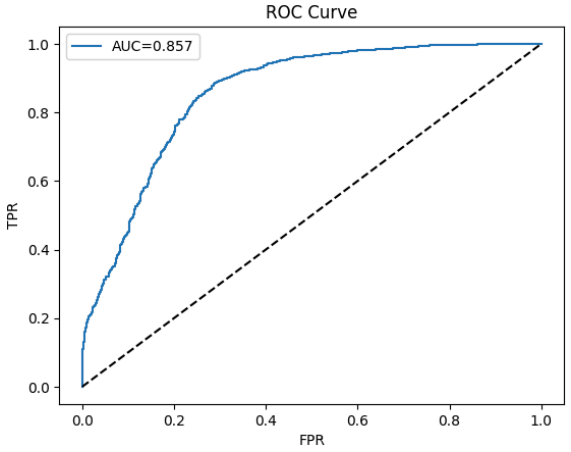
Analysis and Conclusion:

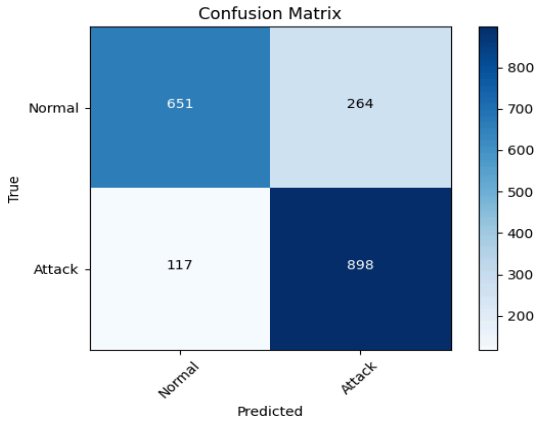
Despite the limited data, the CAAE demonstrated a decent ability to detect attacks (TPR = 0.886). However, the high false positive rate (FPR = 0.289) suggests that the model struggled to learn a clean boundary between normal and anomalous patterns under data-scarce conditions. These results highlight the need for either enhanced regularization, pretraining, or data augmentation when deploying deep models like CAAE in low-resource environments.

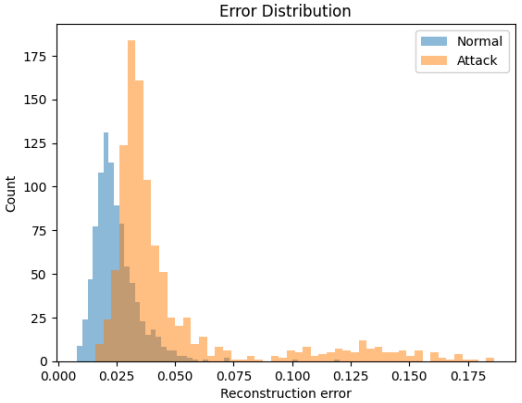
Results: Samples: 1,930 (test) ROC AUC: 0.857 Detection Threshold: 0.0274 TPR: 0.886 FPR: 0.289 Accuracy: 80%

Class	Precision	Recall	F1-Score
Normal	85%	71%	77%
Attack	77%	88%	82%









8.6. Overall Review and Analysis

The complete series of simulations across different dataset sizes and architectures confirms that the adversarial autoencoder framework is effective for CAN intrusion detection. The standard AAE consistently produced strong results on medium and large datasets, demonstrating its ability to learn meaningful latent representations and separate attack behaviors through reconstruction error. However, the CAAE model incorporating convolutional layers provided slightly better performance, particularly on large datasets where its ability to capture spatial patterns offered an edge.

Across all experiments, reconstruction loss decreased steadily, and the ROC curves showed strong separability between normal and anomalous traffic. Confusion matrices revealed that both models captured the majority of attacks with high recall, though the CAAE tended to achieve lower false positives, especially on larger datasets. Classification reports consistently showed that the models were balanced, performing well on both normal and attack classes.

When applied to a small dataset, performance dropped for both models, especially in terms of false positives. Nevertheless, even in these limited-data conditions, the CAAE retained a functional level of detection. This demonstrates its robustness and adaptability to low-resource scenarios. While overfitting was managed well through architectural choices and regularization, the convolutional layers proved beneficial in extracting more structured features from input data.

In summary, all results support the conclusion that both AAE and CAAE models are capable of effectively detecting intrusions in CAN bus systems. CAAE exhibits a consistent but modest advantage, particularly when handling structured or moderately sized data. These findings validate the use of adversarial autoencoders as a promising solution for intelligent vehicle security systems.

9. Future Research and Development Directions

This project introduced a deep learning-based framework for CAN bus intrusion detection using Convolutional Adversarial Autoencoders (CAAE) and Adversarial Autoencoders (AAE). Through various simulations, we demonstrated that both models are capable of identifying attack traffic with high accuracy by learning the structure of normal CAN messages and detecting deviations. While our results indicate strong detection capabilities,

several opportunities remain for improving the model's generalization, robustness, and deployment feasibility. This section outlines potential directions for future development.

9.1. Enhancing Generalization Across Vehicle Types

Our cross-vehicle simulation showed a noticeable performance drop when training and testing on data from different car models. This highlights a need to improve the model’s generalization across heterogeneous CAN environments. Future work should explore domain adaptation techniques and contrastive learning strategies to help the model extract invariant features that are robust to inter-vehicle variations in message formats and timing.

9.2. Improving Real-Time Inference Efficiency

Although the models performed well in offline simulations, real-time deployment on embedded ECUs demands optimization. Future work could focus on compressing the model using pruning, quantization, or knowledge distillation, enabling fast and energy-efficient inference on automotive hardware without sacrificing accuracy.

9.3. Benchmarking Against Adversarial Threats

Finally, as machine learning-based IDS solutions become more prevalent, they will inevitably face adversarial attacks. Future research should evaluate the resilience of CAAE and AAE architectures to evasion or poisoning attacks. Investigating adversarial training or defense-aware architectures will be critical to ensure the system’s reliability under adversarial conditions.

10. Summary

In this project, we developed an advanced intrusion detection system for in-vehicle networks using Adversarial Autoencoders (AAE) and Convolutional Adversarial Autoencoders (CAAE). Our models were trained to learn the underlying structure of benign CAN bus traffic and detect anomalies based on deviations from learned patterns. Through a series of simulations ranging from single-vehicle scenarios to heterogeneous cross-vehicle datasets we demonstrated the system’s effectiveness across multiple conditions and dataset sizes.

Both architectures showed strong performance, with the CAAE slightly outperforming the AAE in several scenarios, especially in terms of recall and generalization to unseen vehicle

data. The models consistently achieved high classification accuracy and robust anomaly detection capability, validating the viability of deep generative approaches for real-time automotive cybersecurity.

Future work will focus on improving domain adaptation, optimizing inference speed for embedded deployment, and integrating sequential temporal models. In conclusion, this project establishes a strong foundation for secure, intelligent in-vehicle communication systems leveraging deep learning. In addition, this work has been mentored By The Israeli Smart Transportation Research Center (ISTRC), and of course the By our supervisor, Prof. Michael Segal.

References

[1]	Flora Amato, Luigi Coppolino, Francesco Mercaldo, Francesco Moscato, Roberto Nardone, and Antonella Santone, "CAN-Bus Attack Detection with Deep Learning," IEEE Transactions on Intelligent Transportation Systems, 2021
[2]	Adrian Taylor, Sylvain Leblanc, and Nathalie Japkowicz, "Anomaly Detection in Automobile Control Network Data with Long Short-Term Memory Networks," IEEE .International Conference on Data Science and Advanced Analytics (DSAA), 2016
[3]	Ilia Odeski, "Anomaly Detection in CAN-BUS Using Pattern Matching Algorithm," Master's Thesis, Ben-Gurion University of the Negev, 2020
[4]	Ken Tindell, "CAN Bus Security: Attacks on CAN Bus and Their Mitigations," Canis Automotive Labs, White Paper, 2020
[5]	Khan, J.; Lim, D.-W.; Kim, Y.-S. "Intrusion Detection System CAN-Bus In-Vehicle .Networks Based on the Statistical Characteristics of Attacks." *Sensors* 2023, 23, 3554
[6]	Flora Amato, Luigi Coppolino, Francesco Mercaldo, Francesco Moscato, Roberto Nardone, and Antonella Santone, "CAN-Bus Attack Detection with Deep Learning," *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[7]	Qian Wang, Zhaojun Lu, and Gang Qu, "An Entropy Analysis Based Intrusion Detection System for Controller Area Network in Vehicles," *2018 31st IEEE International System-on-Chip Conference (SOCC)*, 2018.
[8]	S. Tuohy, M. Glavin, C. Hughes, E. Jones, M. Trivedi, L. Kilmartin, Intra-vehicle networks: a review, IEEE Trans. Intell. Transp. Syst. 16 (2) (2015) 534–545,
[9]	[2] N. Deo, M.M. Trivedi, Multi-modal trajectory prediction of surrounding vehicles with maneuver based LSTMs, in: IEEE Intelligent Vehicles Symposium, Proceedings 2018-June, 2018, pp. 1179–1184,

Appendix

Project Repository

The full source code, simulation scripts, preprocessed datasets, and all supporting resources related to this project are publicly available on GitHub. This repository enables complete reproducibility of the experiments, including model training, evaluation procedures, and result visualization for both AAE and CAAE architectures applied to CAN bus intrusion detection. Researchers and practitioners are welcome to explore, validate, or extend the work presented in this report.

GitHub Repository: **https://github.com/omri440/CAE_project/tree/main**

A: ROC AUC – Receiver Operating Characteristic Area Under Curve

The ROC curve plots the True Positive Rate (TPR) against the False Positive Rate (FPR) at various classification thresholds. The AUC (Area Under Curve) quantifies the overall ability of a model to distinguish between classes. ROC AUC is threshold-independent and offers a global measure of model performance. AUC = 1 indicates perfect classification; AUC = 0.5 suggests no discriminative power (random guess).

B: Youden's J Statistic Threshold Selection Metric

Youden's J is a scalar metric to determine the optimal threshold for binary classification. Defined as:

$$J = TPR - FPR$$

Connection to ROC:

On the ROC curve, the threshold with the maximum vertical distance from the diagonal line (random classifier) corresponds to the highest J. It balances sensitivity (true positive rate) and specificity (true negative rate).

Why use it:

Helps choose a cut-off point that maximizes correct classifications

Avoids bias toward either class in imbalanced data

Gives a principled, interpretable threshold

C: Dense Layers and Dropout

Dense Layer (Fully Connected):

Each neuron in a dense layer receives input from all neurons in the previous layer. It performs:

$$output = \Omega(Wx + b)$$

where W is the weight matrix, x the input, b the bias, and Ω the activation function (e.g., ReLU).

Dropout Rate:

Dropout is a regularization technique where, during training, a fraction of neurons is randomly "dropped" (ignored) to prevent the network from over-relying on specific paths.

D: GAN-Architectue

Generative Adversarial Networks (GANs) are deep learning frameworks designed to approximate a target data distribution using a generative model . GANs consist of two competing neural networks: a generator G and a discriminator D.

The generator takes as input a noise vector , drawn from a known prior (e.g., Gaussian), and produces a sample . The discriminator attempts to distinguish real data samples from those generated by G.

The training is defined by the minimax objective:

In the context of AAE, the encoder acts as G and the discriminator is trained on the latent space instead of image or time-series space. This mechanism allows AAE to shape its latent distribution and thus detect anomalies through deviations in encoding or reconstruction.

The discriminator is trained to distinguish real latent vectors (sampled from the prior) from fake ones (output of the encoder), and the encoder is trained to fool the discriminator. This adversarial training helps impose structured priors on the latent space, which is key for robust anomaly detection.

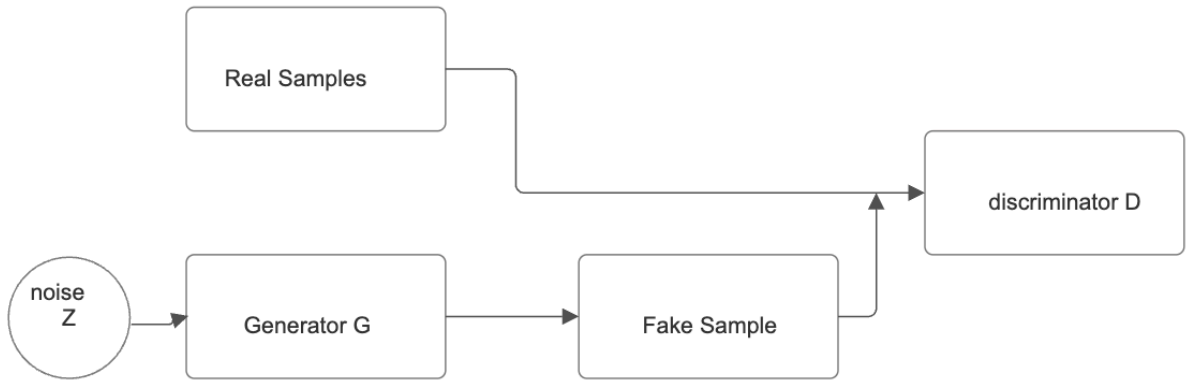


Fig 4. GAN Architecture Scheme

Mathematical Foundations of GAN

A Generative Adversarial Network (GAN) is defined by a game between two players: a generator G(z; θ_G)and a discriminator D(x; θ_G). The generator maps a latent variable $z \sim p_z(z)$(usually Gaussian or uniform) to the data space x, aiming to produce realistic

samples. The discriminator tries to distinguish between real samples $x \sim p_{data}$ and generated samples $\hat{x} = G(z) \sim p_G$

The training objective is a two-player minimax game:

$$min_G\, max_D\, V(D,\, G) \;=\; \mathbb{E}_{x \sim p_{data}}\, [log\, D(x)] \;+\; \mathbb{E}_{z \sim p_z}[log(1 \;-\; D(G(z)))]$$

Where:

- G: Generator network
- D: Discriminator network
- x: Real data sample from distribution p_data
- z: Random noise sample from distribution p_z
- $\mathbb{E}$: Expectation operator

$D(x) \in [0, 1]$ is the probability assigned to input x being real.
The discriminator is optimized to maximize classification accuracy.
The generator is optimized to minimize the probability of being classified as fake.

Why Classical GANs Collapse: Divergence Pathology

The classical GAN training implicitly minimizes the Jensen–Shannon (JS) divergence:

$$D_{JS}(p_{data} || p_G)$$

But JS divergence suffers when data distribution and Generator distribution lie on low-dimensional manifolds that do not overlap in such cases, the gradients vanish and training becomes unstable.

WGAN: Wasserstein Distance and the Kantorovich Duality

To overcome these issues, Wasserstein GAN (WGAN) replaces JS divergence with the Wasserstein-1 distance (Earth Mover's Distance, EMD):

$$W(p,\, q) \;=\; inf_{\gamma \in \pi(p,\, q)}\, E_{(x,y)}\gamma[\, ||x \;-\; y||\,]$$

Using Kantorovich–Rubinstein duality:

$$W(p,\ q)\ =\ sup\|f\|_L \leq 1\ \mathbf{E}_{x\sim p}\,[f(x)]\ -\ \mathbf{E}_{x\sim q}\,[f(x)]$$

This leads to the WGAN objective:

$$max_{D\,\in\,L_1}\mathbf{E}_{x\sim p_{data}}\,[D(x)]\ -\ \mathbf{E}_{z\sim p_z}\,[D(G(z))]$$

Where D is a 1-Lipschitz function acting as a critic, not a classifier.

Enforcing 1-Lipschitz Constraint: Gradient Penalty

To ensure D remains 1-Lipschitz, WGAN-GP introduces a gradient penalty.

Let x ~ p_{data}, x̂ ~ p_G, and ε ~ U[0,1], and define:

$$\tilde{x}\ =\ \varepsilon x\ +\ (1\ -\ \varepsilon)\hat{x}$$

Then:

$$GP\ =\ \lambda\cdot E_{\tilde{x}}\,[(||\nabla_{\tilde{x}}D(\tilde{x})||_2\ -\ 1)^2]$$

The full discriminator loss becomes:

$$L_D\ = E_{x\sim\,p_{data}}\,[D(x)]\ -\ E_{z\,\sim\,p_z}[D(G(z))]\ +\ GP$$

The generator loss remains:.

$$L_G\ =\ -\,E_{z\,\sim\,p_z}[D(G(z))]\ +\ GP$$

AAE and Latent Regularization via GAN

In Adversarial Autoencoders (AAE), GAN is applied in latent space. The encoder E(x) outputs z ~ q(z|x), and we want q(z) ≈ p(z) (e.g., N(0, I)).

Introduce a discriminator D(z) to distinguish p(z) from q(z). The GAN objective becomes:

$$max_D E_{z \sim p_z}[D(z)] - E_{x \sim p_{data}}[D(E(x))] + GP$$

$$min_E - E_{x \sim p_{data}}[D(E(x))]$$

This encourages q(z) to match p(z) under the Wasserstein metric.

Information Theoretic Interpretation

This adversarial setup implicitly minimizes divergence between q(z) and p(z), without explicit KL divergence computation:

$$D_{KL}(q(z|x) \,||\, p(z))$$

Instead, it relies on sample-based gradients, which is practical for complex distributions.

E. System Components and Cost Functions

In this project, the design of the AAE and CAAE models was guided by a careful selection of mathematical components to ensure both stable training and expressive model capacity. This section presents the key computational building blocks of the system.

E.1. Reconstruction Loss (MSE) 1D & 3D:

At the core of the autoencoder lies the Mean Squared Error (MSE) loss function, which measures the average squared difference between the input sample and its

reconstruction. This function enforces the decoder to reconstruct samples as closely as possible to the original and serves as the main learning signal for the autoencoder.

Reconstruction Loss:

$$L_{recon}(x, \hat{x}) = ||x - \hat{x}||^2$$

Where:
- x: Original input
- x̂: Reconstructed input
- ||·||²: Squared L2 norm

The reconstruction error for a 3D image tensor is calculated using the Mean Squared Error (MSE) as follows:

$$MSE = \frac{1}{(H \cdot W \cdot C)} \cdot \sum_{(i,j,k)} (x_{i,j,k} - \hat{x}_{i,j,k})^2$$

Explanation

Here, H, W, and C are the dimensions of the input image:

- H = Height
- W = Width
- C = Channels

In our specific case, the input to the model is a tensor of shape 32×32×2:

- H = 32 (image height)
- W = 32 (image width)
- C = 2 (channels: one for CAN ID bits, one for CAN data bits)

The MSE computes the average of the squared differences between the original input x and its reconstruction x̂ across all pixels and all channels.

E.2. Categorical Cross-Entropy for Label Consistency

To guide the encoder in learning a structured label-aware latent representation y, a cross-entropy loss is added:

Label Loss:

$$L_y = CE(y_{true}, y_{logits})$$

Where:
- CE: Categorical cross-entropy function
- y_{true}: Ground truth labels
- y_{logits}: Predicted logits from the encoder

E.3. Activation Functions: ELU and LeakyReLU

For nonlinear transformations, the Exponential Linear Unit (ELU) was used across most layers, offering smoother gradients than ReLU and mitigating dead neuron issues. In some alternative architectures (e.g., AAE), LeakyReLU was tested to preserve non-zero gradients for negative inputs, especially beneficial in deep or sparse settings.

E.4. Optimization Strategy: Adam Optimizer

All components of the system — autoencoder, discriminators, and encoder — were trained using the Adam optimizer, chosen for its adaptive learning rate and stability in high-dimensional spaces. Each component had a separately tuned learning rate:

Learning Rates:

$$LR_{AE} = 5\times10^{-4}$$

$$LR_{D_z}, LR_{D_y} = 1\times10^{-4}$$

$$LR_G = 5\times10^{-5}$$

E.5. Gradient-Based Training and Backpropagation

Training was performed using TensorFlow's `tf.GradientTape()` API, enabling fine-grained control over the updates of each model component. Each loss term’s gradient was applied only to its designated variable group, ensuring decoupled and stable optimization dynamics.

E.6. Error-Based Anomaly Detection

Once trained, the system detects anomalies by measuring reconstruction error:

Anomaly Score:

$$e(x) = MSE_{3D}((x - \hat{x})^2) \; over\; all\; pixels$$

Where:
- e(x): Anomaly score for input x
- x̂: Reconstructed version of x
- High e(x) indicates a potential anomaly