USENIX

THE ADVANCED COMPUTING
SYSTEMS ASSOCIATION

Lightweight Deep Learning for Cyber-Resilient Heavy Vehicles: Efficient Signal Reconstruction on Embedded Systems

Maxwel Bar-on, *Colorado State University;* Hossein Shirazi, *San Diego State University;* Indrakshi Ray and Jeremy Daily, *Colorado State University*

https://www.usenix.org/conference/vehiclesec25/presentation/bar-on

This paper is included in the Proceedings of the
3rd USENIX Symposium on Vehicle Security and Privacy.

August 11–12, 2025 • Seattle, WA, USA

978-1-939133-49-6

Open access to the Proceedings of the 3rd USENIX Symposium
on Vehicle Security and Privacy is sponsored by USENIX.

Lightweight Deep Learning for Cyber-Resilient Heavy Vehicles: Efficient Signal Reconstruction on Embedded Systems

Maxwel Bar-on
Department of Computer Science
Colorado State University
Maxwel.Bar-on@colostate.edu

Hossein Shirazi
Department of Management Information Systems
San Diego State University
hshirazi@sdsu.edu

Indrakshi Ray
Department of Computer Science
Colorado State University
indrakshi.ray@colostate.edu

Jeremy Daily
Department of Systems Engineering
Colorado State University
jeremy.daily@colostate.edu

Abstract

Modern heavy vehicles rely on insecure protocols (CAN and SAE-J1939) to facilitate communication between the embedded devices that control their various subsystems. Due to the growing integration of wireless-enabled embedded devices, vehicles are becoming increasingly vulnerable to remote cyberattacks against their embedded networks. We propose an efficient deep-learning-based approach for mitigating such attacks through real-time J1939 signal reconstruction. Our approach uses random feature masking during training to build a generalized model of a vehicle's network. To reduce the computational and storage burden of the model, we employ 8-bit Quantization-Aware Training (QAT), enabling its deployment on resource-constrained embedded devices while maintaining high performance. We evaluate Transformer and LSTM-based architectures, demonstrating that both effectively reconstruct signals with minimal computational and storage overhead. Our approach achieves signal reconstruction with error levels below 1% of their operating range while maintaining a very low storage footprint of under 1 MB, demonstrating that lightweight deep-learning models can enhance resiliency against real-time attacks in heavy vehicles.

Keywords: Controller Area Network, SAE J1939, Quantization, Signal Reconstruction, Transformer, LSTM, Embedded Systems, Vehicle Networks.

1 Introduction

Heavy-duty vehicles are essential to the global economy, playing a crucial role in the transportation of goods. In the United States alone, trucks move over **$13 trillion** worth of material annually, accounting for **72%** of domestic shipments [3].

Modern Heavy vehicles rely on networks of embedded devices for controlling and monitoring operations [33]. Communication between these devices is facilitated by insecure protocols, **SAE-J1939** and **Controller Area Network (CAN)**. These protocols were designed with the assumption that attacks against a vehicle's network would require physical access to the CAN bus. Due to the integration of wireless-enabled embedded devices (e.g., Telematics Unit, GPS) into vehicular networks [1, 22], this assumption no longer holds. A remotely compromised embedded device can exploit vulnerabilities in these protocols to subvert critical vehicle functions by disrupting communication between legitimate devices [26, 33]. Given their integral role in global supply chains, cyberattacks against heavy vehicle embedded networks can result in severe logistical disruptions, economic losses, and safety hazards. Therefore, it is essential to detect and mitigate such attacks.

Prior research has primarily focused on **Intrusion Detection Systems (IDS)** for detecting anomalous behavior in CAN networks [4, 9, 23, 34]. While IDS can identify cyber threats, they do not provide mechanisms for mitigation, which is critical for maintaining vehicle operability during an attack. Other works have proposed **protocol modifications** [15, 35, 37] for preventing CAN-based attacks. However, these approaches require widespread implementation, which is infeasible due to a lack of standardization across manufacturers of embedded devices.

Deep-learning approaches have been proposed for mitigating attacks through real-time reconstruction of compromised J1939 signals [40]. While effective, these models cannot be deployed on resource-constrained embedded systems due to **high computational and memory demands**. Furthermore, existing methods require **separate models for each type of signal**, leading to excessive storage overhead and limiting scalability. Offloading

computation to external servers is also impractical as heavy vehicles may operate in areas with limited or intermittent connectivity. For a solution to be effective, it must be capable of accurately reconstructing signals while operating efficiently on embedded devices.

1.1 Proposed Approach

To address these limitations, we propose a **deep-learning-based approach for J1939 signal reconstruction** with reduced computational and storage overhead for deployment on resource-constrained embedded devices. In the event of an attack, our approach is capable of reliably reconstructing disrupted signals. This enables real-time recovery from attacks by replacing disrupted signals with their reconstructed values. We utilize **random feature masking** during training to build a **unified model** of a vehicle's network, which eliminates the need for separate models and improves its generalizability for unseen attack patterns. Additionally, we employ **Quantization-Aware Training (QAT)** to reduce the precision of model parameters from **32-bit to 8-bit**, significantly lowering computational and storage costs while maintaining high accuracy.

For our reconstruction model, we explore **Transformer and LSTM-based architectures**. These architectures have both demonstrated strong performance in various multivariate time-series learning tasks [28,40,42,44]. Through extensive evaluation, we identify deep-learning architectures that achieve an optimal balance between performance and efficiency.

1.2 Contributions

This paper presents an efficient deep-learning approach for reconstructing compromised J1939 signals. Unlike traditional Intrusion Detection Systems (IDS), our approach enables mitigation of J1939-based attacks by actively reconstructing disrupted signals in real-time. The key contributions of this paper are as follows:

- A **lightweight deep-learning framework for J1939 signal reconstruction** that enhances the resilience of heavy-duty vehicles to cyberattacks, enabling real-time operation in resource-constrained environments.
- A **unified, scalable approach** that improves generalizability and eliminates the need for separate models for each signal type, significantly reducing storage costs.
- A **comparative analysis of Transformer and LSTM architectures** based on efficiency, performance for J1939 signal reconstruction, and robustness to quantization.
- An **evaluation across multiple simulated attacks** based on realistic threat scenarios in heavy vehicle networks.
- A **novel computational cost estimation framework** for evaluating the efficiency of quantized models in embedded vehicular networks.

The rest of the paper is organized as follows. In Section 2, we summarize related literature on the security of heavy-vehicle networks, as well as time-series learning and quantization. In Section 3, we provide a background on vehicular embedded networks and the specific threat model that we address. In Section 4, we explain our approach and provide the details of our deep-learning architecture and quantization. In Section 5, we evaluate our approach in terms of its efficiency, performance, and robustness to quantization. In Section 6, we evaluate our approach under simulated attacks based on realistic threat scenarios. Finally, Section 7 concludes our work and suggests future directions for research.

2 Related Work

In this section, we review relevant research on the security of heavy vehicles, deep-learning-based time-series modeling, and quantization of neural networks. We highlight existing solutions, their limitations, and how our approach addresses key challenges in vehicular network security.

2.1 Security of Vehicular Networks

With the growing reliance on wireless-enabled embedded systems, vehicular networks are increasingly vulnerable to cyberattacks. These attacks range from sensor disruptions to full network intrusions, exploiting the inherent weaknesses of protocols like CAN and J1939. El-Rewini et al. [12] investigated attacks targeting embedded sensors in vehicles. While these attacks focused on individual sensors, other works [13, 19, 33] demonstrated the potential for large-scale exploits by compromising communication protocols [26] used in heavy vehicle networks.

To mitigate such attacks, multiple security mechanisms have been proposed. Protocol-level defenses such as message authentication [15,35,37] aim to prevent unauthorized access. However, implementing these changes requires standardization across manufacturers, making them impractical for large-scale deployment. An alternative approach is Intrusion Detection Systems (IDS),

which monitor CAN traffic for anomalies [4, 9, 23, 34]. IDS solutions have demonstrated effectiveness in detecting attacks, but they do not offer mechanisms for recovery once a breach has occurred.

Cho et al. [4] introduced a clock-skew-based method to detect anomalies in messages originating from components with regular communication intervals. Later, Mukherjee et al. [34] proposed a precedence-graph-based approach for distinguishing between legitimate and malicious CAN messages. While these methods enhance attack detection, they do not address the issue of signal recovery, leaving compromised systems unable to function properly after an attack.

Shirazi et al. [40] proposed an LSTM-based approach for reconstructing compromised J1939 signals to restore vehicular network functionality. Their approach involved training individual models for different signal types, selecting input features based on correlation coefficients between signals. However, maintaining separate models for each signal significantly increases storage overhead, making real-time deployment on embedded devices impractical. Additionally, this approach would not be effective for unseen attack patterns or attack scenarios that impact multiple correlated signals. Our approach improves upon this by introducing a unified model capable of reconstructing multiple signal types simultaneously. Additionally, we employ Quantization-Aware Training (QAT) to reduce model size and computation costs while maintaining high reconstruction accuracy.

2.2 Deep Learning for Time-Series Modeling

Deep-learning models have surpassed traditional statistical methods for complex time-series problems due to their ability to learn nonlinear relationships in multivariate data [27]. The dominant architectures for time-series modeling rely on convolutional [6, 29, 30], recurrent [17, 28, 31, 32], or attention-based [14, 24, 41, 42, 44] layers to capture temporal dependencies.

Recurrent Neural Networks (RNNs) learn temporal patterns by sequentially updating a hidden state, which serves as the network's short-term memory. However, deep RNNs suffer from the vanishing gradient problem [16, 38], which limits their ability to train on long-sequence datasets. To address this, Hochreiter et al. [17] introduced Long Short-Term Memory (LSTM), a RNN variant incorporating memory cells with gating mechanisms to regulate information flow. LSTMs have demonstrated superior performance in various time-series applications, including traffic speed prediction [7], consumer demand forecasting [6], and pedestrian trajectory modeling [43].

Shirazi et al. [40] applied multivariate LSTMs to vehicular network signal reconstruction. Instead of training multiple models as in [40], our approach trains a single LSTM with a multivariate output to efficiently reconstruct different signal types. Like [40], we use a vanilla LSTM [17], which is well-suited for sequence-to-vector tasks in vehicular networks.

While LSTMs effectively capture temporal dependencies, they struggle with long-range dependencies due to their sequential nature. Transformer models address this limitation by leveraging self-attention mechanisms instead of recurrent connections [41]. Transformers have achieved state-of-the-art performance in natural language processing [11, 39] and have been successfully adapted for time-series forecasting, anomaly detection, and classification [14, 42, 44].

Notably, Zerveas et al. [44] demonstrated strong results in multivariate time-series regression using a Transformer pre-trained with random feature masking. We adopt a similar approach, training both our Transformer and LSTM models with random masking to create a unified model for signal reconstruction. Unlike [44], our approach does not require fine-tuning because the masking task itself enables the model to learn vehicular network behavior.

2.3 Quantized Neural Networks

Quantization is an essential technique for reducing the computational and storage costs of deep-learning models, especially for deployment in resource-constrained embedded systems. It works by representing model parameters with lower-precision data types, reducing memory usage and enabling more efficient operations [18]. However, quantization can degrade model performance due to clipping and rounding errors [36]. These errors arise when floating-point values are truncated or mapped to a limited numerical range (e.g., $[-127, 127]$ for signed 8-bit integers).

Existing quantization methods fall into two categories: Post-Training-Quantization (PTQ) and Quantization-Aware Training (QAT). PTQ applies quantization after training, using a small calibration dataset to estimate scaling factors [2, 5]. While PTQ is computationally efficient, it often leads to performance degradation due to inaccurate quantization parameter estimation.

In contrast, QAT incorporates quantization effects during training, allowing the model to learn compensatory adjustments [21] instead of relying solely on precise

quantization parameter estimation to reduce clipping and rounding errors. This approach enables higher accuracy than PTQ but requires careful implementation. In our QAT approach, we use full-precision parameters and gradients during training while simulating quantization of weights and activations during forward passes. While this results in a higher training cost compared to mixed-precision QAT [10], it improves the generalizability of our model for unseen samples of J1939 traffic. To further minimize quantization error, we estimate scaling factors using a statistical approach based on weight and activation distributions [36]. This method balances simplicity and accuracy, making it well-suited for embedded vehicular applications.

3 System Model

In this section, we provide an overview of communication between embedded devices on heavy vehicles and how it can be represented as a multi-variate time-series, define our threat model, and explain how our approach can be used to mitigate attacks.

3.1 Background

Heavy vehicles rely on a distributed network of Electronic Control Units (ECUs) to manage their various subsystems. Unlike centralized computing architectures, this distributed approach enables real-time processing and fault isolation. These ECUs function as independent embedded devices but communicate with one another through data packets formatted according to the SAE-J1939 standard.

J1939 packets are transmitted over the vehicle's Controller Area Network (CAN) bus following the CAN physical-layer protocol. Each packet consists of:

- A 29-bit header, which contains metadata including the Parameter Group Number (PGN) that defines how to interpret the packet.
- An 8-byte data payload, which encodes one or more vehicle signals.

Each signal type is uniquely identified by a Suspect Parameter Number (SPN). The J1939 standard provides a structured mapping between PGNs and SPNs; the PGN in a packet header specifies which SPNs are contained in the payload. In a J1939 packet, SPNs are assigned fixed positions in the payload. For each SPN, the J1939 standard defines the starting bit index and bit length. This can be used to decode packets by locating the positions of each SPN.

In our experiments, we collect SPN signals by connecting a J1939 packet decoder module to the CAN bus through a diagnostics port as shown in Figure 1 This module receives messages transmitted over the bus and extracts SPN signals according to the J1939 standards.

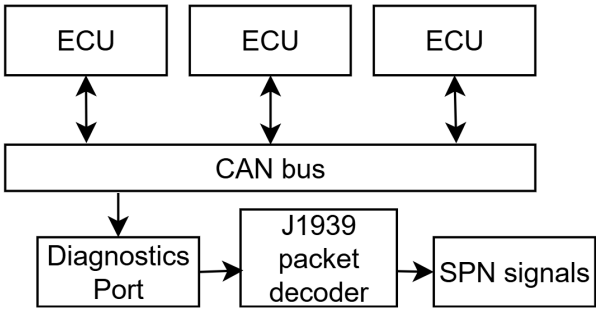

Figure 1: **Architecture of heavy vehicle network.** Electronic Control Units (ECU) transmit and receive J1939 packets from the Controller Area Network (CAN) bus. A J1939 packet decoder observes packets by connecting to the CAN through the diagnostics port and extracts SPN signals according to the J1939 protocol standards.

3.2 Vehicular Network Time-Series

J1939 packets enable the delivery of vehicular signals between embedded devices, SPNs define the rules for extracting individual signals. Abstractly, each SPN represents a distinct signal in a vehicle's embedded network. For the purposes of reconstruction, we interpret the vehicular network as a multi-variate time-series where each SPN is represented by a different variable. This allows us to apply deep time-series learning models to CAN traffic.

To generate a time series, we record the most recently observed value from each SPN at regular 60ms intervals. For example, let $X \in \mathbb{R}^{N \times m}$ be a time-series of N timesteps for a vehicle with m different SPNs. $X_{i,j}$ will contain the most recently emitted value for SPN j at timestep i. Timestep i represents the state of the vehicle's network after $i * 0.06$ seconds of driving.

3.3 Threat Model

The goal of the adversary in this study is to disrupt communication between embedded devices on the CAN bus, thereby disabling critical vehicle functions. We assume the adversary has access to the CAN bus through a compromised ECU, enabling them to transmit unauthorized messages.

Due to bandwidth limitations and constraints on embedded devices, the adversary cannot fully saturate the CAN bus but is capable of selectively interfering with legitimate messages. This assumption is consistent with real-world CAN bus attacks, where adversaries typically exploit vulnerabilities in message arbitration rather than using brute-force techniques [33]. Targeted disruption of important messages can cause significant system failures even without full CAN-bus saturation [26].

The adversary may employ various attack techniques, including:

- **Message Injection:** Sending false messages to override legitimate control signals, causing unintended vehicle behavior.
- **Filter Overloading:** Flooding the network with high-priority messages to congest the message filtering system, leading to delayed or dropped critical commands.
- **ECU Connection Exhaustion:** Exploiting ECU resource limitations by continuously opening and maintaining network connections, preventing normal operations.

These attacks can compromise key vehicle functionalities including braking, throttle control, and engine diagnostics. Traditional countermeasures, such as message authentication and anomaly-based intrusion detection systems (IDS), cannot be used on the current SAE-J1939 to provide resiliency. Our approach aims to address this gap by providing a real-time signal reconstruction mechanism to mitigate the effects of compromised signals.

In this work, we simulate CAN-based cyberattacks by artificially disrupting signals from a portion of SPNs at every timestep.

3.4 SPN Signal Reconstruction

Our approach enables SPN reconstruction by predicting a value for each disrupted SPN using a deep-learning model. When an SPN is disrupted, we remove its emitted signal from the input to our reconstruction model. We train a deep-learning model to fill in the "gaps" in the time-series that result from our simulated attacks. When we simulate an attack, we provide a sample time-series to the model and it produces a prediction for each variable in the final timestep. In a real vehicular network, SPN reconstruction can be achieved by observing J1939-traffic for several timesteps, extracting a multi-variate time-series representation of the different SPNs, then applying the model to the time-series. The model will predict the disrupted signals at the most recently observed timestep using the previous timesteps as context. Following the initial prediction, this process can be applied continuously using a sliding-window of SPN values to reconstruct signals at every timestep.

4 Methods

In this section, we describe our approach for reconstructing disrupted vehicular signals across multiple SPNs using a unified deep-learning model, and for reducing the computational cost of this model through Quantization-Aware Training (QAT).

4.1 Signal Reconstruction

Our method trains a deep-learning model to reconstruct missing SPN signals at the current timestep by leveraging the recent history of values for all SPNs in the network. We assume that one or more SPN signals may be disrupted at each timestep; these disrupted values are represented by -1. Let $x \in \mathbb{R}^{w \times m}$ denote a sample time-series of J1939 traffic, where w is the window size (number of timesteps) and m is the number of SPNs. The goal is to predict the missing values in the final timestep x_w. Given x, the model outputs a vector $y \in \mathbb{R}^m$ containing the reconstructed values for all m SPNs. For example, if $x_{w,j}$ is disrupted, y_j will contain the reconstructed value for SPN j.

We explore two deep-learning architectures for signal reconstruction: a Long Short-Term Memory (LSTM)-based model and a Transformer-based model.

4.1.1 SPN Disruption

To simulate attacks, we mask a subset of SPN values at every timestep in the model input. We use two techniques for selecting which SPNs to disrupt: **random disruption**, and **attack-path-based disruption**. Random disruption is designed to efficiently cover a wide range of distinct attack patterns. Attack-path-based disruption is designed to simulate realistic attack scenarios. For an input $x \in \mathbb{R}^{w \times m}$, these techniques generate a mask $M \in \{0,1\}^{w \times m}$ and apply it to the input as follows:

$$x_{i,j} = \begin{cases} -1 & \text{if } M_{i,j} = 1 \\ x_{i,j} & \text{otherwise} \end{cases}$$

For **random disruption**, we generate masks from a binomial distribution where each element in the $(w \times m)$-dimensional grid has a constant p probability of being

assigned the value 1 where p is the mask-rate. The mask is generated as:

$$M \sim \mathcal{B}(1,p)^{w \times m} \tag{1}$$

where $\mathcal{B}(1,p)$ denotes a binomial distribution with 1 trial and a p probability of success.

For **attack-path-based disruption**, we use a predefined mask that disrupts SPNs based on the expected impact of the corresponding attack. For each attack path in Table 6.1, we define a mask that blocks the affected SPNs in the model input. For example, let J be the set of SPNs disrupted by a particular attack path. To simulate this attack, the mask will be defined as:

$$M_{i,j} = \begin{cases} 1 & \text{if } j \in J \ \wedge \ i \geq w - \delta \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

where δ is the duration of the attack.

4.1.2 LSTM-Based Reconstruction

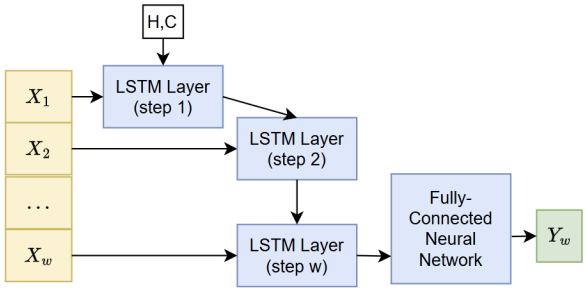

Figure 2: Diagram of the LSTM model for SPN reconstruction.

LSTM networks [17] address the vanishing gradient problem inherent in traditional RNNs by employing memory cells that capture long-range dependencies. Our LSTM architecture consists of a single unidirectional LSTM layer with $d = 160$ hidden units, followed by a fully-connected neural network. The LSTM layer maintains a cell state $c \in \mathbb{R}^d$ and hidden state $h \in \mathbb{R}^d$, and employs three gating mechanisms: the forget gate $f(x,h;\theta_f,U_f,\beta_f)$, the input gate $I(x,h;\theta_I,U_I,\beta_I)$, and the output gate $o(x,h;\theta_o,U_o,\beta_o)$, along with a cell-update function $\tilde{c}(x,h;\theta_{\tilde{c}},U_{\tilde{c}},\beta_{\tilde{c}})$. At each timestep t (with $0 \leq t < w$), given the input $x_{(t)} \in \mathbb{R}^m$ and previous hidden state $h_{(t-1)}$, the gates are computed as:

$$f_{(t)} = \sigma(\theta_f x_{(t)} + U_f h_{(t-1)} + \beta_f), \tag{3}$$

$$I_{(t)} = \sigma(\theta_I x_{(t)} + U_I h_{(t-1)} + \beta_I), \tag{4}$$

$$o_{(t)} = \sigma(\theta_o x_{(t)} + U_o h_{(t-1)} + \beta_o), \tag{5}$$

$$\tilde{c}_{(t)} = \tanh(\theta_{\tilde{c}} x_{(t)} + U_{\tilde{c}} h_{(t-1)} + \beta_{\tilde{c}}). \tag{6}$$

The cell state is updated as:

$$c_{(t)} = f_{(t)} \odot c_{(t-1)} + I_{(t)} \odot \tilde{c}_{(t)}, \tag{7}$$

and the hidden state is given by:

$$h_{(t)} = \text{LayerNorm}\left(o_{(t)} \odot \tanh(c_{(t)})\right). \tag{8}$$

After processing the input sequence $x \in \mathbb{R}^{w \times m}$ (with $h_{(0)} = 0$ and $c_{(0)} = 0$), the final hidden state $h_{(w)}$ is fed into a fully-connected network. With hidden layer weights $\theta_z \in \mathbb{R}^{d \times 40}$ and bias $\beta_z \in \mathbb{R}^{40}$, and output layer weights $\theta_\omega \in \mathbb{R}^{40 \times m}$ and bias $\beta_\omega \in \mathbb{R}^m$, the model output is:

$$y = \tanh\left(h_{(w)}\theta_z + \beta_z\right)\theta_\omega + \beta_\omega. \tag{9}$$

4.1.3 Transformer-Based Reconstruction

The Transformer architecture [41] employs attention mechanisms rather than recurrent connections to capture temporal dependencies. Our Transformer model consists of an embedding layer, a multi-block Transformer encoder, and a linear decoder as shown in Fig. 3. The embedding layer projects the input $X \in \mathbb{R}^{w \times m}$ onto a d-dimensional space using learnable weights $\theta_e \in \mathbb{R}^{m \times d}$ and bias $\beta_e \in \mathbb{R}^d$, with added positional encodings. The encoder comprises three identical blocks, each with a self-attention sub-layer and a fully-connected sub-layer, followed by residual connections and layer normalization.

The self-attention sub-layer projects inputs onto Query $Q \in \mathbb{R}^{d \times u}$, Key $K \in \mathbb{R}^{d \times u}$, and Value $V \in \mathbb{R}^{d \times u}$ matrices. For our attention mechanism, we use multi-head self attention where the attention operation is repeated across 24 'heads' defined as $\frac{u}{24}$-dimensional subspace of Q,K and V. The attention operation is defined as:

$$attn(q,k,v) = softmax(\frac{qk^T}{\sqrt{\frac{u}{24}}})v \tag{10}$$

where $q,k,v \in \mathbb{R}^{w \times \frac{u}{24}}$ are subspaces of Q,K and V respectively. The results of each head are concatenated then combined via a linear projection with weights $\theta \in \mathbb{R}^{u \times d}$ and bias $\beta \in \mathbb{R}^d$. We set the input projection dimension $d = 40$ and the attention sub-layer projection size $u = 96$. Only the final timestep of the encoder output is used, and a linear decoder projects this onto m-dimensional space using weights $\theta_y \in \mathbb{R}^{d \times m}$ and bias $\beta_y \in \mathbb{R}^{d \times m}$.

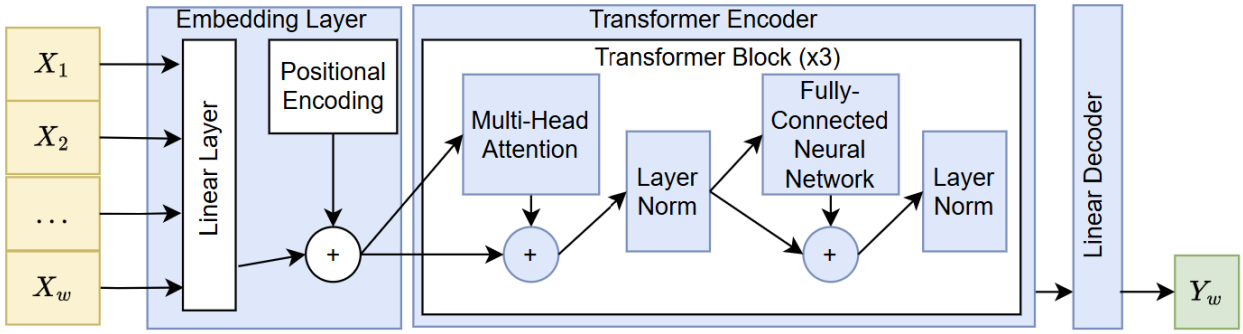

Figure 3: Diagram of the Transformer architecture for SPN reconstruction.

4.1.4 Model Training

The model is trained to minimize the Mean Squared Error (MSE) loss computed only on the masked entries. Let $Y \in \mathbb{R}^{n\times m}$ be the model output for a batch of n input samples $X \in \mathbb{R}^{n\times w\times m}$, with corresponding mask $M \in \{0,1\}^{n\times w\times m}$ generated according to one of the disruption techniques described in Section 4.1.1. the loss is defined as:

$$\mathcal{L} = \sum_{i=1}^{n}\sum_{k=1}^{m} \mathbf{M}_{i,w,k} * (X_{i,w,k} - Y_{i,k})^2 \tag{11}$$

We optimize the parameters using the Adam optimizer [25] with $\beta_1 = 0.9$, $\beta_2 = 0.999$, and an initial learning rate of 0.01. Training is performed in mini-batches (batch size of 1,000 for the Transformer and 100 for the LSTM) until the learning rate decays below 10^{-5} or after 1,000 epochs. We decay the learning rate by a factor of $3\times$ when we observe no improvement in $\mathcal{L}$ evaluated on the validation data for 10 consecutive epochs. After each epoch, the masks are re-calculated to change which SPNs are masked in each training sample.

4.2 Quantization-Aware Training

We employ Quantization-Aware Training (QAT) to reduce the memory and computational requirements of our models, enabling deployment on resource-constrained embedded devices. QAT simulates the effects of quantization during the forward pass while using full-precision values for backpropagation. In our implementation, 32-bit floating-point values are converted to 8-bit signed integers, achieving up to $4\times$ reduction in storage and significant computational savings [36]. Quantization is simulated during training by converting 32-bit weights and activations to the nearest whole number in the range $[-127, 127]$.

The quantized value of a full-precision number x is computed as:

$$\text{clamp}\left(\left\lfloor \frac{x}{s} \right\rceil\right), \tag{12}$$

where s is a scale parameter, $\lfloor\cdot\rceil$ denotes rounding to the nearest integer, and the clamp function is defined by:

$$\text{clamp}(x) = \begin{cases} -127, & x < -127, \\ x, & -127 \geq x \geq 127, \\ 127, & x > 127. \end{cases} \tag{13}$$

For each linear layer, we maintain a scaling parameter for each set of full-precision elements: s_θ for weights and s_x for activations. Given full-precision weights $\theta \in \mathbb{R}^{m\times d}$, bias $\beta \in \mathbb{R}^d$ and input $x \in \mathbb{R}^{w\times m}$, the quantized operation is simulated as:

$$(s_\theta s_x)\left(\text{clamp}\left(\left\lfloor \frac{x}{s_x} \right\rceil\right) \cdot \text{clamp}\left(\left\lfloor \frac{\theta}{s_\theta} \right\rceil\right) + \beta\right). \tag{14}$$

To prevent overflow, the product is accumulated in 32-bit registers initialized to the value of the bias β. Scaling parameters are updated based on the running minimum and maximum of their associated full-precision elements:

$$s = \begin{cases} \frac{2\max}{255}, & \text{if } \max > |\min|, \\ \frac{2|\min|}{255}, & \text{otherwise.} \end{cases} \tag{15}$$

After training, full-precision weights are replaced by their quantized counterparts, and the scaling parameters are combined into a single parameter for reverse-quantization, $s_o = s_x * s_\theta$. The activation scaling parameter s_x is retained for quantizing activations during inference. All linear layers, except the transformer's decoder, are quantized, as empirical results indicate superior performance with a full-precision decoder.

5 Results

In this section, we evaluate our deep-learning based approach for SPN value reconstruction and explore the impact of quantization on the performance and efficiency of our LSTM and Transformer architectures. We evaluate our approach across a range of window sizes to demonstrate how the amount of context impacts the efficiency and the ability of each architecture to reconstruct missing SPNs. Window size refers to the number of timesteps that are used during signal reconstruction. Smaller window sizes reduce computational overhead, making them preferable under general attack scenarios. However, focused attacks that target individual ECUs for longer durations may require larger window sizes ($w \geq 30$) for accurate predictions. Therefore, it is essential that the models can perform well with small and large windows.

Window Size	Data Subset			
	Train	Validate	Test	**Total**
10	37457	5351	10702	53510
20	37380	5340	10680	53400
30	37303	5329	10658	53290
40	37226	5318	10636	53180
50	37149	5307	10614	53070
60	37072	5296	10592	52960

Table 1: **Dataset Size.** Number of samples in each subset after partitioning data and applying sliding-window sampling with each window size.

5.1 Experimental Setup

We train our models using random disruption and evaluate them using both random and attack-path-based disruption. We use random disruption during training to provide generalizability by rotating which SPNs are disrupted after every training epoch. Shirazi et al. [40] removed up to 16.6% of SPNs (5 real SPNs and 1 disrupted SPN per model) during training by carefully selecting which SPNs to include in each model based on an analysis of correlations between SPNs. We used a slightly lower mask rate of 10% during training to improve generalizability by not relying on pre-determined correlations between SPNs. During evaluation, we test our models using a range of mask rates to demonstrate their performance in scenarios with varying difficulty.

	Window Size	**Transformer**		**LSTM**	
		FP	Q	FP	Q
Base Mask Rate	10	0.0035	0.0036	0.0032	0.0043
	20	0.0027	0.0029	0.0035	0.0033
	30	0.0028	0.0029	0.0031	0.004
	40	0.0028	0.0029	0.0036	0.0058
	50	0.0026	0.0027	0.0055	0.0051
	60	0.0026	0.0027	0.0046	0.0066
Overall	10	0.0057	0.0058	0.0061	0.0062
	20	0.0042	0.0043	0.0082	0.0054
	30	0.0049	0.005	0.0117	0.0127
	40	0.0047	0.0049	0.0123	0.0192
	50	0.0045	0.0047	0.0156	0.0174
	60	0.0046	0.0047	0.0179	0.0179

Table 2: **Model Performance**. Mean Squared Error with and without quantization with 6 different window sizes. "Overall" shows the average performance across mask rates, "Base Mask Rate" uses the same mask rate as training (10%).

5.1.1 Data Collection

The data was collected by monitoring the diagnostics port of a Kenworth T270 Class 6 heavy vehicle and recording every message transmitted over the CAN bus during a 57 minute drive between two cities. In total, over 2.68 million total J1939 messages were recorded. The raw J1939 traffic was collected and published by Daily et al. [8]. The recorded messages are decoded using a python script and SPN values are extracted from the raw network traffic according to the J1939 standards. We generated a time-series using the collected messages as described in Section 3.2. The generated time-series includes a total of $53,626$ time-steps.

The collected traffic includes 65 different SPNs. For our experiments, we removed all SPNs with fewer than $6,000$ recorded messages or non-continuous values. There were also a few SPNs with monotonically increasing values, which we removed because they are trivial to reconstruct and would artificially inflate our results. Our final selection includes 28 SPNs. As shown in Table 9, our SPN selection covers a range purposes and subsystems in the vehicular network, ensuring the generality of our model.

The observed distributions vary among the selected SPNs, so we standardize their values to improve the stability of our deep-learning models. Let $\hat{X}$ be the observed data. To obtain the model input X, we standardize the

Mask Rate		Transformer FP	Transformer Q	LSTM FP	LSTM Q
5%	Minimum	0.0008	0.0009	0.0003	0.0019
	Average	0.0028	0.0029	0.0044	0.0053
	Maximum	0.009	0.0091	0.0114	0.0116
10%	Minimum	0.0008	0.001	0.0003	0.0018
	Average	0.0028	0.0029	0.0039	0.0049
	Maximum	0.009	0.0091	0.0101	0.0104
15%	Minimum	0.0011	0.0012	0.0004	0.002
	Average	0.0032	0.0033	0.0047	0.0057
	Maximum	0.0093	0.0094	0.0125	0.0123
20%	Minimum	0.0018	0.002	0.0008	0.0027
	Average	0.0041	0.0043	0.0076	0.0089
	Maximum	0.0102	0.0104	0.0217	0.0181
25%	Minimum	0.0033	0.0035	0.0031	0.0061
	Average	0.0061	0.0063	0.0163	0.0174
	Maximum	0.0122	0.0124	0.0384	0.0337
30%	Minimum	0.006	0.0062	0.0096	0.0196
	Average	0.0096	0.0097	0.0349	0.0367
	Maximum	0.0157	0.0159	0.0715	0.0674

Table 3: **Model Performance**. Mean Squared Error with and without quantization with 6 different mask rates.

observed data using min-max scaling as follows:

$$X_{i,j} = \frac{\hat{X}_{i,j} - min(\hat{X}_{*,j})}{max(\hat{X}_{*,j}) - min(\hat{X}_{*,j})} \tag{16}$$

where $X_{i,j}$ is an instance of SPN j at timestep i and $\hat{X}_{*,j}$ denotes the jth column vector of $\hat{X}$.

5.1.2 Data Sampling and Partitioning

We split the time-series into 10 non-overlapping subsequences, then generate input samples for our model by sliding a window of size w over each subsequence. Each input sample is a $[w \times 28]$-dimensional matrix with w rows where each row is a 28-dimensional feature vector containing the most recently observed value for the 28 SPNs at a single timestep. For a subsequence $X \in \mathbb{R}^{N \times 28}$ with N time-steps, sliding window sampling will generate $(N-w)+1$ samples. The sliding window will produce a tensor of samples $X' \in \mathbb{R}^{((N-w)+1) \times w \times 28}$ where $X'_i = \{X_i, X_{i+1}, \ldots, X_{i+w-1}\}$. We use subsequences to prevent overlap between samples that are assigned to different subsets when partitioning the data. In each evaluation trial, we partition the time-series dataset by randomly assigning each subsequence to one of the training, validation or test data subsets. When a subsequence is assigned to a data subset, it will contain all windows generated from that subsequence. We assign $7/10$ subsequences to the training subset, $1/10$ to the validation subset, and $2/10$ to the test subset, resulting in a train:validation:test ratio of 7:1:2. We show the number of window samples in each subset in Table 1. We note that some samples are discarded to ensure that the total number of samples will divide evenly into 10 subsets.

5.2 Performance Evaluation

In this section, we compare our two deep-learning architectures based on their ability to reconstruct signal values and their robustness to quantization. We perform each experiment using 5 different dataset partitions and 2 different seeds for random weight initializations. We evaluate trained models based on their Mean Squared Error (MSE as defined in 11) on the test data subset after applying one of our two masking techniques.

We evaluate the general performance of our approach for SPN reconstruction by testing models using randomized disruption. To reflect a variety of attack paths, we record the average MSE over 10 trials where each trial uses a unique feature mask. This way, each trial will mask a different subset of SPN values in each sample. For each model, we report the average MSE obtained over these 100 trials (10 feature masks * 5 partitions * 2 seeds). We evaluate our approach using 6 different testing mask rates: {5%, 10%, 15%, 20%, 25%, 30%}. Larger mask rates represent more challenging scenarios since there are fewer trustworthy SPN values that the model can use for reconstructing each disrupted SPN.

As shown in Tables 2 and 3, all models are capable of reconstructing SPN values with minimal error rates. On average, the full-precision Transformer has the best performance, followed closely by the quantized Transformer. As we will demonstrate in Section 5.3, the slight increase in error level of the Transformer architecture caused by quantization comes with significant efficiency improvements.

Table 2 shows the performance of our approach with varying window sizes. As the window size increases, the performance of the LSTM degrades due to the limited memory capacity of the cell-state vector. On the other hand, the Transformer performs better with larger window sizes. In scenarios where an SPN malfunctions for longer durations, it is important that the model can effectively interpret long-range dependencies as it may need to pull from older values in the input window to accurately reconstruct the missing values.

As shown in Table 3, our approach is capable of handling various levels of disruption. As expected, both architectures perform better with lower mask rates. How-

ever, the Transformer maintains strong performance even under severe network disruption. The Transformer achieves average error rates under 1% of the standardized SPN operating range with up to 30% of SPNs masked in each sample. For the LSTM, the error rate is only below 1% with mask rates $\leq 20\%$.

Both the Transformer and the LSTM are highly robust to quantization. The quantized Transformer achieves similar error levels to the full-precision Transformer across window sizes and mask rates. The impact of quantization is more significant on the LSTM because it has fewer arithmetic operations per quantized parameter.

5.3 Efficiency

In this section, we evaluate the efficiency of our approach and explore the impact of quantization on the computational and memory cost of our LSTM and Transformer architectures. We start by outlining our method for measuring the computational and storage costs, then summarize our findings for each model.

5.3.1 Computational Cost Estimation

We measure the computational cost of each model by estimating the total number of 32-bit floating point operations (FLOPs) and 8-bit integer multiplications required for SPN reconstruction. As the computational cost of each operation is hardware-dependent, we represent the relative cost of floating-point multiplications as ε and the relative cost of integer multiplications as γ. Quantization improves the computational cost of models by replacing a portion of floating point multiply-accumulate (MAC) operations with cheaper 8-bit integer MAC operations.

For a linear layer $\mathbf{L} : \mathbb{R}^z \rightarrow \mathbb{R}^d$ in a full-precision model, there are no 8-bit multiplications and the FLOPs count can be estimated as: $(d * z * w)$ where w is the window size. For the quantized version of $\mathbf{L}$, we can estimate the integer multiplication count as $(d * z * w)$ and the FLOPs count as $(w * d) + (w * z)$. $(w * d)$ is the cost of scaling the full-precision input to $\mathbf{L}$ during quantization, while $(w * z)$ is the cost of reverse-scaling the output of $\mathbf{L}$.

We estimate the cost of each model as a function of the window size, w, and the relative cost of each operation (ε for floating point and γ for integer). We show the equation for estimating the computational cost of each model in Table 4.

Horowitz et al. [18] observed that the energy cost of 32-bit floating point multiplication is 18.5 times higher than 8-bit integer multiplication. Based on this, we assign a value of 18.5 to ε and a value of 1 to γ in Table 5 to estimate the computational savings of quantization.

Model	Cost Estimate
FP Transformer	$\varepsilon(720w^2 + 78160w + 1120)$
Q Transformer	$\varepsilon(720w^2 + 3188w + 1120) + 77920w\gamma$
FP LSTM	$\varepsilon(121920w + 7560)$
Q LSTM	$\varepsilon(2992w + 308) + \gamma(120320w + 7520)$

Table 4: Simplified equations for estimating the computational cost of each model in terms of the window size w, relative cost of floating-point multiplication ε, and relative cost of 8-bit integer multiplication γ. We show the full derivation of these equations in Appendix 7.2.

5.3.2 Storage Cost

We measure the storage cost in terms of the minimal memory overhead required for storing each model. Reducing the storage cost improves the efficiency of a model by decreasing the time and energy required for loading model parameters onto compute registers/caches and decreasing the energy expended by housing the model in DRAM. Quantization improves storage cost by representing weights using $4\times$ fewer bits.

Let $\mathbf{L} : \mathbb{R}^z \rightarrow \mathbb{R}^d$ be a linear layer in the model with weights $\theta_L \in \mathbb{R}^{z \times d}$ and bias $\beta_L \in \mathbb{R}^d$. The storage cost of $\mathbf{L}$ in a full-precision model will be $32((z+1)d)$ because it stores both the weights and bias using 32-bit values. In the quantized version, the storage cost will be $8zd + 32d + 64$ because it stores the weights using 8-bit integers and the bias using 32-bit values, along with two 32-bit floating point scaling parameters for quantizing the input and reverse-quantizing the output. Quantization reduces the storage cost for $\mathbf{L}$ by a factor of $\frac{4zd}{64}$. In total, the Transformer includes 19 quantized layers while the LSTM includes 6 quantized layers.

5.3.3 Results

Computation. As shown in Table 5, quantization results in significant computational savings for both architectures. For the LSTM, the savings are higher because of the increased amount of parameter-dependent computation. With a window size of 60, the quantized LSTM has the highest savings, reducing the cost by a factor of $12.84\times$. The quantized Transformer achieves a greater reduction in computation cost with smaller window sizes because the attention operation (Equation 10) is performed using full-precision values in both versions

		Transformer			LSTM		
		FP	Q	Savings	FP	Q	Savings
Computation Cost	w=10	15,812,320	2,721,700	5.81×	22,695,060	1,769,938	12.82×
	w=20	34,267,920	8,086,680	4.24×	45,250,260	3,526,658	12.83×
	w=30	55,387,520	16,115,660	3.44×	67,805,460	5,283,378	12.83×
	w=40	79,171,120	26,808,640	2.95×	90,360,660	7,040,098	12.84×
	w=50	105,618,720	40,165,620	2.63×	112,915,860	8,796,818	12.84×
	w=60	134,730,320	56,186,600	2.4×	135,471,060	10,553,538	12.84×
Storage Cost (KB)		2,690.43	820.96	3.28×	4,123.78	1,055.81	3.9×

Table 5: **Storage and computational costs of each model.** Savings are calculated by dividing the cost of the full-precision models by the cost of the quantized version. Computational costs are estimated by counting the total number of 32-bit floating point multiplications and 8-bit integer multiplications and multiplying the floating point operations by 18.5 to represent their higher energy cost as outlined by Horowitz et al. [18]. We show the computational cost with a range of window sizes (w) to show how the savings change as the window size increases.

of the model. The complexity of this operation increases quadratically with window size, which leads to diminishing savings as the window size increases. The window size does not have a significant impact on the computational savings for the LSTM architecture because the number of floating point operations in both versions scales linearly with the window size.

Storage. Quantization results in over 3× reduction in storage cost for both architectures. For the LSTM, the storage savings is 3.9×, which is slightly below the maximum possible savings that can be achieved through a 4× reduction in parameter bit-width. The Transformer also exhibits a considerable reduction in storage cost through quantization, although the savings are less significant than the LSTM. This is because the Transformer has a higher number of parameters that are not quantized including the decoder, the positional-encoding, and the 6 *LayerNorm* layers. The Transformer also has fewer total weights, which are distributed among a higher number of quantized layers, so it must maintain a higher number of 32-bit scaling parameters per quantized weight. Despite this, both versions of the Transformer have a lower storage cost than their equivalent LSTM versions. This is because Transformers are capable of being as expressive as LSTMs with fewer total parameters, as demonstrated in Section 5.2. The full-precision Transformer is 1.53× smaller than the full-precision LSTM, while the quantized Transformer is 1.29× smaller than the quantized LSTM.

Compared to computation, storage cost can be more impactful on the overall efficiency when running these models on embedded devices, which typically have limited cache space and memory bandwidth. With a storage footprint under 1 MB, the quantized Transformer has improved cache utilization, which significantly reduces the time and energy required for memory access and communication.

Shirazi et al. [40]			Our Approach		
Dis. Rate	**MSE**	**Error %**	**Mask Rate**	**MSE**	**Error %**
1/6	0.0293	2.93%	7%	0.0029	0.29%
1/11	0.0254	2.54%	10%	0.0029	0.29%
1/16	0.0246	2.46 %	17%	0.0036	0.36%

Table 6: **Performance Comparison**. Average performance of the approach implemented in [40] compared with our approach with different levels of disruption. For each disruption rate, we evaluate our model using a similar mask rate rounded up to the nearest whole percent.

5.4 Comparison With Prior Work

In this section, we compare the performance of our approach with the state-of-the-art SPN reconstruction approach proposed by Shirazi et al. [40]. Rather than using masking, they trained a separate model for each SPN using other correlated SPNs as input. To enable a fair

comparison, we estimate the disruption rate based on the number of input signals per model. For example, a model in [40] that uses 5 input SPNs to reconstruct 1 disrupted SPN corresponds to a mask rate of 16.6%, as 1 out of 6 correlated SPNs are disrupted.

Shirazi et al. [40] implement 3 disruption rates: (i.) 1/6 5 input SPNs, (ii.) 1/11 10 input SPNs, (iii.) 1/16 15 input SPNs. For each disruption rate, we compare the average performance of the approach in [40] with the performance of our quantized Transformer using a similar mask rate. We use a quantized Transformer as it has the best tradeoff between performance and storage cost. As shown in Table 6, our approach achieves significantly lower error rates than [40] for all disruption rates. Our consistently lower error rates demonstrate that our approach is competitive with the state-of-the-art.

6 Complex Attacks

In this section, we evaluate the ability of our approach to mitigate complex attack scenarios. We use attack-path-based disruption to simulate these complex attacks. This allows us to evaluate multiple attack scenarios targeting separate subsystems across thousands of real samples, which would not be possible if we had implemented these attacks on a test-bed. For our model, we use the quantized Transformer because it achieves the best tradeoff between storage cost and performance as demonstrated in Section 5.2 and 5.3. As described in Section 5.1, each model is trained using random disruption.

We simulate attacks where targeted messages are unavailable for extended periods of time. Therefore, we assume that the model does not have access to any trustworthy values from the affected SPNs within the context window. This forces the model to utilize other SPNs instead of relying on historical values from the disrupted signals. To simulate this, we set the duration of the attack (δ) equal to the window size.

We simulate three different attacks based on potential threat scenarios in a vehicular network. We determine attack paths analytically using a top-down approach as outlined in ISO/SAE 21434 [20]. Attack paths represent the steps that an attacker may take for a threat scenario to be realized. In Table 6.1, we show our attack paths and define which SPNs in our system will be disrupted as a result of the attack. Although these attacks are hypothetical, we ensured that they align with realistic threat scenarios. In each attack, an attacker gains access to the CAN bus, then subverts some critical functionality in the heavy vehicle by exploiting vulnerabilities in the J1939 protocol. The attacker can gain access to the CAN physically through a diagnostic port, such as when the vehicle is parked, or remotely by compromising an ECU with a wireless interface such as the telematics ECU.

Attack	SPN	MSE	Error %
61444	190	0.0129	1.29%
	512	0.0471	4.71%
	513	0.0503	5.03%
	2432	0.0491	4.91%
	Average	0.0398	3.98%
64908	3610	0.0091	0.91%
65266	183	0.0039	0.39%
	184	0.1222	12.22%
	Average	0.063	6.3%

Table 7: **Realistic Attack Performance**. Mean Squared Error of the quantized Transformer on each attack scenario.

6.1 Threat Scenario: Engine Speed Signal Disruption via EEC2 Compromise

In this threat scenario, we simulate a targeted cyberattack against the *EEC2* subsystem of the Engine Control Module in a heavy vehicle. The attacker aims to suppress `SPN 190` (Engine Speed) along with other torque-related SPNs contained in `PGN 61444` during vehicle motion. The attack is stealthy, initially dormant, and triggered only when the vehicle exceeds a predefined speed threshold. This results in the failure of downstream ECUs that rely on these SPNs, potentially impairing subsystems such as braking, diagnostics, and transmission control. We demonstrate how our proposed lightweight deep learning model can reconstruct the missing signals in real-time, preserving safe vehicle operations.

Attack Simulation Narrative

1. **Reconnaissance and Access**: The adversary gains access to the CAN bus through a compromised ECU
2. **Triggering Logic**: When the vehicle exceeds a speed threshold (e.g., 40 km/h), the attack triggers and disables PGN 61444 transmissions.
3. **DoS Attack**: The compromised ECU launches a denial-of-service attack using carefully crafted messages [33] that prevent the ECM from transmitting EEC2 packets.
4. **Message Suppression**: SPNs such as `SPN 190` (Engine Speed), `SPN 512` (Driver Demand Torque), `SPN`

513 (Actual Torque), and SPN 2432 (Engine Demand Percent Torque) are no longer broadcast.
5. **Impact Propagation**: Dependent ECUs experience degraded performance or fault conditions due to the absence of real-time engine speed data.
6. **Recovery and Resilience**: Our embedded ML model reconstructs missing SPNs using a time-series window of other SPN values, enabling operational continuity.

Mitigating Targeted Signal Suppression. We used a quantized Transformer to reconstruct disrupted SPNs from the EEC2. The results, summarized in Table 7, demonstrate the robustness and accuracy of our approach. The model achieved a mean squared error (MSE) of just 0.0129 for SPN 190, corresponding to a remarkably low error rate of 1.29%, indicating highly accurate reconstruction of engine speed. Similarly, for SPNs 512, 513, and 2432, the model maintained error rates below 5.1%, with corresponding MSEs of 0.0471, 0.0503, and 0.0491, respectively. The average reconstruction error across all four SPNs was only 3.98%, showcasing the model's ability to generalize effectively under targeted attack conditions.

6.2 Threat Scenario: DPF Outlet Pressure Signal Tampering via Aftertreatment Controller Compromise

In this scenario, an adversary targets the *Aftertreatment Control Module (ACM)* of a heavy-duty vehicle with the intent to falsify PGN 64908, which carries the mission-critical SPN 3610 (Diesel Particulate Filter—DPF—Outlet Pressure). By broadcasting forged outlet-pressure values that appear nominal, the attacker prevents the engine control logic from detecting soot accumulation. The DPF regeneration cycle is therefore delayed, leading to excessive back-pressure, reduced engine efficiency, and-after several drive cycles—an automatic power-derate or limp-home condition that can immobilize fleet assets and violate emissions regulations.

Attack Simulation Narrative

1. **Reconnaissance and Access**: The adversary gains access to the CAN bus.
2. **Firmware Implantation**: Modified firmware is flashed onto the ACM, embedding logic that remains dormant until the engine warms to operating temperature.
3. **Triggering Logic**: Once exhaust temperature exceeds 250 °C, conditions suitable for passive regeneration, the implant activates and begins transmitting spoofed PGN 64908 frames.
4. **Signal Forgery**: The forged SPN 3610 reports outlet-pressure values lower than actual, masking the true soot load.
5. **Impact Propagation**: Regeneration events are skipped; DPF back-pressure rises, eventually forcing the engine ECU into a staged derate, triggering costly downtime and violation of OBD emissions thresholds.
6. **Detection**: The onboard ML module compares the temporal evolution of SPN 3610 against correlated exhaust-temperature and differential-pressure signals, detecting an anomaly in real time.
7. **Resilience**: Reconstructed pressure values generated by a lightweight quantized Transformer restore trustworthy sensor data, allowing the ECU to initiate regeneration and avoid derate.

Mitigating Outlet-Pressure Manipulation. Using the same quantized Transformer architecture, we reconstruct the disrupted SPN 3610 values from contextual exhaust and engine signals. As summarized in Table 6, the model achieved a mean-squared error (MSE) of 0.0091, corresponding to only 0.91% error across the effective operating range of the DPF outlet-pressure sensor. This sub-one-percent deviation confirms that the lightweight model can deliver near-OEM accuracy under adversarial conditions, enabling timely regeneration and preventing power-derate or limp-home states.

6.3 Threat Scenario: Exhaust Gas Temperature Manipulation via PGN 56266 Compromise

PGN 56266 (0xDBFA) transmits two closely-related exhaust metrics—SPN 183 and SPN 184, representing *Exhaust Gas Temperature 1* and *Exhaust Gas Temperature 2*, respectively. Accurate measurement of these temperatures is essential for turbocharger protection, selective catalytic reduction (SCR) efficiency, and Diesel Particulate Filter (DPF) regeneration scheduling. In this scenario, an adversary compromises the *Aftertreatment Control Module (ACM)* and forges PGN 56266 frames that under-report both exhaust temperatures during periods of prolonged high load. By masking genuine thermal elevation, the attack prevents the engine or aftertreatment logic from triggering protective actions (fuel enrichment, load reduction, or regeneration cooldown), risking com-

#	Attack Phase	Relevant Stndard / Technique		
		PGN 61444	**PGN 64908**	**PGN 65266**
1	Initial Access	ISO/SAE 21434 Clause 9.3		
2	Compromise	Protocol vulnerability exploitation	Unauthenticated firmware update; SAE J3061 §8.6.3	Unauthenticated firmware update; SAE J3061 §8.6.3
3	Dormancy	Condition-based persistence	Temperature-conditioned persistence logic	Load-conditioned persistence logic
4	Disruption Trigger	PGN manipulation; DoS on signal layer	PGN spoofing; `T0803` (Block Command Message)	PGN spoofing; `T0803` (Block Command Message)
5	Propagation	Functional failure propagation; ISO 21434 Clause 8.6	Functional derate; ISO/SAE 21434 Clause 8.6	Over-temperature risk; ISO/SAE 21434 §8.6
6	Detection	Runtime anomaly detection (ML-based)		
7	Resilience	Real-time ML-based signal reconstruction	Real-time ML reconstruction of `SPN 3610`	Real-time ML reconstruction of `SPN 183` & `SPN 184`

Table 8: Attack Phases and Standards Techniques for PGN targets 61444, 65266, and 64908.

ponent over-heating, accelerated catalyst aging, and—in extreme cases—thermal runaway or fire hazard.

Attack Simulation Narrative

1. **Reconnaissance and Access**: The adversary gains access to the CAN bus.
2. **Firmware Implantation**: A modified ACM firmware image is flashed, embedding a dormant payload that monitors engine load and turbo speed.
3. **Triggering Logic**: When engine load exceeds 85% for more than 120s, the implant activates, overriding PGN 56266.
4. **Temperature Forgery**: `SPN 183` and `SPN 184` are transmitted at values ~40 °C below actual, keeping readings in the "safe" band.
5. **Impact Propagation**: Over-temperature events go undetected; ECU avoids derate, catalyst protection, or cooldown. Continuous operation at elevated exhaust temperature accelerates SCR substrate degradation and may breach emissions limits.
6. **Detection**: Anomaly detection model compares exhaust-temperature trends against correlated signals (exhaust mass-flow, turbo speed, fuel rate, etc.) and flags inconsistencies in real time.
7. **Resilience**: Reconstructed temperature estimates replace the forged values, enabling proper protective actions and preserving hardware integrity.

Mitigating Exhaust-Temperature Manipulation. The quantized Transformer was evaluated under the forged-signal attack on PGN 56266. As reported in Table 6, the model reproduced `SPN 183` (Exhaust Gas Temperature 1) with an exceptionally low mean-squared error of 0.0039, equivalent to only 0.39% deviation across the effective temperature range. For `SPN 184` (Exhaust Gas Temperature 2), the reconstruction error was higher—0.1222 MSE or 12.22%—yet still within the accuracy envelope required for aftertreatment protection logic. These results confirm that the lightweight model can deliver near-sensor fidelity for the primary temperature channel and maintain usable accuracy for the secondary channel, thereby restoring trustworthy inputs for thermal safeguards.

7 Conclusion

In this work, we present an approach for mitigating the impact of attacks against vehicular sensor networks through signal reconstruction. Our approach uses a lightweight, unified deep-learning model to reconstruct disrupted signals in a vehicular network in the event of an attack. We implement two alternative deep-learning architectures for this purpose, a Transformer and an LSTM, and evaluate both extensively across a range of different window sizes and mask rates. Our results show that both models can reliably reconstruct disrupted signals, with the Transformer achieving MSE between 0.0008 and 0.0157 and the LSTM achieving MSE between 0.0003 and 0.0715. We additionally employ Quantization-Aware Training to reduce the computational and storage cost of our approach, enabling its deployment on resource-constrained embedded devices that are physically connected to vehicular networks. Through quantization, we achieve computational savings in the range of $[2.4\times, 12.84\times]$ without significantly impacting the performance of the models. Quantization also reduces the storage cost of the Transformer by a factor of $3.27\times$ to 821 KB and the LSTM by a factor of $3.9\times$ to

1.1 MB. Our comparison of the two architectures reveals that the Transformer is superior in terms of efficiency, robustness to quantization, and performance with high mask rates. We simulated three complex attack scenarios and achieved average MSE between 0.0398 and 0.063 using the quantized Transformer.

Future work will include evaluating the runtime efficiency of our approach on hardware-in-the-loop (HIL) platforms. This will involve building and optimizing hardware kernels for SPN reconstruction using the architectures that we explored in this work. Another direction for future work is to explore alternative approaches for improving the efficiency of deep-learning models for sensor reconstruction such as pruning or binarization. Alternative Transformer and LSTM architectures can also be explored to further reduce error levels by improving the contextualization of individual signals.

Acknowledgement

This work was partially supported by NSF under Grant No. CNS 1822118 and CNS 2226232, Award Numbers DMS 2123761, the member partners of the NSF IUCRC Center for Cyber Security Analytics and Automation – AMI, NewPush, Cyber Risk Research, NIST and ARL, and also the State of Colorado Cybersecurity Center (#SB 18-086) and NIST Grant no. 60NANB23D152 and Microsoft Accelerating Foundation Models Research (AFMR).

References

[1] ALIWA, E., RANA, O., PERERA, C., AND BURNAP, P. Cyberattacks and countermeasures for in-vehicle networks. *ACM Comput. Surv. 54*, 1 (Mar. 2021).

[2] BANNER, R., NAHSHAN, Y., HOFFER, E., AND SOUDRY, D. Post-training 4-bit quantization of convolution networks for rapid-deployment, 2019.

[3] BUREAU OF TRANSPORTATION STATISTICS. Moving goods in the united states. https://data.bts.gov/stories/s/Moving-Goods-in-the-United-States/bcyt-rqmu/. Accessed: May 16, 2025.

[4] CHO, K.-T., AND SHIN, K. G. Fingerprinting electronic control units for vehicle intrusion detection. In *Proceedings of the 25th USENIX Conference on Security Symposium* (USA, 2016), SEC'16, USENIX Association, p. 911–927.

[5] CHOUKROUN, Y., KRAVCHIK, E., YANG, F., AND KISILEV, P. Low-bit quantization of neural networks for efficient inference, 2019.

[6] CHU, K.-F., LAM, A. Y. S., AND LI, V. O. K. Deep multi-scale convolutional lstm network for travel demand and origin-destination predictions. *IEEE Transactions on Intelligent Transportation Systems 21*, 8 (2020), 3219–3232.

[7] CUI, Z., KE, R., PU, Z., AND WANG, Y. Deep bidirectional and unidirectional lstm recurrent neural network for network-wide traffic speed prediction, 2019.

[8] DAILY, J. Heavy vehicle can data. https://www.engr.colostate.edu/~jdaily/J1939/candata.html, 2021. Accessed: 2025-05-01.

[9] DAILY, J., NNAJI, D., AND ETTLINGER, B. Securing CAN traffic on J1939 networks. In *Workshop on Automotive and Autonomous Vehicle Security (AutoSec) 2021* (2021).

[10] DEEPSEEK-AI, LIU, A., FENG, B., XUE, B., WANG, B., WU, B., ET AL. Deepseek-v3 technical report, 2024.

[11] DEVLIN, J., CHANG, M.-W., LEE, K., AND TOUTANOVA, K. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)* (2019), pp. 4171–4186.

[12] EL-REWINI, Z., SADATSHARAN, K., SUGUNARAJ, N., SELVARAJ, D. F., PLATHOTTAM, S. J., AND RANGANATHAN, P. Cybersecurity attacks in vehicular sensors. *IEEE Sensors Journal 20*, 22 (2020), 13752–13767.

[13] GAZDAG, A., FERENCZI, C., AND BUTTYÁN, L. Development of a man-in-the-middle attack device for the CAN bus. In *1st Conference on Information Technology and Data Science* (Debrecen, Hungary, 2020).

[14] GORBETT, M., SHIRAZI, H., AND RAY, I. Sparse binary transformers for multivariate time series modeling. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (2023), pp. 544–556.

[15] HERREWEGE, A., SINGELÉE, D., AND VERBAUWHEDE, I. Canauth - a simple, backward compatible broadcast authentication protocol for can bus. p. 7.

[16] HOCHREITER, S. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 6* (04 1998), 107–116.

[17] HOCHREITER, S., AND SCHMIDHUBER, J. Long short-term memory. *Neural Computation 9*, 8 (1997), 1735–1780.

[18] HOROWITZ, M. 1.1 computing's energy problem (and what we can do about it). In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)* (2014), pp. 10–14.

[19] IEHIRA, K., INOUE, H., AND ISHIDA, K. Spoofing attack using bus-off attacks against a specific ECU of the CAN bus. In *2018 15th IEEE Annual Consumer Communications & Networking Conference (CCNC)* (2018), pp. 1–4.

[20] ISO, S. Iso/sae dis 21434: Road vehicles—cybersecurity engineering. *Draft International Standard* (2020).

[21] JACOB, B., KLIGYS, S., CHEN, B., ZHU, M., TANG, M., HOWARD, A., ADAM, H., AND KALENICHENKO, D. Quantization and training of neural networks for efficient integer-arithmetic-only inference, 2017.

[22] JEPSON, J., CHATTERJEE, R., AND DAILY, J. Commercial Vehicle Electronic Logging Device Security: Unmasking the Risk of Truck-to-Truck Cyber Worms. In *Symposium on Vehicles Security and Privacy (VehicleSec) 2024* (San Diego, CA, Feb. 2024), NDSS.

[23] JICHICI, C., GROZA, B., RAGOBETE, R., MURVAY, P.-S., AND ANDREICA, T. Effective intrusion detection and prevention for the commercial vehicle SAE J1939 CAN bus. *IEEE Transactions on Intelligent Transportation Systems 23*, 10 (2022), 17425–17439.

[24] KIM, Y., DENTON, C., HOANG, L., AND RUSH, A. M. Structured attention networks, 2017.

[25] KINGMA, D. P., AND BA, J. Adam: A method for stochastic optimization, 2017.

[26] KOSCHER, K., CZESKIS, A., ROESNER, F., PATEL, S., KOHNO, T., CHECKOWAY, S., MCCOY, D., KANTOR, B., ANDERSON, D., SHACHAM, H., AND SAVAGE, S. Experimental security analysis of a modern automobile. In *2010 IEEE Symposium on Security and Privacy* (2010), pp. 447–462.

[27] LI, W., AND LAW, K. L. E. Deep learning models for time series forecasting: A review. *IEEE Access 12* (2024), 92306–92327.

[28] LINDEMANN, B., MÜLLER, T., VIETZ, H., JAZDI, N., AND WEYRICH, M. A survey on long short-term memory networks for time series prediction. *Procedia CIRP 99* (2021), 650–655. 14th CIRP Conference on Intelligent Computation in Manufacturing Engineering, 15-17 July 2020.

[29] LIU, C.-L., HSAIO, W.-H., AND TU, Y.-C. Time series classification with multivariate convolutional neural network. *IEEE Transactions on Industrial Electronics 66*, 6 (2019), 4788–4797.

[30] LOPEZ-MARTIN, M., CARRO, B., SANCHEZ-ESGUEVILLAS, A., AND LLORET, J. Network traffic classifier with convolutional and recurrent neural networks for internet of things. *IEEE Access PP* (09 2017), 18042–18050.

[31] MIENYE, I. D., SWART, T. G., AND OBAIDO, G. Recurrent neural networks: A comprehensive review of architectures, variants, and applications. *Information 15*, 9 (2024).

[32] MOHAJERIN, N., AND WASLANDER, S. L. State initialization for recurrent neural network modeling of time-series data. In *2017 International Joint Conference on Neural Networks (IJCNN)* (2017), pp. 2330–2337.

[33] MUKHERJEE, S., SHIRAZI, H., RAY, I., DAILY, J., AND GAMBLE, R. Practical dos attacks on embedded networks in commercial vehicles. In *Information Systems Security: 12th International Conference, ICISS 2016, Jaipur, India, December 16-20, 2016, Proceedings 12* (2016), Springer, pp. 23–42.

[34] MUKHERJEE, S., WALKER, J., RAY, I., AND DAILY, J. A precedence graph-based approach to detect message injection attacks in J1939 based networks. In *2017 15th Annual Conference on Privacy, Security and Trust (PST)* (2017), IEEE, pp. 67–6709.

[35] MURVAY, P.-S., AND GROZA, B. Security shortcomings and countermeasures for the sae j1939 commercial vehicle bus protocol. *IEEE Transactions on Vehicular Technology 67*, 5 (2018), 4325–4339.

[36] NAGEL, M., FOURNARAKIS, M., AMJAD, R. A., BONDARENKO, Y., VAN BAALEN, M., AND BLANKEVOORT, T. A white paper on neural network quantization, 2021.

[37] NILSSON, D. K., LARSON, U. E., AND JONSSON, E. Efficient in-vehicle delayed data authentication based on compound message authentication codes. In *2008 IEEE 68th Vehicular Technology Conference* (2008), pp. 1–5.

[38] PASCANU, R., MIKOLOV, T., AND BENGIO, Y. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28* (2013), ICML'13, JMLR.org, p. III–1310–III–1318.

[39] RADFORD, A., AND NARASIMHAN, K. Improving language understanding by generative pre-training.

[40] SHIRAZI, H., PICKARD, W., RAY, I., AND WANG, H. Towards resiliency of heavy vehicles through compromised sensor data reconstruction. In *Proceedings of the Twelfth ACM Conference on Data and Application Security and Privacy* (New York, NY, USA, 2022), CODASPY '22, Association for Computing Machinery, p. 276–287.

[41] VASWANI, A., SHAZEER, N., PARMAR, N., USZKOREIT, J., JONES, L., GOMEZ, A. N., KAISER, L., AND POLOSUKHIN, I. Attention is all you need, 2023.

[42] WEN, Q., ZHOU, T., ZHANG, C., CHEN, W., MA, Z., YAN, J., AND SUN, L. Transformers in time series: A survey, 2023.

[43] XUE, H., HUYNH, D. Q., AND REYNOLDS, M. Ss-lstm: A hierarchical lstm model for pedestrian trajectory prediction. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)* (2018), pp. 1186–1194.

[44] ZERVEAS, G., JAYARAMAN, S., PATEL, D., BHAMIDIPATY, A., AND EICKHOFF, C. A transformer-based framework for multivariate time series representation learning. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining* (New York, NY, USA, 2021), KDD '21, Association for Computing Machinery, p. 2114–2124.

Appendix

7.1 SPN Information

PGN	SPN	Description	Min	Max	Unit
61443	91	Accelerator Pedal Position #1	0	100	%
	92	Engine Percent Load At Current Speed	0	250	%
	5398	Estimated Pumping – Percent Torque	-125	125	%
61444	190	Engine Speed	0	8,031.88	rpm
	512	Driver's Demand Engine – Percent Torque	-125	125	%
	513	Actual Engine – Percent Torque	-125	125	%
	2432	Engine Demand % Percent Torque	-125	125	%
64443	3357	Actual Maximum Available Engine – Percent Torque	0	100	%
65247	514	Nominal Friction – Percent Torque	-125	125	%
65245	103	Engine Turbocharger 1 Speed	0	257,020	rpm
65262	110	Engine Coolant Temperature	-40	210	°C
	100	Engine Oil Pressure	0	1,000	kPa
	101	Engine Crankcase Pressure	-250	252	kPa
65264	187	Power Takeoff Set Speed	0	8,031.88	rpm
65266	183	Engine Fuel Rate	0	3,212.75	l/h
	184	Engine Instantaneous Fuel Economy	0	125.5	km/L
65270	102	Engine Intake Manifold #1 Pressure	0	500	kPa
65271	168	Battery Potential / Power Input #1	0	3,212.75	V
61450	132	Engine Inlet Air Mass Flow Rate	0	3,212.75	kg/h
61454	3216	Aftertreatment #1 Intake NOx	-200	3,012.75	ppm
64931	641	Engine Turbocharger Variable Geometry Actuator #1	0	100	%
64908	3610	DPF Outlet Pressure Sensor	0	6,425.5	kPa
64946	3251	Aftertreatment #1 DPF Trap Differential Pressure	0	6,425.5	kPa
64800	4765	Aftertreatment #1 Diesel Oxidation Catalyst Intake Gas Temperature	-273	1,735	°C
65178	1172	Engine Turbocharger 1 Compressor Inlet Temperature	-273	1,735	°C
65170	1209	Engine Exhaust Gas Pressure	-250	252	kPa
	1242	Instantaneous Estimated Brake Power	0	32,127.5	kW

Table 9: **Selected SPNs**. The Suspect Parameter Number, description, collected data range (split into Min and Max), and unit for all 28 sensors selected for our experiments.

7.2 Computational Cost Estimation

We derive our equations for estimating the computational cost of our models (shown in Table 4) by counting the number of 32-bit and 8-bit integer multiply, divide or multiply/accumulate (MAC) operations. For the full-precision LSTM, the number of floating point operations is:

$$\begin{aligned} G &= (28+160)*160+160 = 30240 \\ CT &= 320 \\ HT &= 320 \\ LayerNorm &= 2*160 = 320 \\ C &= 4G+CT+HT+LayerNorm = 121920 \\ FC &= (160*40)+40+(40*28) = 7560 \\ FLOPs &= wC+FC = 121920w+7560 \end{aligned} \tag{17}$$

For the quantized LSTM, the number of floating point operations is:

$$\begin{aligned} G &= 28+160+160+160 = 508 \\ CT &= 320 \\ HT &= 320 \\ LayerNorm &= 2*160 = 320 \\ C &= 4G+CT+HT+LayerNorm = 2992 \\ FC &= 160+40+40+40+28 = 308 \\ FLOPs &= wC+FC = 2992w+308 \end{aligned} \tag{18}$$

while the number of 8-bit integer operations is:

$$\begin{aligned} G &= (28+160)*160 = 30080 \\ C &= 4G = 120320 \\ FC &= (160*40)+(40*28) = 7520 \\ intOps &= wC+FC = 120320w+7520 \end{aligned} \tag{19}$$

For the full-precision Transformer, the number of floating point operations is:

$$\begin{aligned} p &= (w*28*40) = 1120w \\ MHA &= (40*96*4)w+96w^2+24w^2+24w^2+96w^2 \\ &= 15360w+240w^2 \\ FC &= w*40*128+w*40*128 \\ &= 10240w \\ LayerNorm &= 2*40w = 80w \\ FLOPs &= P+3(MHA+2LayerNorm+FC)+(28*40) \\ &= 1120w+3(15360w+240w^2+160w+10240w)+(28*40) \\ &= 1120w+77280w+720w^2+1120 \\ &= 720w^2+78160w+1120 \end{aligned} \tag{20}$$

For the quantized Transformer, the number of floating point operations is:

$$\begin{aligned} p &= 28w+40w = 68w \\ MHA &= 4(40w+96w)+96w^2+24w^2+24w^2+96w^2 \\ &= 544w+240w^2 \\ FC &= 2(40w+128w) \\ &= 336w \\ LayerNorm &= 2*40w = 80w \\ FLOPs &= P+3(MHA+2LayerNorm+FC)+(28*40) \\ &= 68w+3(544w+240w^2+160w+336w)+(28*40) \\ &= 68w+3120w+720w^2+1120 \\ &= 720w^2+3188w+1120 \end{aligned} \tag{21}$$

and the number of 8-bit integer operations is:

$$\begin{aligned} p &= (w*28*40) = 1120w \\ MHA &= (40*96*4)w = 15360w \\ FC &= w*40*128+w*40*128 \\ &= 10240w \\ intOps &= P+3(MHA+FC) \\ &= 1120w+3(15360w+10240w) \\ &= 77920w \end{aligned} \tag{22}$$