Vehicle-Model Agnostic Intrusion Detection System through Zero-Shot Temporal Learning on BERT

Md Rezanur Islam, Donghyun Ryu, Munkhdelgerekh Batzorig, Kangbin Yim

***Abstract*—**

***Index Terms*—Article submission, IEEE, IEEEtran, journal, LaTeX, paper, template, typesetting.**

I. Introduction

THIS

This research was supported by the MSIT(Ministry of Science and ICT), Korea, under the Convergence security core talent training business support program (IITP-2024-2710008611) supervised by the IITP(Institute for Information & Communications Technology Planning & Evaluation) and Soonchunhyang University Research Fund. (Corresponding author: Kangbin Yim.)

Md Rezanur Islam, Donghyun Ryu, Munkhdelgerekh Batzorig, and Kangbin Yim are with the Department of Software Convergence and the Department of Information Security Engineering, Soonchunhyang University, Asan-si, Korea (arupreza@sch.ac.kr; rdh1999@sch.ac.kr; munkhdelgerekh@sch.ac.kr; yim@sch.ac.kr).

[1]

II. Evolution of In-Vehicle Network Architectures and Security Implications

ICEVs rely on fossil fuel combustion to convert chemical energy into mechanical propulsion via a powertrain managed by a relatively simple electronic architecture. The Engine Control Unit (ECU/ECM) serves as the primary controller, optimizing fuel injection, ignition, and emissions based on sensor data. Supporting this are the Transmission Control Module (TCM), which manages gear shifts and clutch engagement, and the Body Control Module (BCM), which handles lighting, locks, and climate systems. These modules communicate via a Controller Area Network (CAN) bus limited to 1Mbps bandwidth and operate on a standard 12V electrical system. While traditionally characterized by low software dependency and mechanical priority, modern ICEVs increasingly integrate advanced networking for driver assistance systems [2].

Hybrid Electric Vehicles (HEVs) integrate internal combustion and electric propulsion by adding a central Hybrid Control Unit (HCU) to the traditional Engine Control Module (ECM) and Transmission Control Module (TCM) architecture [3]. To manage the complex coordination between these dual power sources, HEVs utilize CAN-FD (8 Mbps, 64-byte payload) alongside standard CAN (1 Mbps), providing the necessary bandwidth for real-time torque splitting and regenerative braking decisions. This enhanced network structure allows the HCU to seamlessly transition between power modes and manage medium-voltage systems (48V-200V), ensuring optimal efficiency and safety transparency [4].

BEVs represent a paradigm shift in automotive design, replacing the internal combustion engine with a software-centric architecture powered by high-voltage batteries and managed by the Powertrain Control Module (PCM) and Battery Management System (BMS). Unlike traditional ICEVs that rely on standard CAN (1 Mbps, 8-byte payload), the BEV architecture detailed in the diagram implements high-bandwidth protocols like CAN-FD and CAN-XL to handle

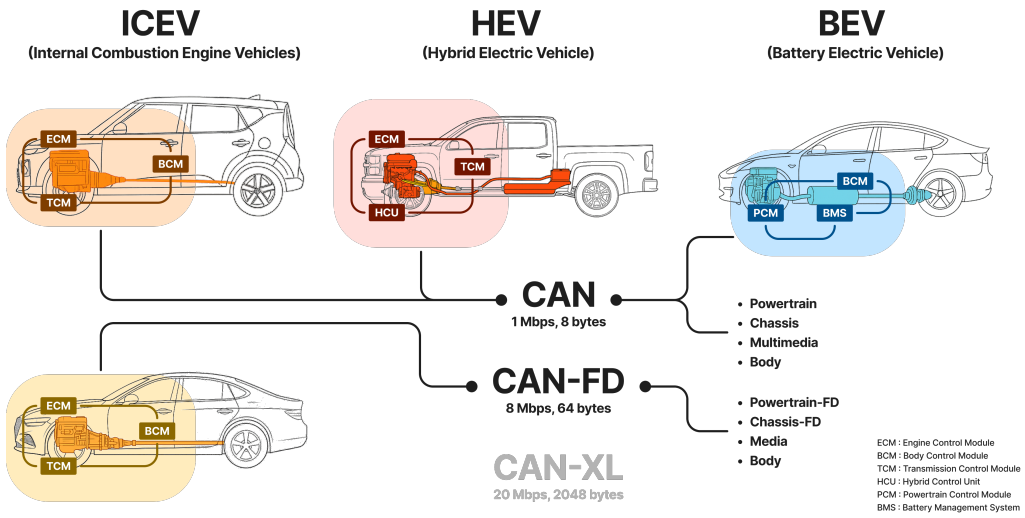


Fig. 1: Comparison of Vehicle Communication Network Architectures Across Different Automotive Types

complex data across Powertrain, Chassis, and Multimedia domains. CAN-FD significantly increases throughput to 8 Mbps with 64-byte payloads, while the advanced CAN-XL scales further to 20 Mbps with 2048-byte payloads. This robust network capacity is essential for managing 400V+ systems, coordinating regenerative braking, and supporting the massive data flow required for Over-The-Air (OTA) updates and modern cybersecurity measures [5].

The progression from classic CAN to CAN-FD and CAN-XL addresses increasing automotive networking demands while maintaining backward compatibility. Classic CAN, operates at 1 Mbps with 8-byte payloads and employs multi-master arbitration based on message identifier priority—lower identifiers gain bus access during contention, ensuring deterministic behavior for time-critical communications. CAN-FD (2012) preserves the same arbitration mechanism but implements dual-phase operation: arbitration at standard CAN speeds (1 Mbps) for compatibility, then switches to 8 Mbps for data transmission. This increases throughput eightfold while extending payloads to 64 bytes and implementing enhanced CRC-17/CRC-21 error detection. CAN-XL (2019) dramatically expands capabilities to 20 Mbps with 2048-byte payloads, introduces an 11-bit priority field, and adds addressing mechanisms enabling point-to-point communication alongside traditional broadcasting. Unlike classic CAN and CAN-FD where all nodes receive all messages, CAN-XL's targeted communication reduces processing overhead and supports service-oriented architectures. Additionally, CAN-XL integrates security provisions including frame protection and message authentication—critical features absent in earlier variants. This evolution reflects the automotive industry's transition from simple sensor-actuator networks in ICEVs to software-defined platforms in BEVs demanding higher bandwidth, larger data structures, and integrated security.

Despite these architectural advancements, the CAN protocol's fundamental lack of encryption or authentication mechanisms creates critical security vulnerabilities across all vehicle types. While numerous Intrusion Detection System (IDS) solutions have been proposed, widespread deployment faces a significant barrier: architectural fragmentation. Each vehicle platform (ICEV, HEV, BEV) implements distinct ECU distributions and communication rules defined by proprietary CAN DBC files, forcing current IDS solutions to be model-specific and requiring resource-intensive customization for each platform. Furthermore, sophisticated attackers exploit this limitation by varying message injection frequencies to evade detection, challenging static IDS models that struggle to adapt to dynamic attack patterns without generating excessive false alarms. This architectural heterogeneity necessitates a Universal Intrusion Detection System (UIDS) capable of functioning across different vehicle types by leveraging fundamental communication principles shared across CAN variants, while adapting detection thresholds and analysis techniques to accommodate the distinct timing patterns, payload sizes, and communication frequencies characteristic of each protocol implementation.

III. Related Work

IV. Temporal Normalization and Segmentation of CAN Traffic

To standardize heterogeneous CAN traffic, a structured preprocessing and segmentation pipeline is implemented. Each CAN frame is represented as $f_i = (t_i, c_i)$, where t_i is the timestamp and c_i the CAN identifier. Two temporal features are derived to capture both global and intra-identifier dynamics:

$$\Delta t_i = t_i - t_{i-1}, \quad \Delta t_i^{(c)} = t_i - t_{j(c)}, \tag{1}$$

where Δt_i represents the global inter-frame delay (*Time Delta*) and $\Delta t_i^{(c)}$ represents the intra-identifier delay (*Intra-ID Time Gap*) relative to the last occurrence of the same CAN ID.

Both features are normalized using discrete bin-based mappings to preserve ordinal relationships while mitigating noise sensitivity. The normalization function for the Intra-ID Time Gap is defined as:

$$\text{IntraIDTimeGapNorm}(x) = \begin{cases} 0, & x \leq 5.1, \\ 1, & 5.1 < x \leq 10.1, \\ 2, & 10.1 < x \leq 20.1, \\ 3, & 20.1 < x \leq 30.1, \\ 4, & 30.1 < x \leq 40.1, \\ 5, & 40.1 < x \leq 50.1, \\ 6, & 50.1 < x \leq 2010.1, \\ 7, & 2010.1 < x \leq 5010.1, \\ 8, & x > 5010.1. \end{cases} \tag{2}$$

Similarly, the Time Delta normalization is defined as:

$$\text{TimeDeltaNorm}(x) = \begin{cases} 0, & 0 \leq x \leq 0.05, \\ 1, & 0.05 < x \leq 0.1, \\ 2, & 0.1 < x \leq 0.2, \\ 3, & 0.2 < x \leq 0.3, \\ 4, & 0.3 < x \leq 0.4, \\ 5, & 0.4 < x \leq 0.5, \\ 6, & x > 0.5. \end{cases} \tag{3}$$

After normalization, the CAN traffic is segmented into fixed-duration temporal windows of size τ. Let the observation interval be $[t_{\min}, t_{\max})$; segmentation is performed as:

$$W_k = [t_k, t_k + \tau), \quad k = 1, 2, \ldots, K, \tag{4}$$

where each window W_k aggregates frames with timestamps $t_i \in W_k$. For each segment, a feature matrix is constructed as:

$$X_k = \{(\text{Time } \Delta \text{ Norm}, \text{Intra-ID Gap Norm})\}_{t \in W_k}, \tag{5}$$

and a binary label $y_k \in \{0, 1\}$ is assigned such that:

$$y_k = \begin{cases} 1, & \text{if any frame in } W_k \text{ is labeled as an attack,} \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

To align data generation frequencies across heterogeneous in-vehicle datasets, each vehicle log was segmented into fixed-duration time chunks using dataset-specific parameters (τ_v), such as 160 s for *Subaru Forester*, 100 s for *Kia Soul* and *Chevrolet Silverado*, 125 s for *Kia Soul*, and 83 s for *Tesla*. The segmentation function iteratively partitions the time domain $[t_{\min}, t_{\max})$ into windows defined as:

$$W_k = [t_k, t_k + \tau_v),$$

retaining only segments containing sufficient frame density (> 265 messages) to ensure statistically meaningful temporal representation. This adaptive tuning of τ_v harmonizes message-rate variability and produces comparable temporal granularity across datasets collected under different hardware configurations and driving environments.

The datasets used in this study were collected from multiple vehicle types representing different powertrain architectures—Internal Combustion Engine Vehicles (ICEV), Hybrid Electric Vehicles (HEV), and Battery Electric Vehicles (BEV)—as illustrated in Figure 1. Each vehicle type utilizes distinct communication frameworks, including traditional CAN, CAN-FD, and emerging CAN-XL protocols, covering domains such as powertrain, chassis, multimedia, and body systems. Data collection was conducted under varying operational and attack conditions, encompassing Denial-of-Service (DoS), Fuzzing, and Replay scenarios with multiple injection intensities ranging from lower-low to high, as summarized in Table I. This comprehensive dataset composition ensures representation of diverse communication behaviors and network loads across heterogeneous vehicular environments. Such variability enables robust training and evaluation of the proposed BERT-based Intrusion Detection System, facilitating accurate and generalizable detection of malicious activities across different automotive architectures and communication standards.

Figure **??** illustrates the resulting attack-free value distributions across datasets after harmonized segmentation. The tightest and most stationary baseline appears in the *Hyundai Sonata (HCRL)* dataset, which exhibits the smallest interquartile range (IQR) and short whiskers, indicating highly stable benign message generation. The *Subaru Forester (DTU)* distribution is similarly compact with only a few mild upper

TABLE I: Message Injection Statistics for DoS, Fuzzing, and Replay Attack Scenarios

Dataset	Total Messages	Injected Messages	Injection (%)
Denial-of-Service (DoS) Attacks			
DoS High	760,000	160,000	21.05%
DoS Medium	700,000	100,000	14.29%
DoS Low	640,000	40,000	6.25%
DoS Lower-Low	618,100	18,179	2.94%
DoS Random	690,362	90,362	13.09%
Fuzzing Attacks			
Fuzz High	760,000	160,000	21.05%
Fuzz Medium	700,000	100,000	14.29%
Fuzz Low	640,000	40,000	6.25%
Fuzz Lower-Low	618,181	18,181	2.94%
Fuzz Random	690,054	90,054	13.05%
Replay Attacks			
Replay High	760,000	160,000	21.05%
Replay Medium	700,000	100,000	14.29%
Replay Low	640,000	40,000	6.25%
Replay Lower-Low	618,100	18,179	2.94%
Replay Random	689,663	89,663	13.00%

Fig. 2: CAN Simulator

outliers, confirming consistent data periodicity. The *Kia Soul (HCRL)* distribution shows moderate spread, while the *Kia Soul (LISA)* variant shifts upward and widens noticeably, suggesting site- or sensor-protocol differences in normal traffic capture. *Chevrolet Silverado (DTU)* is centered near the Kia datasets but exhibits a slightly elongated upper tail. *Tesla (LISA)* displays higher median values and broader whiskers, typical of electric vehicle (EV) communication dynamics. Finally, *Genesis (LISA)* presents the largest benign variability (wide IQR and outliers up to approximately 281–282), reflecting heterogeneous configurations or operating conditions during data acquisition.

From a powertrain classification perspective, *Tesla (LISA)* represents a battery electric vehicle (BEV), while the remaining datasets (*Subaru Forester*, *Chevrolet Silverado*, *Hyundai Sonata*, *Genesis*, and both *Kia Soul* sets) are likely internal combustion engine (ICE) platforms. No datasets are explicitly tagged as hybrid; although hybrid trims exist in production vehicles, these logs should be treated as ICE unless metadata states otherwise. For model calibration, *Sonata (HCRL)* serves as the most reliable benign baseline, while higher-variance sources like *Genesis (LISA)* and *Tesla (LISA)* require more conservative normalization and domain-shift handling between sites.

In this study, we developed a testbed system designed to safely reproduce cyber attacks in a CAN(Controller Area Network) communication environment and effectively design an IDS(Intrusion Detection System). This system provides an isolated environment capable of repeatedly reproducing various attack scenarios without affecting actual vehicle networks.

V. System Architecture

The testbed consists of three main components: Transmitter, Attacker, and Receiver. Each component operates as an independent CAN node and is interconnected through a standard CAN bus.

A. Transmitter Node

The Transmitter is responsible for generating and transmitting normal CAN messages. It emulates the operation of actual vehicle ECUs (Electronic Control Units), generating periodic message transmission patterns and specific event-based messages. This establishes a baseline for normal traffic patterns. The Transmitter generates standard CAN frames supporting both Standard 11-bit and Extended 29-bit IDs, and can transmit messages at various intervals ranging from 10ms to 1000ms. Additionally, it generates various CAN IDs and data patterns to faithfully simulate normal traffic in actual vehicle networks.

B. Attacker Node

The Attacker node is a core component that simulates various cyber attack scenarios. This node is connected to the CAN bus and can inject malicious messages or disrupt normal communication. In this system, we have implemented Fabrication Attack, Modification Attack, Replay Attack, DoS(Denial of Service), and Fuzzing Attack. The Fabrication Attack involves forging and transmitting messages from non-existent ECUs, while the Modification Attack intercepts normal messages, alters their content, and retransmits them. The Replay Attack confuses the system by retransmitting previously captured normal messages at inappropriate times, and the DoS attack monopolizes the bus by transmitting large volumes of high-priority messages. Finally, the Fuzzing Attack explores system vulnerabilities by transmitting messages containing random data.

C. Receiver Node

The Receiver is a node that receives and analyzes all messages on the CAN bus, equipped with core IDS functionality. This node monitors traffic in real-time and executes detection algorithms to distinguish between normal and attack traffic. The Receiver continuously monitors all traffic on the CAN bus and records timestamps for each message to analyze temporal patterns. It performs statistical anomaly detection by analyzing message frequency, periodicity, and data patterns, and can identify sophisticated attacks by applying machine learning-based classification algorithms. Furthermore, when anomalies are detected, it generates real-time alerts to enable immediate response.

D. CAN Bus Infrastructure

The three nodes are connected by a physical bus compliant with the standard CAN 2.0B protocol. The bus speed is set to 500 kbps, with 120Ω termination resistors installed at both ends to prevent signal reflection. Each node interfaces with the bus through CAN transceivers, communicating via differential signals (CAN High/Low).

VI. System Design Objectives

A. Safety

The most important design goal of this testbed is safety. Directly connecting to and simulating attacks on actual vehicles could have critical impacts on vehicle control systems. By conducting all experiments in a completely isolated environment, various attack scenarios can be safely tested without physical risk.

B. Reproducibility

Reproducibility of experiments is essential in scientific research. This system is designed to execute identical attack scenarios under precisely identical conditions repeatedly. This enables consistent evaluation of IDS algorithm performance and quantitative measurement of improvements.

C. IDS Development and Validation

The testbed serves as a platform for developing and validating intrusion detection systems. By generating both normal traffic and various attack traffic, it enables the construction of labeled datasets for machine learning model training. Additionally, real-time detection performance can be evaluated to optimize False Positive Rates and False Negative Rates.

D. Security Vulnerability Research

The CAN protocol has inherent vulnerabilities because security was not considered during its design. This testbed enables systematic analysis of these vulnerabilities and research into effective countermeasures.

VII. System Implementation

A. Hardware Configuration

This testbed is built on the NVIDIA Jetson AGX Xavier platform. The Jetson AGX Xavier includes an integrated CAN Controller, eliminating the need for a separate CAN Controller chip. For the CAN physical layer interface, we utilized the SN65HVD230 CAN transceiver, which converts the digital signals from the CAN Controller into differential signals (CAN High/Low). The Jetson AGX Xavier is equipped with an 8-core ARM v8.2 64-bit CPU and a 512-core Volta GPU with Tensor Cores, providing sufficient computational power to process complex machine learning algorithms in real-time. The SN65HVD230 transceiver operates at a 3.3V operating voltage, complies with the ISO 11898-2 standard, and supports communication speeds up to 1 Mbps. The system operates at a CAN 2.0B communication speed of 500 kbps, with 120Ω termination resistors installed at both ends of the bus to prevent signal reflection. Power supply uses both 12V DC for vehicle standard voltage and 5V DC for driving the Jetson Xavier.

B. Software Configuration (CAN Simulator System Design)

This study designed a software-based CAN simulator to establish a safe and controllable experimental environment that accurately reproduces real-world vehicular CAN networks. The simulator comprises three independent nodes sharing a single CAN bus interface (`can0`), with each node performing the role of transmitter, receiver, or attacker. Each node operates independently while interacting through the common network interface, enabling simultaneous reproduction of legitimate communication and attack scenarios.

1) Transmitter Node Implementation: The transmitter node faithfully reproduces CAN traffic collected from actual vehicles in the experimental environment. Implemented as `CAN_transmitter.cpp`, this module accepts Vector TRC format log files as input, skips header lines, and loads up to 3 million CAN frames into memory. During parsing, each frame is decomposed into CAN ID, Data Length Code (DLC), and data fields, with automatic identification of extended identifiers. To ensure temporal accuracy, timestamps from the log file are normalized into monotonically increasing form, and transmission timing is controlled at microsecond precision using the POSIX `CLOCK_MONOTONIC` clock. This mechanism precisely replicates the inter-frame gaps and periodicity recorded in the original logs. Prior to transmission, the initialization phase verifies the `can0` link status and registers a SIGINT signal handler for graceful termination. Upon user confirmation via Enter key, the preloaded frame sequence is transmitted to the CAN bus while maintaining original timing characteristics, establishing baseline traffic for subsequent attack and detection experiments.

2) Receiver Node Implementation: The receiver node functions as a monitoring module that captures all CAN bus traffic in real-time and records it in an analyzable format. Implemented as `Unified_Receiver.cpp`, this node opens the `can0` socket in SocketCAN RAW mode and supports CAN FD (Flexible Data-rate) frames to accommodate variable-length DLC. At program startup, a `can_receive_logs` directory is automatically created, and a CSV log file with a unique timestamp-based filename is generated. The log records message sequence numbers, reception time offsets, extended frame indicators, DLC values, 8-byte data fields, and inter-frame time differences (Δt). The main reception loop reads frames in real-time via the `read()` system call, automatically distinguishes between standard CAN frames (`CAN_MTU`) and CAN FD frames (`CANFD_MTU`), and forwards them to the `log_message()` function. Within the logging function, CAN ID masking identifies extended identifiers, data fields are converted to hexadecimal with padding, and cumulative time is calculated. These detailed logs are simultaneously displayed on the console and written to files, providing a foundation for quantitative analysis of anomalies such as periodicity changes, data corruption, and bus load abnormalities, even in scenarios where legitimate and malicious traffic coexist.

3) Attacker Node Implementation: The attacker node is a research-oriented module implementing diverse CAN bus attack vectors, implemented as `CAN_attacker.cpp` and designed using C++17 standards and POSIX threads. The core class, `CANMessageAttacker`, encapsulates functionalities

including socket initialization, frame transmission, random number generation, pattern learning, and timing tracking, providing eight attack methods. The DoS (Denial of Service) attack floods the bus with high-frequency frames using randomized delays within specified ranges (10–50ms, 100–500ms, 1–5s, or adaptive mode). The fuzzing attack generates random CAN IDs and data fields to probe ECU exception handling logic and parser vulnerabilities. The replay attack captures legitimate traffic via the `startRecording()` method for a specified duration, then retransmits the identical frame sequence in original order. The fabrication attack injects arbitrary data under predefined CAN IDs to impersonate non-existent ECUs, while the malfunction simulation attack targets CAN IDs corresponding to specific system functions and injects anomalous data to induce malfunctions. The spoofing attack continuously impersonates a single target CAN ID to manipulate the state of receiving ECUs, and the masquerade attack observes and learns data patterns of the target ID to generate data similar to the legitimate ECU. The enhanced masquerade attack employs background threads to monitor target ECU activity in real-time, performing frequency analysis and timing pattern learning. When the legitimate ECU becomes active, it transmits priority frames continuously to win bus arbitration and suppress original messages, then inserts sophisticatedly crafted masquerade frames based on learned periodicity and data distributions. All attack routines utilize precise timing control via `std::this_thread::sleep_for()` and uniform distribution random number generation based on `std::mt19937` to reflect the non-deterministic characteristics of real attack scenarios. A SIGINT signal handler ensures safe termination of attack loops and resource cleanup.

4) System Integration and Experimental Workflow: The three nodes execute independently at the process level but interact in real-time through the shared virtual CAN interface `can0`. The experiment begins with an initialization phase where the virtual CAN interface is created and activated using commands `sudo modprobe vcan`, `sudo ip link add dev can0 type vcan`, and `sudo ip link set can0 up`. Subsequently, the transmitter node is executed first to reproduce legitimate traffic and establish a baseline, while the receiver node is launched in parallel to initiate logging. The attacker node is then executed with a specific attack vector selected to inject attacks, and CSV logs are analyzed after experiment completion for data collection. This closed-loop experimental environment provides a systematic structure that clearly separates log-based reproduction (Transmitter), real-time observation and instrumentation (Receiver), and attack vector verification (Attacker), enabling step-by-step documentation of experimental procedures and effects. Notably, it enables safe validation of CAN Intrusion Detection System (IDS) effectiveness and defense mechanisms without requiring physical vehicles or expensive ECU hardware, ensuring experimental reproducibility and scalability.

VIII. MobileBERT-based Fine-Tuned CAN-Attack Classifier

A. Model Architecture

MobileBERT is a compact, task-agnostic transformer architecture derived from $\text{BERT}_{\text{LARGE}}$, designed for high efficiency on resource-constrained edge devices. It maintains functional equivalence with the original BERT transformer but achieves substantial compression through architectural distillation and parameter bottlenecking. Specifically, MobileBERT retains $L = 24$ transformer encoder layers while reducing the hidden dimension from 1024 (in $\text{BERT}_{\text{LARGE}}$) to 512, and introduces an inverted-bottleneck structure that separates internal feed-forward dimensionality ($d_{ff} = 3072$) from the external hidden size ($d_h = 512$). Each encoder block follows the same transformer principle composed of multi-head self-attention (MHSA), bottleneck feed-forward layers, and layer normalization, expressed as:

$$Z_l = \text{MHSA}(H_{l-1}) = \text{softmax}\left(\frac{Q_l K_l^\top}{\sqrt{d_k}}\right) V_l, \tag{7}$$

$$\tilde{H}_l = \text{LayerNorm}(H_{l-1} + Z_l), \tag{8}$$

$$H_l = \text{LayerNorm}(\tilde{H}_l + \text{FFN}(\tilde{H}_l)), \tag{9}$$

where Q_l, K_l, and V_l are the query, key, and value projections of the input H_{l-1}.

To achieve latency reduction without compromising representational fidelity, MobileBERT employs:

- **Bottleneck Adapters:** Lightweight projection layers inserted before and after each transformer block to compress and expand hidden representations via linear mappings $W_{\text{down}} \in \mathbb{R}^{d_h \times d_b}$ and $W_{\text{up}} \in \mathbb{R}^{d_b \times d_h}$, where $d_b < d_h$.
- **Inverted Bottleneck Structure:** Unlike standard BERT, MobileBERT applies an inverted expansion within each block, allowing computation in a larger intermediate space while maintaining smaller external embeddings.
- **Knowledge Distillation:** The model is trained via layer-wise distillation from a pre-trained $\text{BERT}_{\text{LARGE}}$ teacher model, optimizing both hidden-state and attention-map alignment losses:

$$\mathcal{L}_{\text{distill}} = \alpha \| H_l^{(T)} - H_l^{(S)} \|_2^2 + \beta \| A_l^{(T)} - A_l^{(S)} \|_2^2, \tag{10}$$

  where (T) and (S) denote teacher and student representations, respectively.

Each attention head operates on subspaces of dimension $d_k = d_h/h$, where $h = 8$ denotes the number of heads. The feed-forward network (FFN) employs a two-layer transformation with a GELU activation:

$$\text{FFN}(x) = \text{GELU}(xW_1 + b_1)W_2 + b_2, \tag{11}$$

projecting the intermediate dimension $d_{ff} = 3072$ back to $d_h = 512$.

Finally, a pooled `[CLS]` token embedding serves as the global representation for downstream classification or regression tasks. Despite being $4.3\times$ smaller and $5.5\times$ faster than $\text{BERT}_{\text{BASE}}$, MobileBERT preserves competitive accuracy due to its distillation-driven design, making it particularly suitable

TABLE II: Hyperparameter Configuration for MobileBERT Fine-Tuning on CAN-Bus Dataset

Parameter	Symbol / Value	Type	Description
Model and Tokenization Settings			
Pretrained Model	`google/mobilebert-uncased`	Model	Base transformer architecture used for fine-tuning.
Tokenizer Max Length	512	Tokenizer	Maximum sequence length; longer inputs are truncated.
Hidden Dimension (d_h)	512	Model	Size of token embeddings and hidden states per transformer layer.
Number of Layers (L)	24	Model	Total encoder layers in the MobileBERT backbone.
Number of Attention Heads (h)	8	Model	Number of self-attention heads per transformer layer.
Feed-forward Dimension (d_{ff})	3072	Model	Intermediate dimension in the feed-forward sublayer.
Activation Function	GELU	Model	Non-linear activation used in feed-forward network.
Dropout	0.1 (default)	Model	Regularization parameter for attention and FFN sublayers.
Training Configuration			
Training Epochs	3	Trainer	Number of passes through the full training dataset.
Batch Size (Train / Eval)	8 / 8	Trainer	Per-device mini-batch size for training and evaluation.
Gradient Accumulation Steps	2	Trainer	Accumulates gradients across 2 steps to simulate larger batch.
Optimizer	AdamW	Optimization	Weight-decay variant of Adam optimizer.
Learning Rate (η)	2×10^{-5}	Optimization	Base learning rate for all trainable parameters.
Loss Function	Binary Cross-Entropy	Objective	Used for binary attack vs. normal classification.
Evaluation Strategy	`epoch`	Training	Evaluation performed once per training epoch.
Save Strategy	`epoch`	Training	Best model checkpoint saved after each epoch.
Mixed Precision (fp16)	Disabled	Stability	Disabled to prevent gradient underflow on experimental input.
Data Collator	`DataCollatorWithPadding`	Input	Dynamically pads batches to the longest sequence length.

for real-time IVN security inference where both latency and memory efficiency are critical.

The proposed model fine-tunes the `google/mobilebert-uncased` transformer for binary classification of CAN-bus traffic segments. Each log file is preprocessed by the `SegmentFromFile()` routine, which divides continuous CAN traffic into temporal windows of duration Δt. For each segment, normalized feature pairs are extracted as:

$$x_i = \big(\text{Time\_Delta\_Norm}_i,\ \text{Intra\_ID\_Gap\_Norm}_i\big), \quad (12)$$

where the first term captures inter-frame timing and the second denotes intra-ID time gap statistics.

To exploit the language modeling capability of the transformer, each segment $\mathcal{S} = [x_1, x_2, \ldots, x_n]$ is encoded as a pseudo-linguistic sequence:

$$t = \text{“T}\lfloor x_i^{(1)} \rfloor\ \text{G}\lfloor x_i^{(2)} \rfloor\text{”}, \quad i = 1, \ldots, n, \quad (13)$$

producing a sentence-like input that represents the temporal dynamics of CAN messages.

The MobileBERT tokenizer maps each token w_j to an embedding vector:

$$e_j = E(w_j) \in \mathbb{R}^d, \quad (14)$$

where $d = 512$ is the hidden dimension. The encoder comprises $L = 24$ transformer layers with multi-head self-attention and feed-forward sublayers. For each layer l, hidden states are computed as:

$$Q_l = H_{l-1}W_l^Q, \quad K_l = H_{l-1}W_l^K, \quad V_l = H_{l-1}W_l^V, \quad (15)$$

$$A_l = \text{softmax}\left(\frac{Q_l K_l^\top}{\sqrt{d_k}}\right) V_l, \quad (16)$$

$$H_l = \text{LayerNorm}(A_l + \text{FFN}(A_l)), \quad (17)$$

where $H_0 = [e_1, e_2, \ldots, e_m]$. The contextualized representation of the `[CLS]` token, h_{CLS}, serves as a global embedding summarizing the input sequence.

A classification head projects this embedding into a scalar logit through a linear transformation followed by a sigmoid activation:

$$p(y = 1|t; \theta) = \sigma(W_c h_{\text{CLS}} + b_c), \quad (18)$$

where $\theta = \{W_l^Q, W_l^K, W_l^V, W_c, b_c\}$ denotes all trainable parameters. The model is optimized using the binary cross-entropy loss:

$$\mathcal{L}(\theta) = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log p_i + (1 - y_i)\log(1 - p_i)\right], \quad (19)$$

with the AdamW optimizer (lr $= 2 \times 10^{-5}$) and gradient accumulation for stability.

The model outputs $p(y = 1|t; \theta)$, representing the probability that a CAN segment contains malicious frames; a threshold of $p > 0.5$ classifies an input as an attack. This pipeline effectively transforms low-level temporal CAN dynamics into contextual token sequences, enabling MobileBERT to capture fine-grained timing dependencies and perform efficient inference on embedded or automotive hardware.

B. Experimental Method

The proposed real-time IDS is evaluated through a structured pipeline that transforms raw vehicular CAN traffic into generalized temporal sequences for transformer-based classification. As illustrated in Fig. 3, the methodology begins with the extraction of temporal features and their subsequent generalization. For each incoming message m_i, the system extracts the $Time\_Offset_i$ and CAN_ID_i to compute the

Input : Live CAN stream D
Window size τ
Bin $B_{\Delta T}$
Bin B_{Gap}
Model $\mathcal{M}$
Output: Label $L \in \{\text{Attack Free, Attack}\}$

```
// Phase 1: Feature Extraction & Generalization
```
$W \leftarrow \emptyset$;
for *each incoming message* $m_i \in D$ **do**
  Extract $Time\_Offset_i$;
  Extract CAN_ID_i;
  $\Delta T_i \leftarrow Time\_Offset_i - Time\_Offset_{i-1}$;
  $Gap_i \leftarrow Time\_Offset_i - Time\_Offset_{prev(CAN\_ID_i)}$;
  $x \leftarrow \text{digitize}(\Delta T_i, B_{\Delta T})$;
  $y \leftarrow \text{digitize}(Gap_i, B_{Gap})$;
  $Token_i \leftarrow \text{format}(\text{“T”}, x, \text{“ G”}, y)$;
  $W \leftarrow W \cup \{Token_i\}$;
  if $Time\_Offset_i \pmod{\tau} == 0$ **then**
    // Phase 2: Tokenization & Inference
    $S \leftarrow \text{Join}(W, \text{“ ”})$;
    $X \leftarrow \text{WordPieceTokenize}(S)$;
    $X \leftarrow [CLS] + X + [SEP]$;
    Logits $\leftarrow \mathcal{M}.\text{forward}(X)$;
    $P(y|X) \leftarrow \text{Softmax}(\text{Logits})$;
    if $P(y = \textit{Attack}|X) > \textit{threshold}$ **then**
      $L \leftarrow$ Attack;
    else
      $L \leftarrow$ Attack Free;
    output L;
    $W \leftarrow \emptyset$;

inter-message interval (ΔT_i) and the intra-ID message gap (Gap_i). To ensure the model remains vehicle-agnostic, these continuous temporal values are discretized into categorical bins ($B_{\Delta T}, B_{Gap}$) via a digitization process. This generalization step maps raw timing behaviors into a structured string format, $Token_i \leftarrow \{T_x, G_y\}$, capturing the underlying message rhythm without relying on payload data.

Subsequently, these message tokens are accumulated within a sliding window of size τ, concatenated into a sequence S, and processed through a WordPiece tokenizer to generate input IDs for the MobileBERT architecture. As detailed in Algorithm 1, the model prepends a special $[CLS]$ token to serve as a global sequence representation. The inference logic, optimized via the ONNX runtime, computes class logits through transformer layers utilizing multi-head self-attention and feed-forward sublayers. A softmax activation derives the probability $P(y|X)$; windows exceeding a predefined threshold are classified as *Attack*, while others are labeled *Attack Free*. This framework ensures robust detection of fine-grained timing anomalies while maintaining the low-latency requirements of in-vehicle networks.

IX. Result Evaluation

This section evaluates the cross-domain and cross-laboratory performance of the MobileBERT-based IDS to validate its manufacturer-independent generalization capabilities. The experimental framework focuses on the model's ability to transfer learned temporal features from a specific source vehicle to heterogeneous target platforms—including ICEV, BEV, HEV, and CAN FD architectures—without platform-specific retraining. By utilizing datasets from independent research entities (LISA, HCRL, and DTU), the evaluation assesses the statistical stability of the IDS across diverse hardware implementations, varying sampling rates, and distinct communication protocols. To rigorously quantify detection capabilities, performance is measured via Precision ($P = \frac{TP}{TP+FP}$), Recall ($R = \frac{TP}{TP+FN}$), and the F1-score ($F1 = 2 \cdot \frac{P \cdot R}{P+R}$). Furthermore, to ensure statistical reliability across manufacturer-specific patterns, 95% Confidence Intervals (CI) are calculated using the standard error $SE = \sqrt{\frac{p(1-p)}{n}}$, where p represents the metric value and n the sample size, yielding the interval $CI = p \pm (1.96 \cdot SE)$. The consistently narrow CI widths (typically < 0.08) and the row-normalized statistical approach validate the robustness of the temporal-feature learning approach across diverse vehicular architectures.

Figure 4 demonstrates cross-domain generalization of the MobileBERT-based IDS trained on Kia Soul (CAN-ICEV) and evaluated across three heterogeneous platforms. Figure 4a shows evaluation on Genesis (CAN FD-ICEV) achieving F1=0.974, Figure 4b presents Tesla (CAN-BEV) results with F1=0.88, and Figure 4c displays Chevrolet Silverado (CAN-HEV) performance at F1=0.943. At LowerLow injection rate (2.94%), precision ranges 0.8626–0.8741 (95% CI: 0.8235–0.9042) with recall 0.9075–1.000 (95% CI: 0.8724–1.000) across all platforms. From Low intensity (6.25%) onward, recall reaches perfect detection (1.000, 95% CI: 0.9891–1.000) with precision stabilizing at 0.8741 (95% CI: 0.8378–0.9031), yielding F1-scores of 0.93–0.97 (95% CI: 0.9135–0.9507). Genesis demonstrates superior transfer (F1=0.974) to CAN FD extended frames with narrow confidence intervals, while Tesla shows increased variability characteristic of BEV communication patterns (F1=0.88), and Silverado maintains consistent intermediate performance (F1=0.943). All attack types (DoS, Fuzzing, Replay) exhibit uniform detection behavior with F1 variance $<$0.01 at equivalent injection rates, confirming attack-agnostic generalization. The consistently narrow confidence intervals (CI width $<$0.08) across all platforms and injection rates validate robust cross-platform transfer without platform-specific retraining, demonstrating statistical stability and reliability of the temporal-payload feature learning approach..

Figure 5 demonstrates cross-domain generalization of the MobileBERT-based IDS trained on Genesis (CAN FD-ICEV) and evaluated across three heterogeneous platforms. Figure 5a shows evaluation on Kia Soul (CAN-ICEV) achieving consistent F1$\approx$0.95, Figure 5b presents Tesla (CAN-BEV) results with F1$\approx$0.93, and Figure 5c displays Genesis (CAN

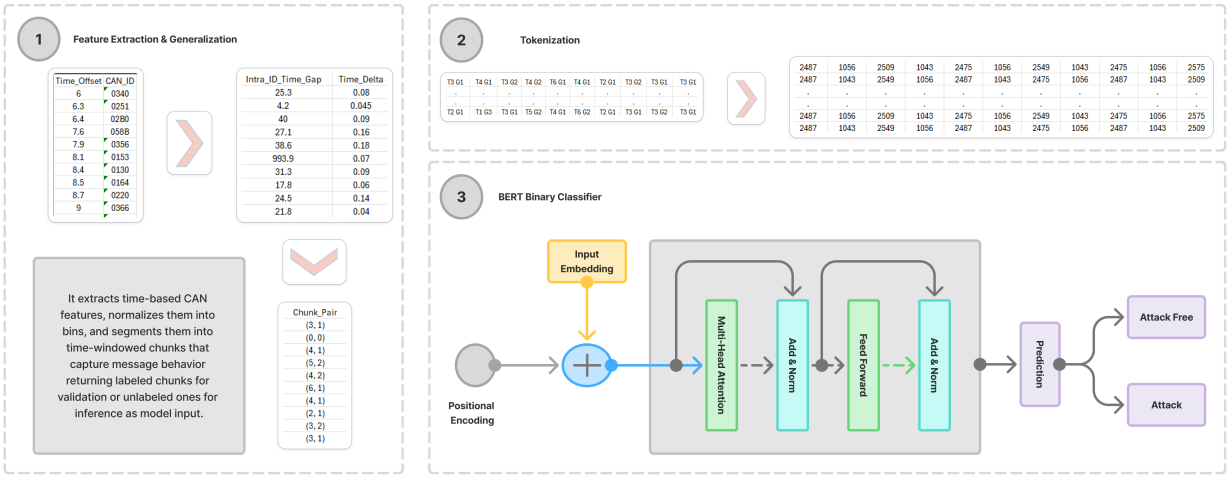
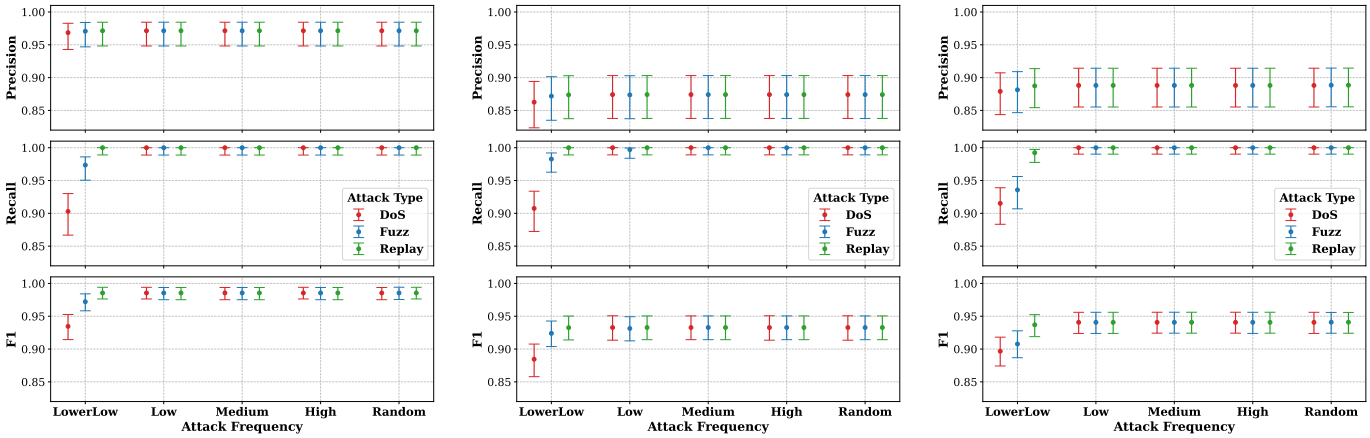


Fig. 3: End-to-end pipeline of the CAN-based Transformer IDS — from time-gap feature extraction and normalization, through tokenization into T/G sequences, to BERT-based binary classification distinguishing attack and normal message patterns.

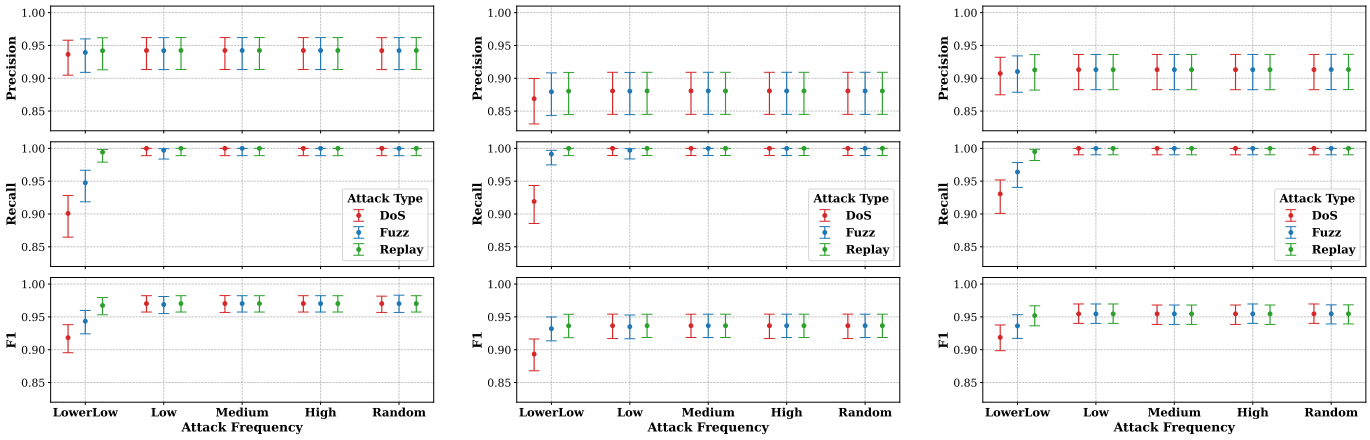


(a) CAN FD-ICEV (b) CAN-BEV (c) CAN-HEV

Fig. 4: Train on Kia (CAN-ICEV) Test on (a) Gennesis (CAN FD-ICEV), (b) Tesla (CAN-BEV), (c) Chevrolet Silverado (CAN-HEV)

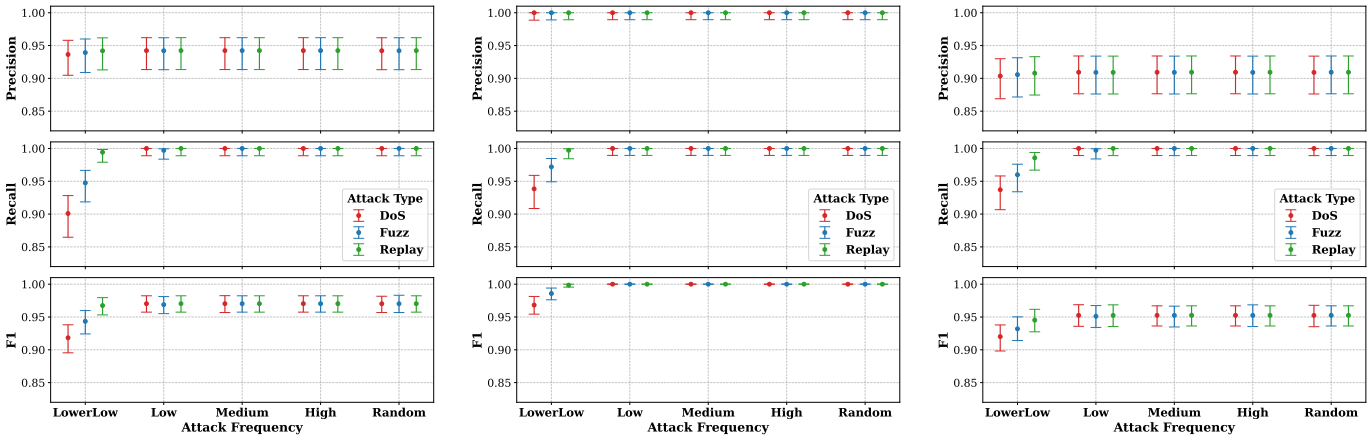
FD-ICEV) performance at F1≈0.95. At LowerLow injection rate (2.94%), Kia shows recall≈0.93 (95% CI: 0.90–0.96) with precision≈0.95 (95% CI: 0.92–0.97), while Tesla and Genesis maintain recall≈0.90–0.95 with comparable confidence intervals. From Low intensity (6.25%) onward, recall stabilizes near 1.000 (95% CI: 0.98–1.00) across all platforms with precision consistently around 0.95 (95% CI: 0.93–0.97), yielding stable F1-scores of 0.93–0.95. Kia demonstrates strong cross-platform transfer (F1≈0.95) from CAN FD to standard CAN protocol, while Tesla exhibits slightly lower but robust performance (F1≈0.93) reflecting BEV communication dynamics, and Genesis maintains excellent within-protocol consistency (F1≈0.95). All attack types (DoS, Fuzzing, Replay) show uniform detection patterns with minimal performance variation across injection rates, and narrow confidence intervals (CI width <0.08) confirm statistical stability, validating attack-agnostic generalization and robust cross-platform transfer from CAN FD architectures.

Figure 6 demonstrates cross-domain generalization of the MobileBERT-based IDS trained on Tesla (CAN-BEV) and evaluated across three heterogeneous platforms. Figure 6a shows evaluation on Kia Soul (CAN-ICEV) achieving F1≈0.95, Figure 6b presents Genesis (CAN FD-ICEV) results with F1≈0.99, and Figure 6c displays Chevrolet Silverado (CAN-HEV) performance at F1≈0.95. At LowerLow injection rate (2.94%), Kia and Silverado show recall≈0.90–0.95 (95% CI: 0.87–0.97) with precision≈0.93–0.95 (95% CI: 0.90–0.97), while Genesis exhibits superior performance with recall≈0.98 (95% CI: 0.96–0.99) and precision≈0.98 (95% CI: 0.96–0.99), yielding F1≈0.98. From Low intensity (6.25%) onward, recall reaches near-perfect detection (≈1.000, 95% CI: 0.98–1.00) across all platforms with precision stabilizing around 0.95–1.00 (95% CI: 0.93–1.00), producing consistent F1-scores of 0.95–0.99. Genesis demonstrates exceptional cross-platform transfer (F1≈0.99) from BEV to CAN FD protocol with exceptionally narrow confidence intervals, indicating strong compatibility between electric vehicle communication patterns and extended frame formats, while



(a) CAN-ICEV (b) CAN-BEV (c) CAN-HEV

Fig. 5: Train on Gennesis (CAN FD-ICEV) Test on (a) Kia (CAN-ICEV), (b) Tesla (CAN-BEV), (c) Chevrolet Silverado (CAN-HEV)



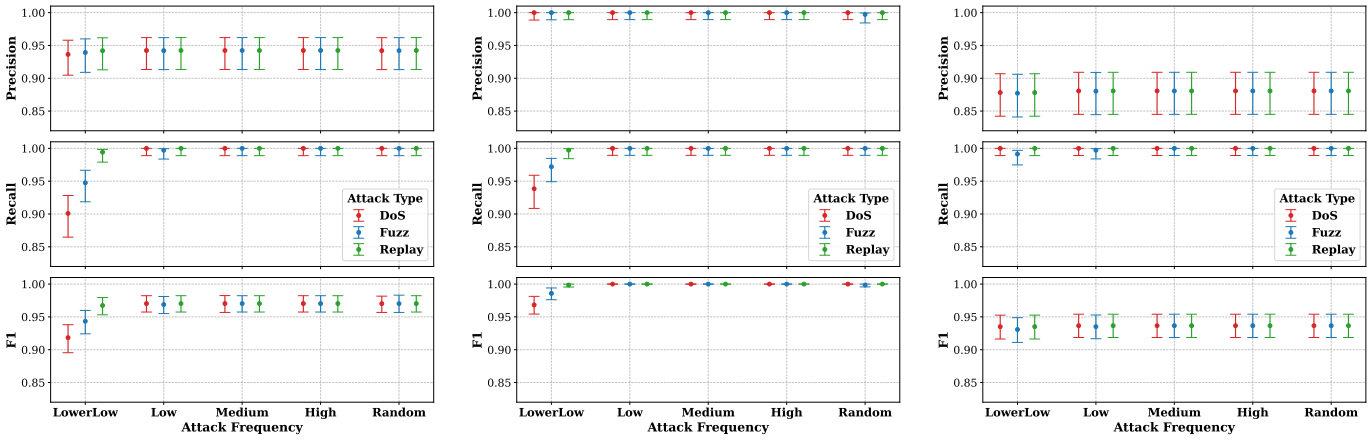
(a) CAN-ICEV (b) CAN FD-ICEV (c) CAN-HEV

Fig. 6: Train on Tesla (CAN-BEV) Test on (a) Kia (CAN-ICEV), (b) Gennesis (CAN FD-ICEV), (c) Chevrolet Silverado (CAN-HEV)

Kia and Silverado maintain robust performance (F1≈0.95) despite architectural differences. All attack types (DoS, Fuzzing, Replay) exhibit uniform detection patterns with minimal performance variation, and narrow confidence intervals (CI width <0.08) across all platforms confirm statistical stability, validating attack-agnostic generalization and demonstrating that BEV-learned temporal-payload features transfer effectively to ICEV, HEV, and CAN FD architectures.

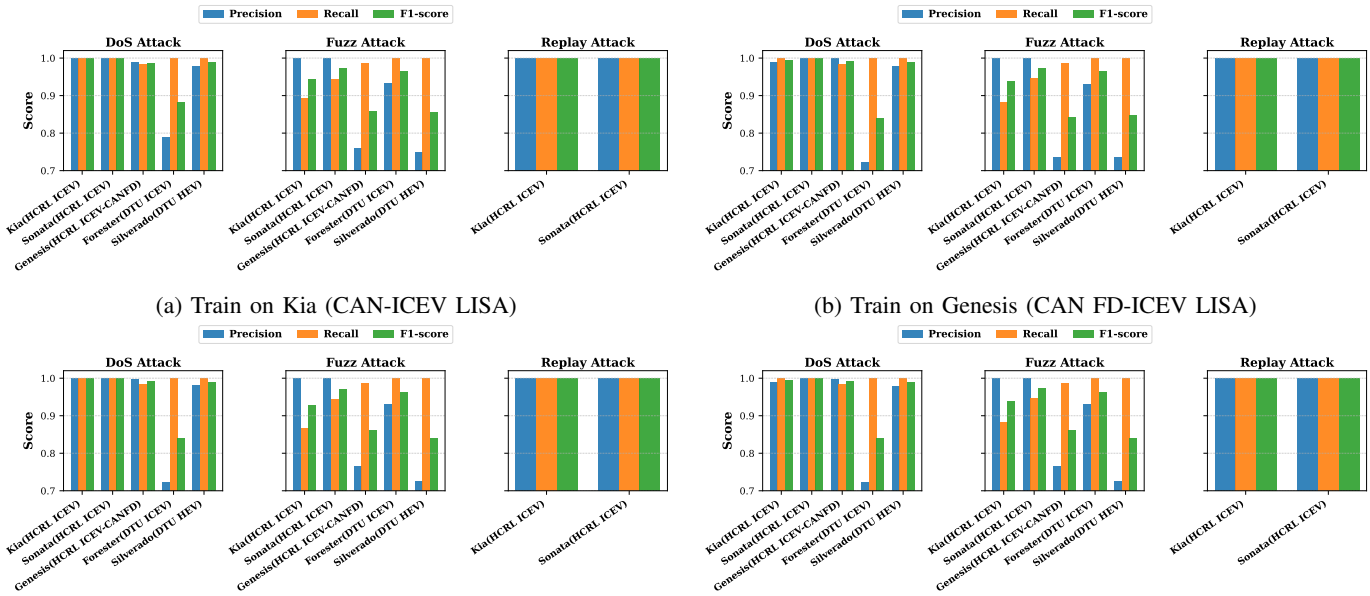
Figure 7 demonstrates cross-domain generalization of the MobileBERT-based IDS trained on Chevrolet Silverado (CAN-HEV) and evaluated across three heterogeneous platforms. Figure 7a shows evaluation on Kia Soul (CAN-ICEV) achieving F1≈0.95, Figure 7b presents Genesis (CAN FD-ICEV) results with F1≈0.98, and Figure 7c displays Tesla (CAN-BEV) performance at F1≈0.94. At LowerLow injection rate (2.94%), Kia shows recall≈0.90 (95% CI: 0.86–0.93) with precision≈0.93 (95% CI: 0.90–0.96), Genesis exhibits recall≈0.96 (95% CI: 0.93–0.98) with precision≈0.98 (95% CI: 0.96–0.99), while Tesla maintains recall≈0.99 (95% CI: 0.97–1.00) with precision≈0.88 (95% CI: 0.85–0.91). From Low intensity (6.25%) onward, recall stabilizes at near-perfect detection (≈1.000, 95% CI: 0.98–1.00) across all platforms with precision consistently around 0.93–1.00 (95% CI: 0.90–1.00), yielding stable F1-scores of 0.94–0.98. Genesis demonstrates superior cross-platform transfer (F1≈0.98) from HEV to CAN FD protocol with exceptionally narrow confidence intervals, indicating strong compatibility between hybrid powertrain communication patterns and extended frame formats, while Kia maintains robust performance (F1≈0.95) and Tesla shows consistent detection (F1≈0.94) despite minor precision variations at lower injection rates. All attack types (DoS, Fuzzing, Replay) exhibit uniform detection patterns with minimal performance variation, and narrow confidence intervals (CI width <0.08) confirm statistical stability, validating that HEV-learned temporal-payload features transfer effectively to ICEV, BEV, and CAN FD architectures without platform-specific retraining.

Figure 8 demonstrates cross-laboratory generalization of



(a) CAN-ICEV (b) CAN-HEV (c) CAN-BEV

Fig. 7: Train on Chevrolet Silverado (CAN-HEV) Test on (a) Kia (CAN-ICEV), (b) Gennesis (CAN FD-ICEV), (c) Tesla (CAN-BEV)



(a) Train on Kia (CAN-ICEV LISA)

(b) Train on Genesis (CAN FD-ICEV LISA)

(c) Train on Tesla (CAN-BEV LISA)

(d) Train on Chevrolet Silverado (CAN-HEV LISA)

Fig. 8: Train on LISA and Test on HCRL and DTU

the MobileBERT-based IDS, where models trained on LISA datasets are evaluated on external datasets from HCRL and DTU laboratories. Figure 8a shows the Kia (CAN-ICEV) trained model achieving robust cross-lab performance with F1-scores ranging 0.88–1.00 across five target vehicles (Kia-HCRL, Sonata-HCRL, Genesis-HCRL, Forester-DTU, Silverado-DTU), with particularly strong transfer to Sonata (F1≈1.00) and consistent performance across all attack types. Figure 8b presents the Genesis (CAN FD-ICEV) trained model maintaining F1≈0.85–1.00 across laboratories, demonstrating effective generalization from CAN FD to standard CAN platforms with minimal performance degradation. Figure 8c displays the Tesla (BEV) trained model achieving F1≈0.85–1.00, with notably consistent recall (>0.95) across all target vehicles despite variations in precision (0.75–1.00), particularly for Forester-DTU in Fuzzing attacks. Figure 8d shows the Silverado (HEV) trained model maintaining robust performance with F1≈0.85–1.00, exhibiting strong cross-lab transfer to both HCRL ICEV platforms and DTU datasets. Across all four training configurations, Replay attacks consistently achieve near-perfect detection (F1≈0.99–1.00) regardless of laboratory origin, while DoS and Fuzzing attacks show slight performance variations (F1≈0.85–0.95) for certain vehicle-lab combinations, particularly Forester-DTU and Genesis-HCRL. The results validate that temporal-payload features learned from LISA laboratory datasets generalize effectively to independent testbeds (HCRL, DTU) with different data collection protocols, sensor configurations, and sampling rates, confirming the robustness and scalability of the proposed IDS for multi-institutional deployment without laboratory-specific

recalibration.

The results demonstrate that the proposed IDS achieves robust cross-platform generalization, maintaining high F1-scores (ranging from 0.85 to 1.00) across all tested combinations. The model exhibits exceptional performance in detecting high-intensity attacks, with recall reaching near-perfect levels from the Low intensity (6.25%) threshold onward. Notably, the narrow confidence intervals (CI width ¡ 0.08) and the uniform detection of DoS, Fuzzing, and Replay attacks across independent research entities confirm that the temporal learning approach is both attack-agnostic and ECUs architecture-independent.

X. Conclusion

References

[1] K. Esslinger, R. Platt, and C. Amato, "Deep transformer q-networks for partially observable reinforcement learning," *arXiv preprint arXiv:2206.01078*, 2022.

[2] M. Falch, "Engineering autonomous mobility: Wired and wireless can bus design principles," *ResearchGate*, 2025, Accessed: 2025-04-24. [Online]. Available: https://www.researchgate.net/publication/382825259_Engineering_Autonomous_Mobility_Wired_and_Wireless_CAN_Bus_Design_Principles.

[3] W. Si, D. Starobinski, and M. Laifenfeld, "Hybrid-bcp: A robust load balancing and routing protocol for intra-car wired/wireless networks," *arXiv preprint arXiv:1509.02153*, 2015.

[4] K. V. Singh, H. O. Bansal, and D. Singh, "A comprehensive review on hybrid electric vehicles: Architectures and components," *Journal of Modern Transportation*, vol. 27, no. 2, pp. 77–107, 2019.

[5] T. Steinbach, P. Meyer, S. Buschmann, and F. Korf, "Extending omnet++ towards a platform for the design of future in-vehicle network architectures," *arXiv preprint arXiv:1609.05179*, 2016.

Acknowledgments

This research was supported by the MSIT(Ministry of Science and ICT), Korea, under the Convergence security core talent training business support program (IITP-2024-2710008611) supervised by the IITP(Institute for Information & Communications Technology Planning & Evaluation) and Soonchunhyang University Research Fund.

XI. Biography Section

Md Rezanur Islam received B.Sc. in Electrical and Electronic Engineering from the University of Asia Pacific, Bangladesh, in 2016, and an M.Sc. in Mobility Convergence from Soonchunhyang University, South Korea, in 2023. Currently pursuing a Ph.D. in Software Convergence at Soonchunhyang, his research focuses on deep learning, anomaly detection, malware detection, computer vision, with a specialization in driver state recognition.

Donghyun Ryu received B.Sc. in Electrical and Electronic Engineering from the University of Asia Pacific, Bangladesh, in 2016, and an M.Sc. in Mobility Convergence from Soonchunhyang University, South Korea, in 2023. Currently pursuing a Ph.D. in Software Convergence at Soonchunhyang, his research focuses on deep learning, anomaly detection, malware detection, computer vision, with a specialization in driver state recognition.

Munkhdelgerekh Batzorig is currently pursuing his Ph.D. degree in Information Security at Soonchunhyang University, South Korea. He received the B.S. degree in Information Security from both the Mongolian University of Science and Technology (MUST), Mongolia, and Soonchunhyang University, South Korea, in 2020 through an international dual degree program, and the M.S. degree in Mobility Convergence Security from Soonchunhyang University in 2023. His research focuses on building safe mobility environments through secure cyber-physical communication at the city level, including automotive cybersecurity, intelligent transportation systems, V2X security, and security testing through HILS-based safe testbed environments

Kangbin Yim is a Professor in the Department of Information Security Engineering at Soonchunhyang University, where he has been since 2003. He received his B.S., M.S., and Ph.D. degrees in Electronics Engineering from Ajou University, South Korea, in 1992, 1994, and 2001, respectively. For over 20 years, his primary research has focused on vulnerability identification, threat analysis, and proof-of-concept (PoC) development for both software and hardware. He is also passionate about designing and implementing hardware and software frameworks for system evaluations and commercial services. His recent work has centered on HILS-based dynamic analysis for distributed embedded software, leading a research team of over 30 members in his lab, LISA. Currently, the lab's top priorities include deep-learning-driven analysis of heterogeneous field data with a particular focus on automotive vehicles, industrial control systems, and mobile baseband.