Efficient Intrusion Detection for In-Vehicle Networks Using Knowledge Distillation From BERT to CNN-BiLSTM

Department of Software, Sookmyung Women's University
Jisoo Kim

Table of Contents

01 Introduction

1. Background issues

2. Proposed framework

3. Experimental Results & Contributions

***Abstract*—Under the development of intelligent transportation systems, In-Vehicle Networks (IVNs) serve as a critical channel for both internal and external communications. However, the inherent complexity and diversity of data traffic present significant challenges for the detection of IVN anomalous flows. Meanwhile, the introduction of various novel technologies has introduced new security vulnerabilities to IVNs. These vulnerabilities significantly impact the security of IVNs and the accuracy of in-vehicle Intrusion Detection Systems (IDS). To address these issues, this paper proposes a lightweight and efficient anomaly detection method based on knowledge distillation technology, termed Knowledge Distillation from BERT to CNN-BiLSTM (KDBC). Specifically, the KDBC distills the deep semantic knowledge from the BERT model into a more lightweight CNN-BiLSTM architecture, significantly reducing computational overhead and storage requirements without substantially compromising detection performance. Experimental results demonstrate that the KDBC model enhances both security and versatility, achieving superior detection accuracy in identifying abnormal attacks across diverse IVN data, including automotive Ethernet and CAN networks. Moreover, the KDBC model has been validated for its effectiveness and robustness in actual in-vehicle gateway environments, achieving an accuracy of over 0.98 and an F1 score greater than 0.98.**

***Index Terms*—Ethernet, CAN, knowledge distillation, BERT, CNN-BiLSTM.**

01 Introduction

<Problem statements>

- Increasing heterogeneity and connectivity introduce new attack surfaces (e.g., DoS, replay, frame injection, etc.)
  - Existing IDSs are limited to single protocols or rely on heavy models (e.g., BERT) that cannot be deployed in real vehicles.

However, the architecture of modern IVNs has evolved significantly, incorporating a heterogeneous mix of CAN bus, automotive Ethernet, and TSN technologies to meet the growing demands for high bandwidth and low-latency communication. This multi-protocol integration introduces new and complex security challenges. Each protocol comes with distinct data formats, transmission patterns, and potential attack surfaces [13]. Consequently, IDSs that are designed to operate within a single-protocol context, such as CAN-only or Ethernet-only solutions, are no longer sufficient. They are incapable of handling the cross-domain traffic that modern centralized gateways must monitor and protect. Therefore, research efforts limited to single or dual network types fall short in addressing the comprehensive security demands of today's in-vehicle communication systems.

<Main Contributions>

Achieve high detection accuracy with lightweight efficiency by distilling knowledge from BERT to a CNN-BiLSTM model (KDBC)!

- The first IDS proposal that covers the three areas of CAN, Automotive Ethernet, and TSN in a single framework

- To the best of our knowledge, the KDBC model is the first to integrate the detection of abnormal traffic across CAN networks, automotive Ethernet, and TSN into a single framework.

02 Related Works

TABLE I

COMPARISON OF RELATED WORK

Related Work	Type of IDS	Datasets	Method	Scenarios	Real Scenario Test
[11]	CNN-LSTM-based IDS	CAN Signal Extraction and Translation Dataset	CNN-LSTM with attention	CAN bus	Real vehicle test
[12]	Hybrid deep learning-based IDS	Car Hacking Dataset (CHD)	CNN, LSTM	CAN bus	No
[15]	Bert-based IDS	CHD, can-train-and-test	BERT	CAN bus	Real vehicle test
[16]	Bert-based IDS	CHD	BERT	CAN bus	No
[17]	Open set recognition-based IDS	CHD	Cluster	CAN bus	No
[18]	Gaussian naive bayes-based IDS	OTIDS	GNN	CAN bus	FPGA test
[1]	Federated GNN-based IDS	CAN Signal Extraction and Translation Dataset	GNN	CAN bus	No
[6]	Deep CNN-based IDS	TOW-IDS	CNN Wavelet Transform	Ethernet TSN	Real vehicle test
[19]	Deep learning-based IDS	TOW-IDS, AEID	RF, CNN	Ethernet TSN	No
[20]	Unsupervised-IDS	TOW-IDS	Autoencoder	Ethernet TSN	Nvidia Jetson AGX Xavier test
[21]	Open source-based IDS	Private dataset	Rule-based	TSN	TSN testbed
[22]	Anomaly-based IDS	Private dataset	Ingress control	TSN	Simulated
Proposed KDBC	Knowledge distillation-based IDS	TOW-IDS can-train-and-test private dataset	BERT CNN-BiLSTM	CAN bus Ethernet TSN	Real in-vehicle gateway test

03 System Model

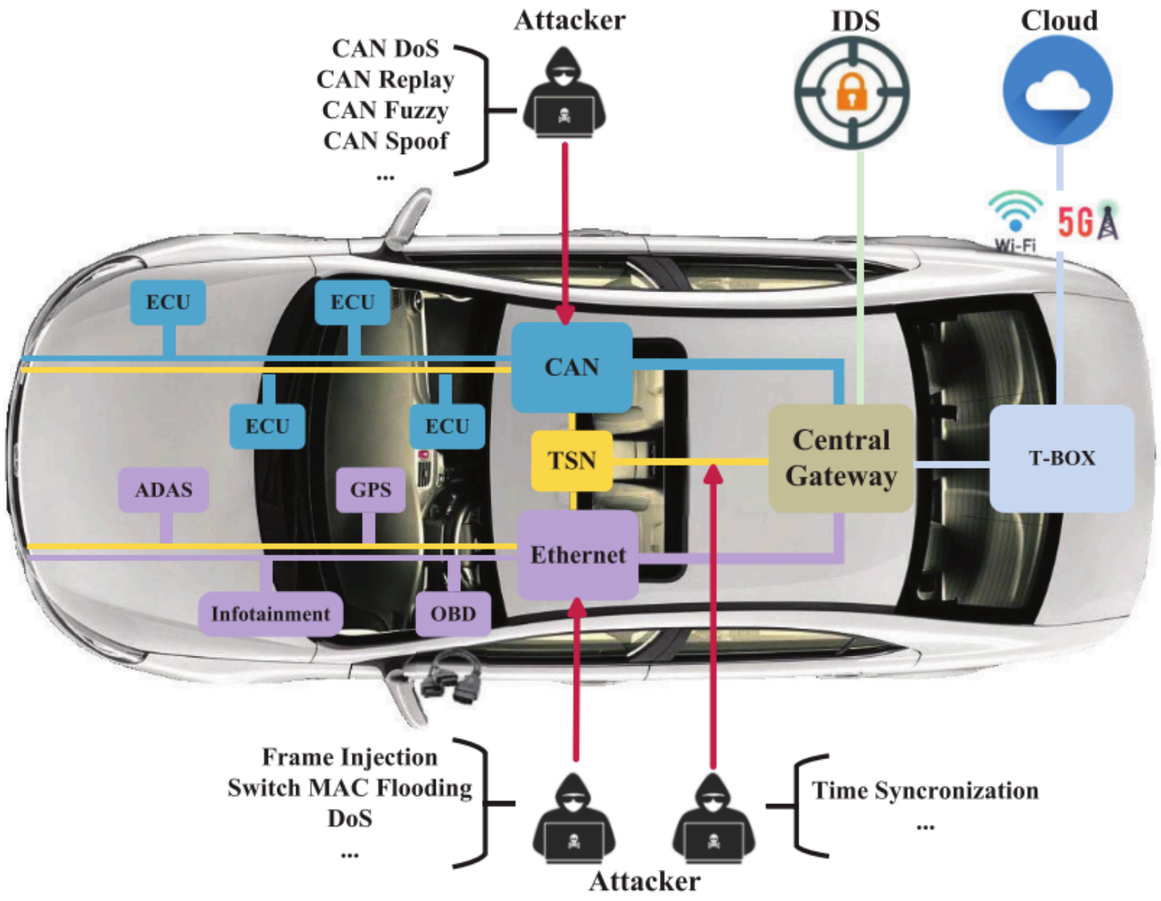


Fig. 1. The scenario of IVN.

04 Proposed KDBC framework

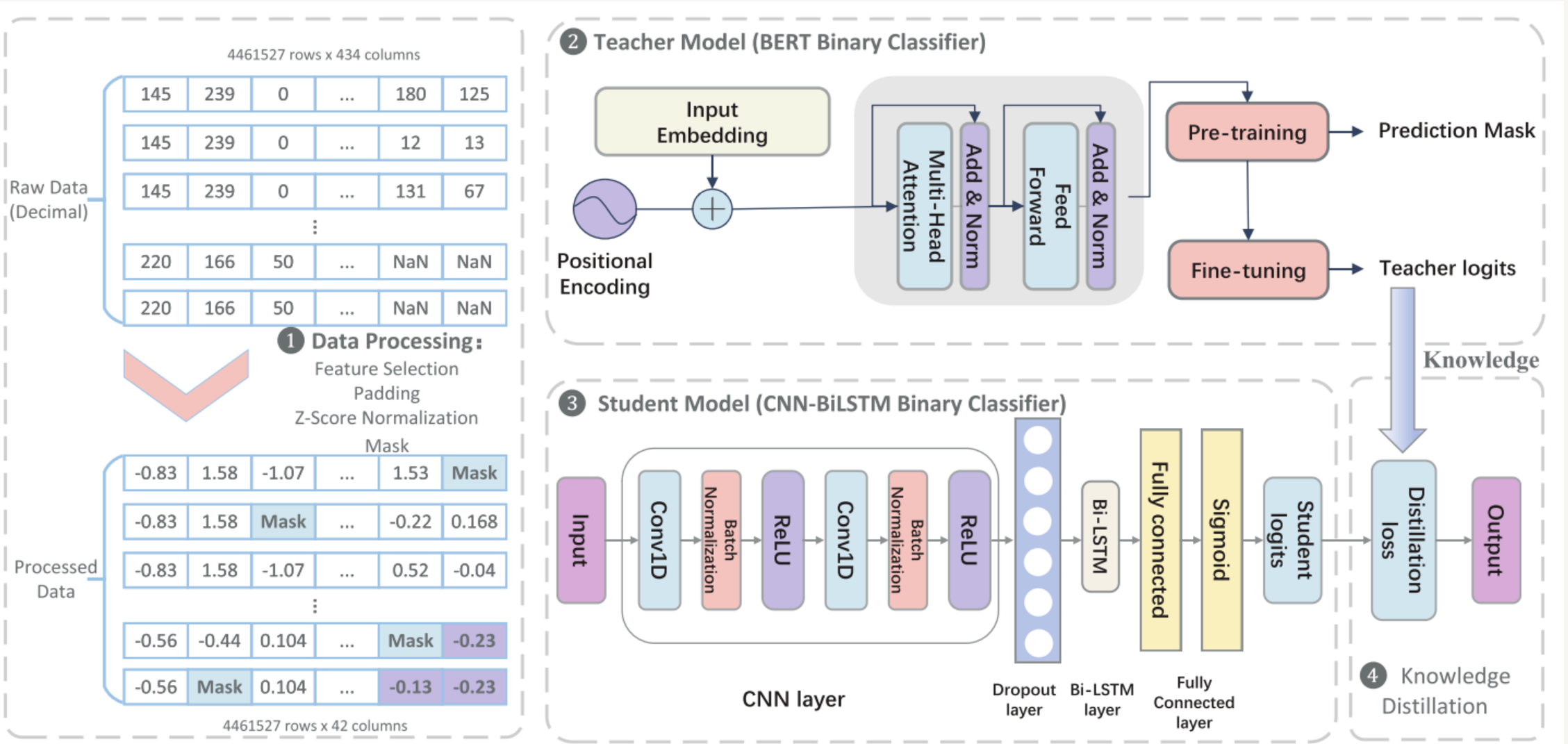


Fig. 2. The framework of KDBC.

04 KDBC - 1) Data Processing

1. **Mutual information-based** feature selection

$$I(A;B) = \sum_{a \in A} \sum_{b \in B} d(a,b) \log \left(\frac{d(a,b)}{d(a)d(b)} \right)$$

d(a, b) : the joint probability distribution of A and B
d(a), d(b) : the marginal probability distributions of variable A and **target variable** B, respectively.

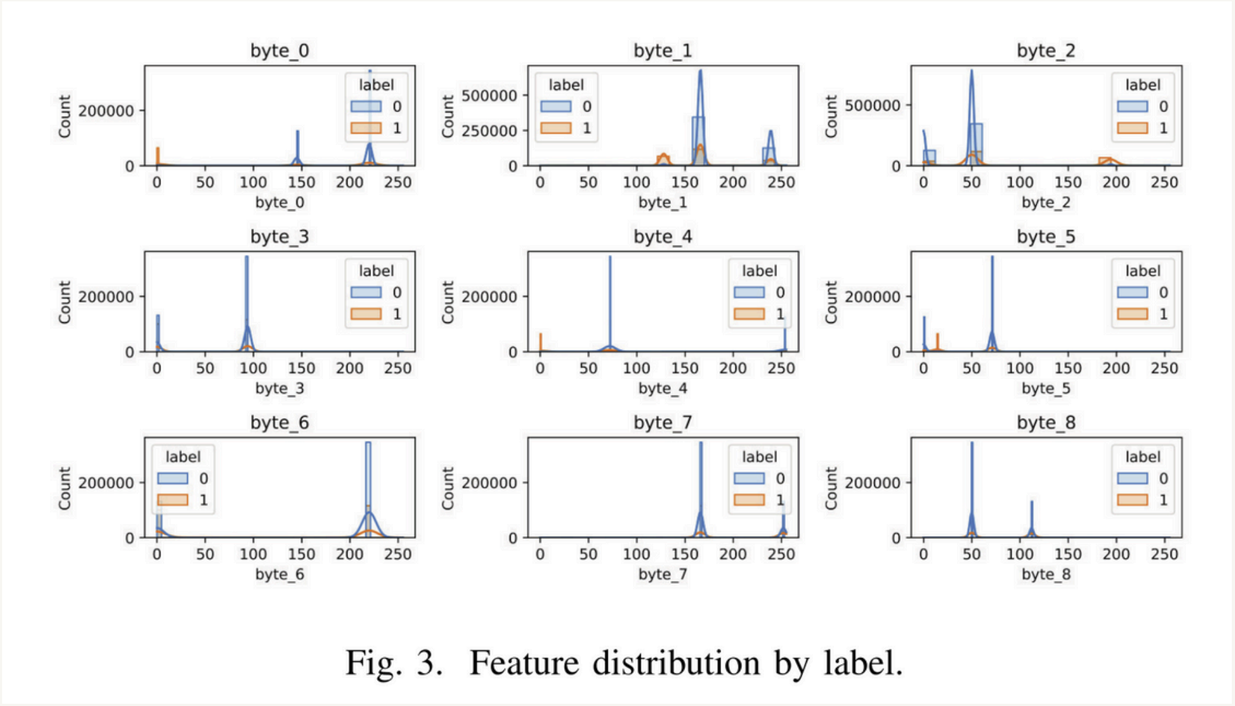


Fig. 3. Feature distribution by label.

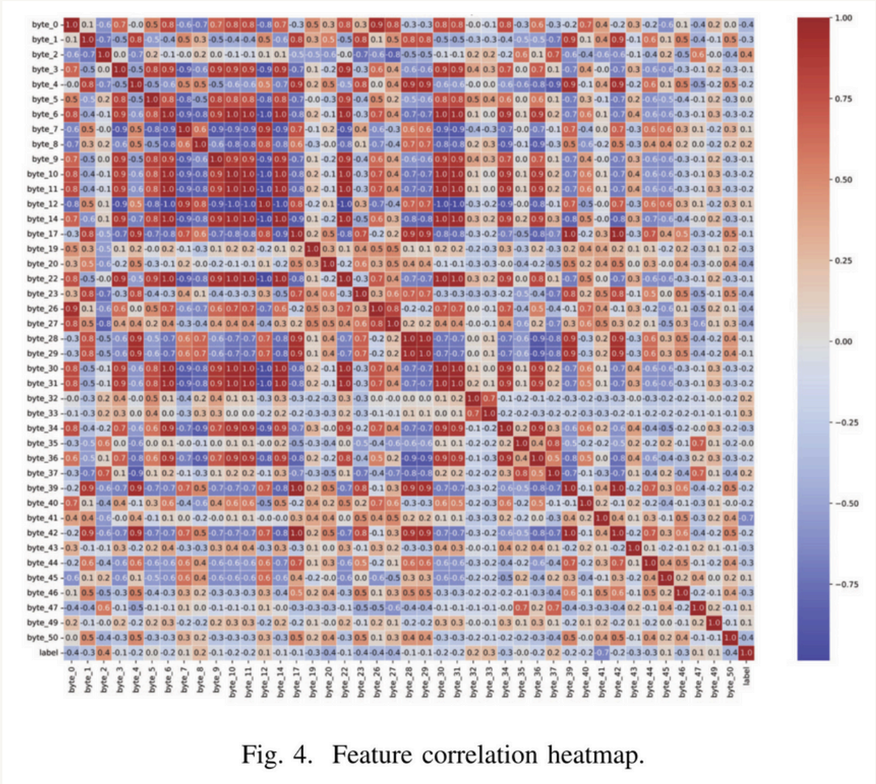
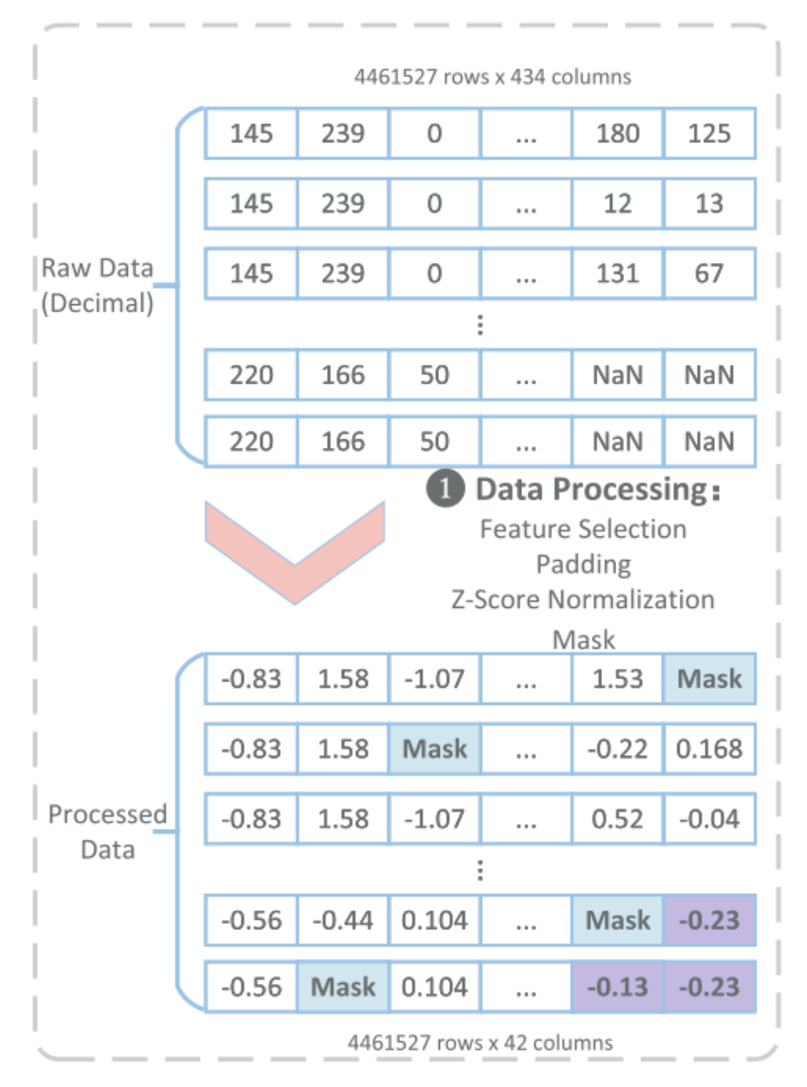


Fig. 4. Feature correlation heatmap.

04 KDBC - 1) Data Processing



2. Zero Padding

After the feature selection process, the data length from both automotive Ethernet and TSN networks was unified to 42 bytes. Therefore, the data length from the CAN bus needs to be extended to 42 bytes using zero padding, while retaining the original 16-byte content. Zero padding is instrumental in enabling intrusion detection models to process data from multiple IVN protocols by standardizing input length, thereby ensuring compatibility across heterogeneous data formats such as CAN, Ethernet, and TSN. This alignment facilitates the ability of model to recognize cross-protocol anomalies without structural inconsistencies, enhancing the versatility and robustness of the KDBC across diverse network traffic.

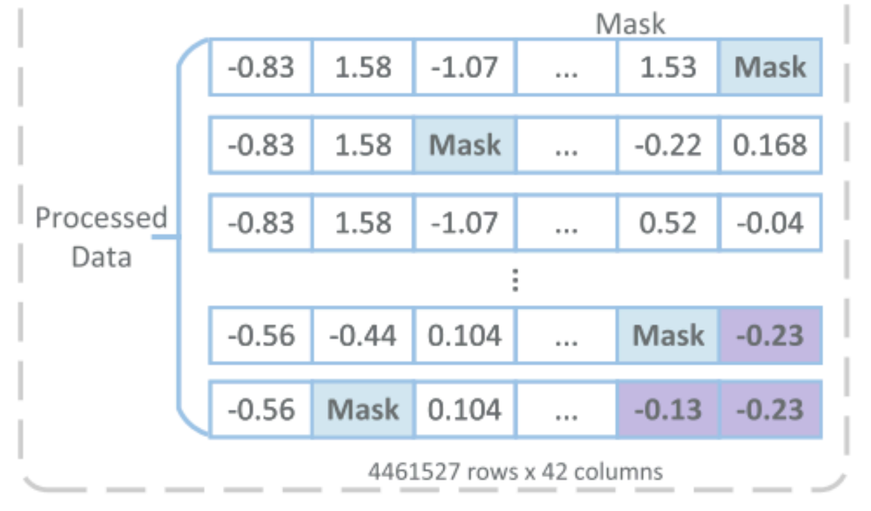
3. Z-score Normalization

$$Score = \frac{S - \mu}{SD}$$

S : individual observation,
i.e., **a data point** in the original dataset
μ : the **mean** of the entire dataset
SD : the **standard deviation** of the dataset

04 KDBC - 1) Data Processing

4. **Masked Language Data Augmentation**



- improves the capacity of model to recognize unknown attacks & mitigates overfitting to extent
- It trains the network traffic to be treated as “sentences” and produce correct output even with partially occluded input.

Algorithm 1 Masked Language Data Augmentation Algorithm

Input: *text_corpus* (a collection of text documents)
Output: *modified_corpus* (text documents with tokens masked or replaced)

Initialize *modified_corpus* as an empty list
for each text in *text_corpus* **do**
  Generate a random number r between 0 and 1
  if $r < 0.1$ **then**
    Append text to *modified_corpus*
    Generate another random number p between 0 and 1
    if $p < 0.7$ **then**
      Randomly select a token from the text
      Replace the selected token with [mask]
    else if $p < 0.9$ **then**
      Randomly select a token from the text
      Replace the selected token with another random token from the vocabulary
    else
      Keep the text unchanged
    end if
  else
    Append the unchanged text to *modified_corpus*
  end if
end for
return: *modified_corpus*

[mask] → Resilient learning to address omissions, sensor errors, and packet loss

Random permutation → Noise and disturbance (attack/interference) resistance

Maintain → Preserve the original distribution (completely prevents distortion)

04 KDBC - 2) Teacher Model

3 key reasons of choosing the BERT model as a teacher model among numerous deep learning models?

reasons. First, unlike traditional feature engineering-based intrusion detection methods, BERT can automatically extract useful features from network traffic data without the need for complex feature extraction rules [34]. It can significantly enhance the efficiency and accuracy of intrusion detection. Second, the pre-training of BERT on a large corpus endows it with strong cross-domain adaptability. In the context of IVN traffic intrusion detection, BERT can quickly adapt to and handle various types of network traffic data, including encrypted traffic [35]. Finally, the BERT model supports fine-tuning, allowing for further training and optimization tailored to specific tasks. In the realm of IVN traffic intrusion detection, fine-tuning the BERT model can improve detection accuracy and efficiency by adapting to different network environments.

The BERT model is a pre-trained deep neural network that centers around the bidirectional encoder of the Transformer architecture. This model generates high-quality language representations through unsupervised pre-training on large-scale corpora, followed by fine-tuning for specific tasks.

04 KDBC - 2) Teacher Model

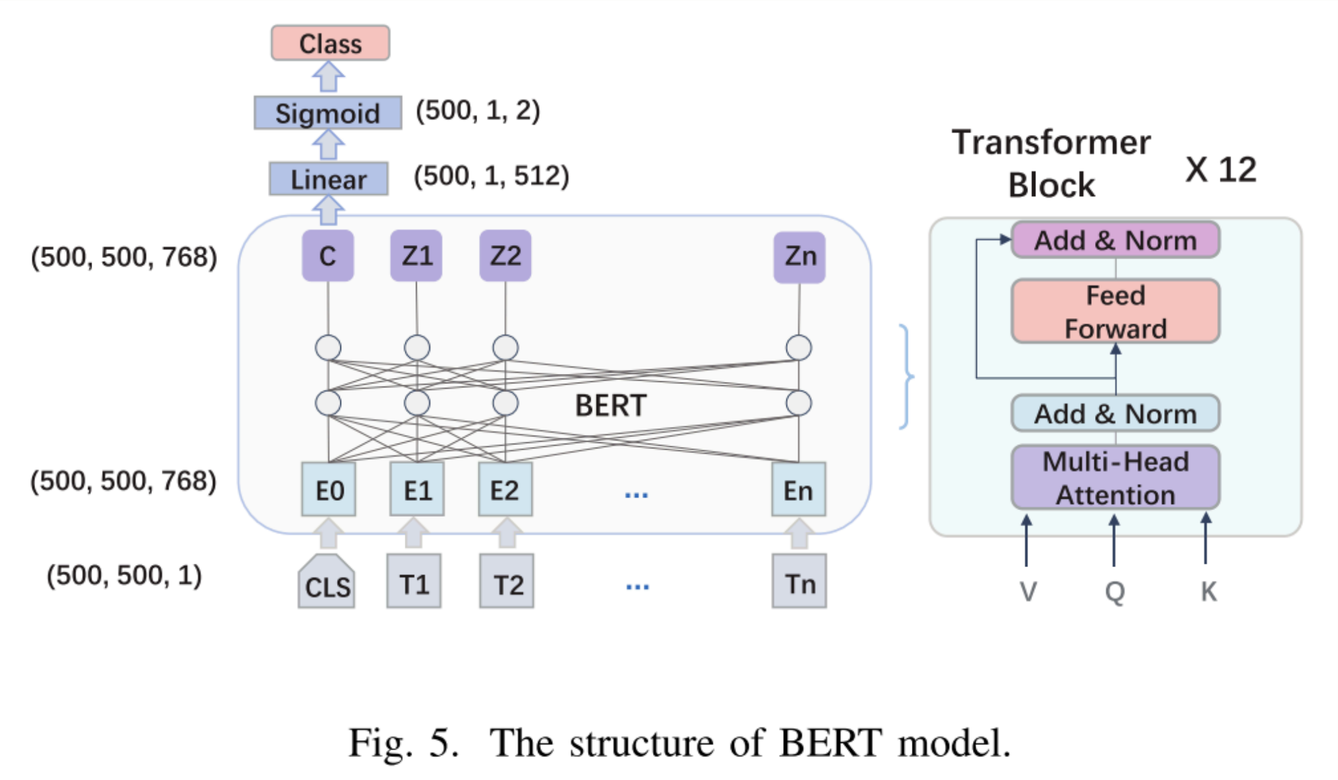


Fig. 5. The structure of BERT model.

1. Input Embedding

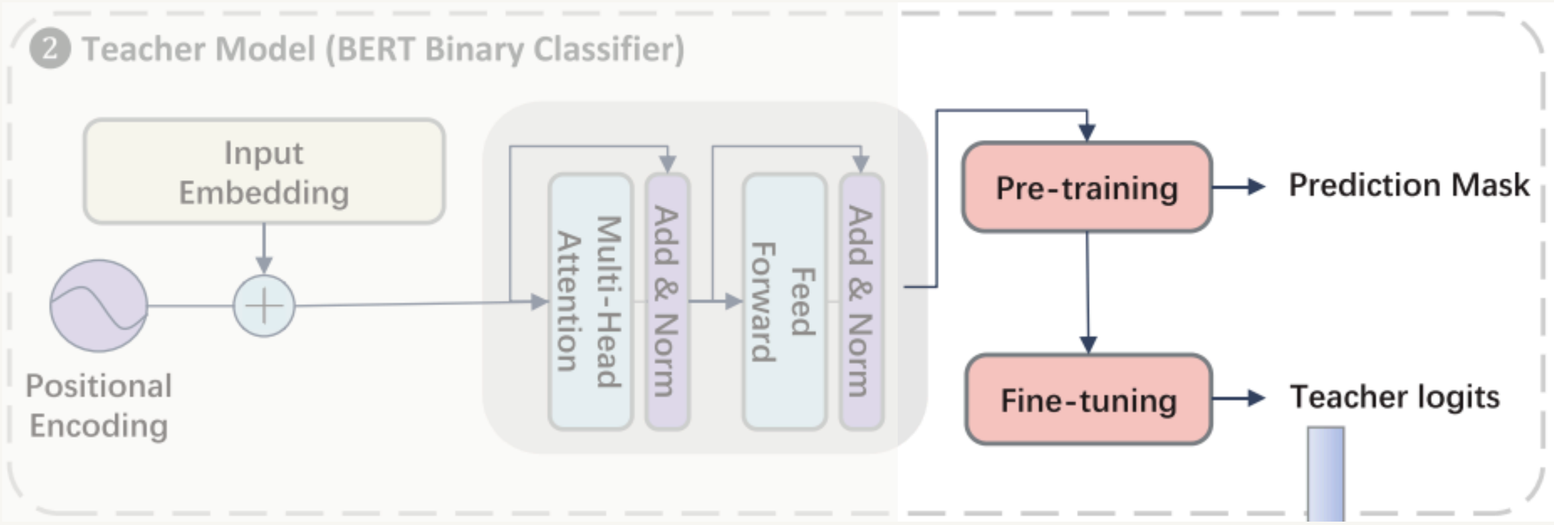
- Token embedding
  - The 'content' of the token as a vector
- Segmentation embedding
  - Which bundle/stream does this token belong to?
- Position embedding
  - Embed the vector to contain the order/position information of the tokens

2. Transformer Encoder

- Each Transformer layer primarily comprises a **multi-head self-attention mechanism** and a **Feedforward Neural Network (FFNN)**

04 KDBC - 2) Teacher Model

3. Output Layer



3-1. **Pretraining Phase**

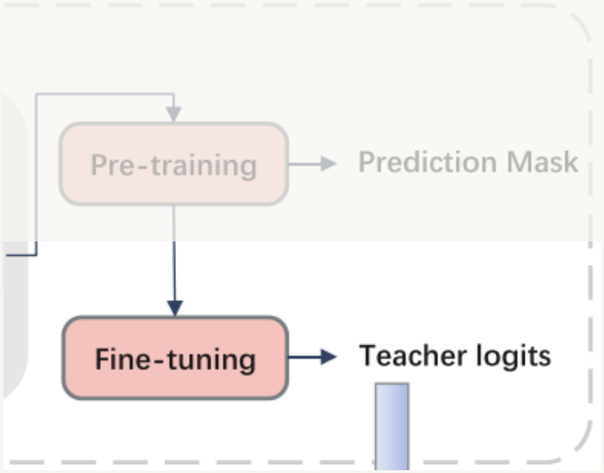
- Loss for MLM(Masked Language Model)

$$\mathcal{L}_{MLM} = -\sum_{t \in M} \log P\left(x_t \mid X_{\setminus M}\right)$$

M : the set of masked positions
X\M : the input with masked tokens
x_t : the true value of the masked word at index t

04 KDBC - 2) Teacher Model

3-2. **Fine-tuning stage**



- **Adding Task-Specific Layers** (for the binary classification)
  - Modify the activation function of FC layer in the final output stage to **sigmoid**
- **Parameter Setting**
  - Initialize using **the parameters of the pre-trained model**
  - => accelerates the training process & enhances model performance

$$L = \frac{1}{N}\sum_i L_i$$
$$= \frac{1}{N}\sum_i -\left[y_i \cdot \log(p_i) + (1-y_i) \cdot \log(1-p_i)\right]$$

y_i : the label for sample i (positive : 1 / negative : 0)

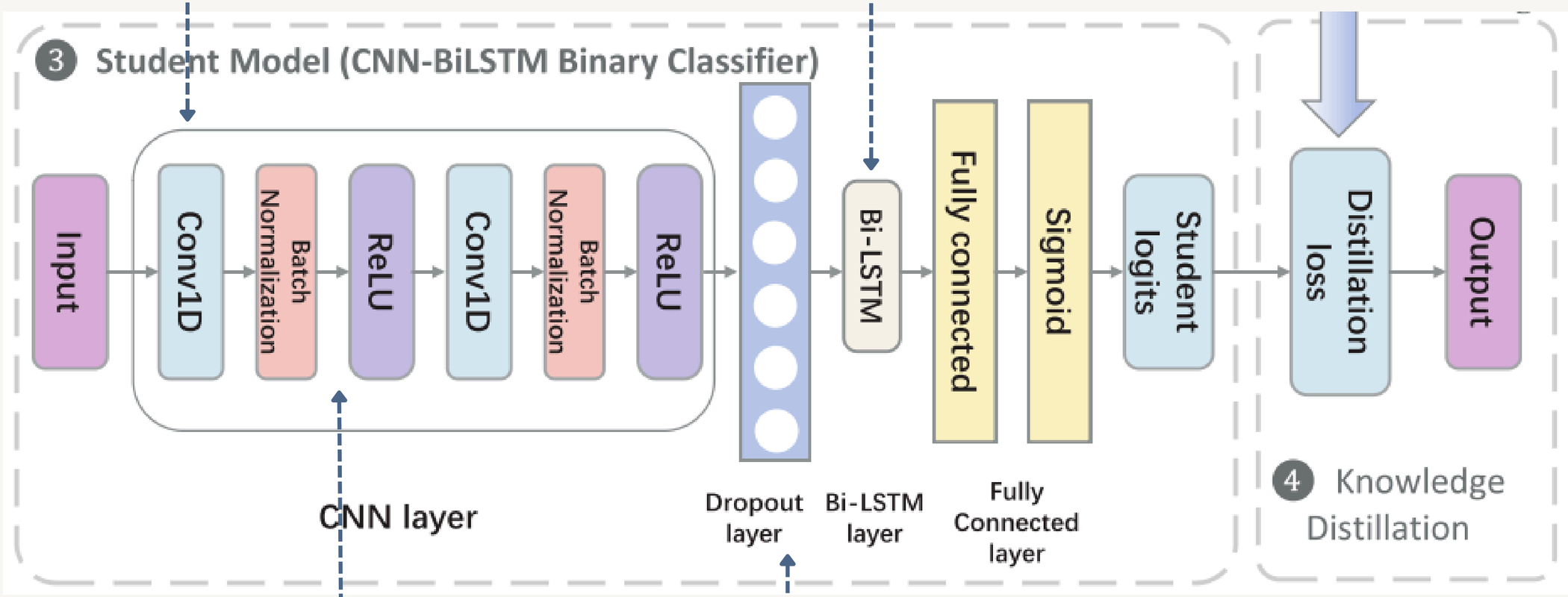
- **Fine-Tuning Training**
  - During the model parameters being fine-tuned, supervised learning is conducted using labeled training data, then the the loss is calculated using the **cross-entropy loss function.**

Now the Teacher is ready to transfer its optimal logits to the student.

04 KDBC - 3) Student Model

Local feature extraction from network traffic

Capture long-term dependencies within the data



Enhance non-linearity and stability

Mitigate risk of Overfitting

04 KDBC - 4) Knowledge Distillation

- **Response-based** knowledge distillation
  - Employ **response-based knowledge** to compute the final loss
  - Idea : Enable student model to **imitate** the final prediction results of the teacher model, by **minimizing the difference(=loss) in responses between them.**

Algorithm 2 The Process of Knowledge Distillation

Input: *train_dataloader*, *teacher_model*, *student_model*
Output: *epoch_loss*

epoch_loss ← 0
$\alpha \leftarrow 0.5$
for epoch in epochs **do**
  total_loss ← 0
  for step, *x_train*, *y_train* in enumerate(*train_dataloader*) **do**
    $teacher\_logits = teacher\_model(x\_train, y\_train)$
    $student\_output, student\_logits = student\_model(x\_train, y\_train)$
    $p^t = sigmoid\left(\frac{teacher\_logits}{temp}\right)$
    $q^t = sigmoid\left(\frac{student\_logits}{temp}\right)$
    $soft\_loss = temp^2 \times KLDivLoss(p^t, q^t)$
    $hard\_loss = BinaryCrossEntropy(student\_output, y\_train)$
    $KDBC\_loss = \alpha \times hard\_loss + (1-\alpha) \times soft\_loss$
    $KDBC\_loss$.backward()
    Adam($student\_model$.parameters(), $lr = 2 \times 10^{-4}$, $weight\_decay = 10^{-5}$).step()
    $total\_loss = total\_loss + KDBC\_loss$
  end for
  $epoch\_loss \leftarrow \frac{total\_loss}{len(train\_dataloader)}$
end for
return: *epoch_loss*

04 KDBC - 4) Knowledge Distillation

KLDivLoss function

$$D_{KL}(P\|Q) = \sum_{o} P(o) \log \frac{P(o)}{Q(o)}$$

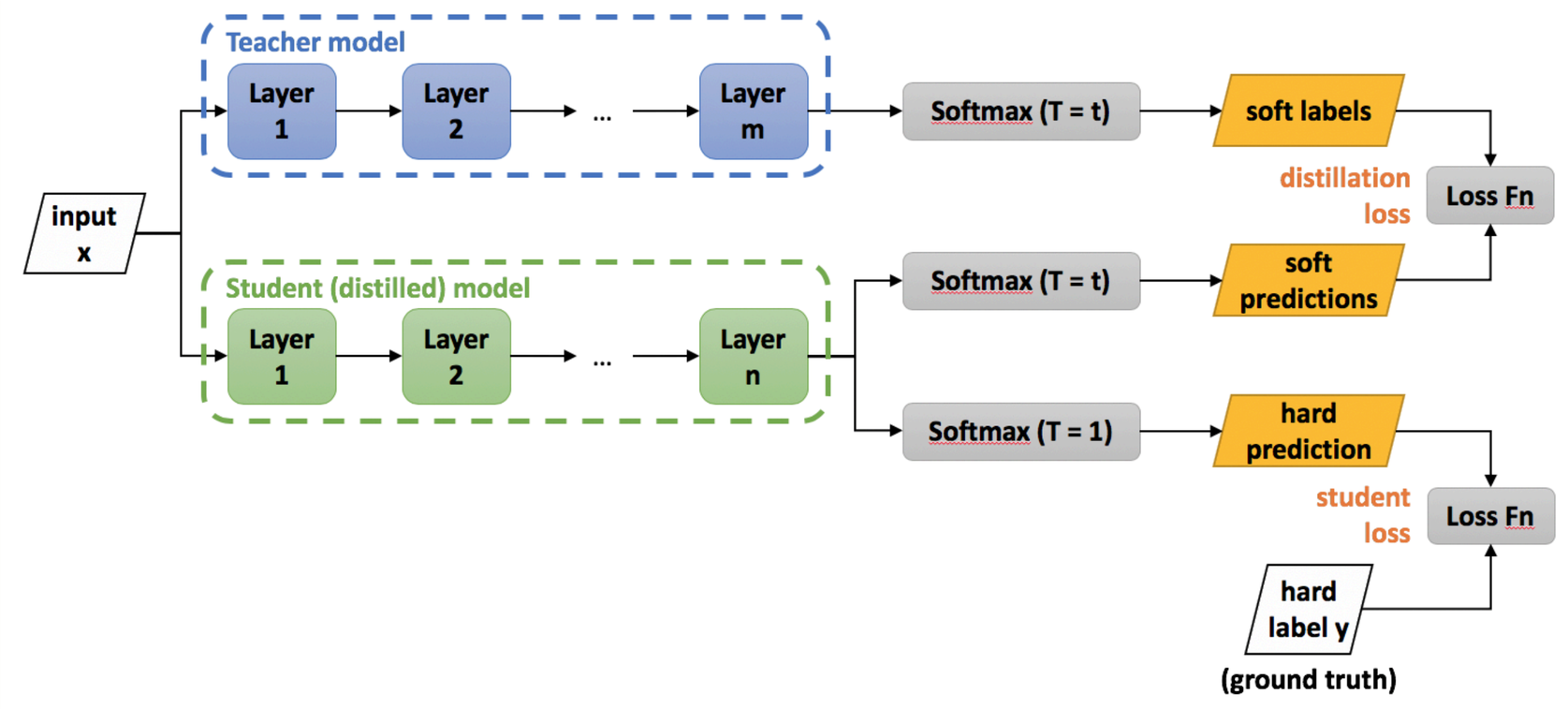
- **D_KL(P||Q) :** KL Divergence from distribution P to distribution Q
  - : non-negative value used to measure the extent to which distribution Q deviates from distribution P
- **P(o)** : the probability of the **o**th event in the **target or true distribution** under discrete conditions.
- **Q(o)** : the probability of the **o**th event in the **approximate or predicted distribution** under discrete conditions.

Algorithm 2 The Process of Knowledge Distillation

Input: *train_dataloader*, *teacher_model*, *student_model*
Output: *epoch_loss*

epoch_loss ← 0
$\alpha \leftarrow 0.5$
for epoch in epochs **do**
  total_loss ← 0
  for step, *x_train*, *y_train* in enumerate(*train_dataloader*) **do**
    teacher_logits = *teacher_model*(*x_train*, *y_train*)
    student_output, *student_logits* = *student_model*(*x_train*, *y_train*)
    $p^t = sigmoid\left(\frac{teacher\_logits}{temp}\right)$
    $q^t = sigmoid\left(\frac{student\_logits}{temp}\right)$
    $soft\_loss = temp^2 \times KLDivLoss(p^t, q^t)$
    hard_loss = *BinaryCrossEntropy*(*student_output*, *y_train*)
    $KDBC\_loss = \alpha \times hard\_loss + (1-\alpha) \times soft\_loss$
    KDBC_loss.backward()
    Adam(*student_model*.parameters(), $lr = 2 \times 10^{-4}$, $weight\_decay = 10^{-5}$).step()
    total_loss = *total_loss* + *KDBC_loss*
  end for
  $epoch\_loss \leftarrow \frac{total\_loss}{len(train\_dataloader)}$
end for
return: *epoch_loss*

04 KDBC - 4) Knowledge Distillation



$$L = \sum_{(x,y)\in\mathbb{D}} L_{KD}(S(x,\theta_S,\tau),T(x,\theta_T,\tau))+\lambda L_{CE}(\hat{y}_S,y)$$

https://intellabs.github.io/distiller/knowledge_distillation.html

05 Experimental Evaluation

A) Datasets

- Employ two diverse & complementary datasets and real vehicle test for training and evaluation
  - CAN bus → **can-train-and-test** (subdataset 3 selected.)
  - automotive Ethernet → **TOW-IDS**
  - TSN → ? (also TOW-IDS. why?)
    - since TOW-IDS contains data for the gPTP protocol + currently no publicly available data on TSN attacks

B) Evaluation Metrics

$$Accuracy = \frac{TRP + TRN}{TRP + TRN + FAP + FAN}$$

$$Precision = \frac{TRP}{TRP + FAP}$$

$$Recall = \frac{TRP}{TRP + FAN}$$

$$F1 - Score = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$$

05 Experimental Evaluation

C) Experimental Results

1) **TOW-IDS**

BERT - accuracy : 0.9980 / F1-Score : 0.9971

KDBC - accuracy : 0.9966 / F1-Score : 0.9935

CNN-BiLSTM - accuracy : 0.9600 / F1-Score : 0.9401

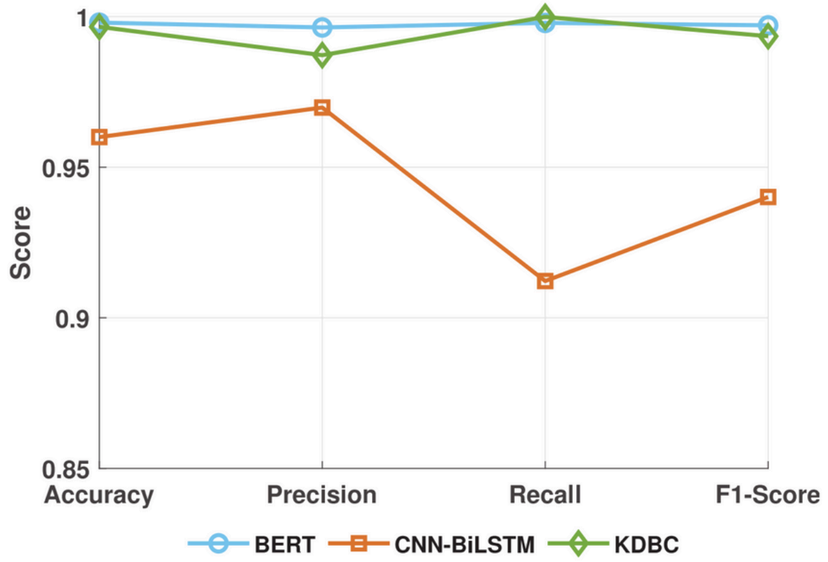
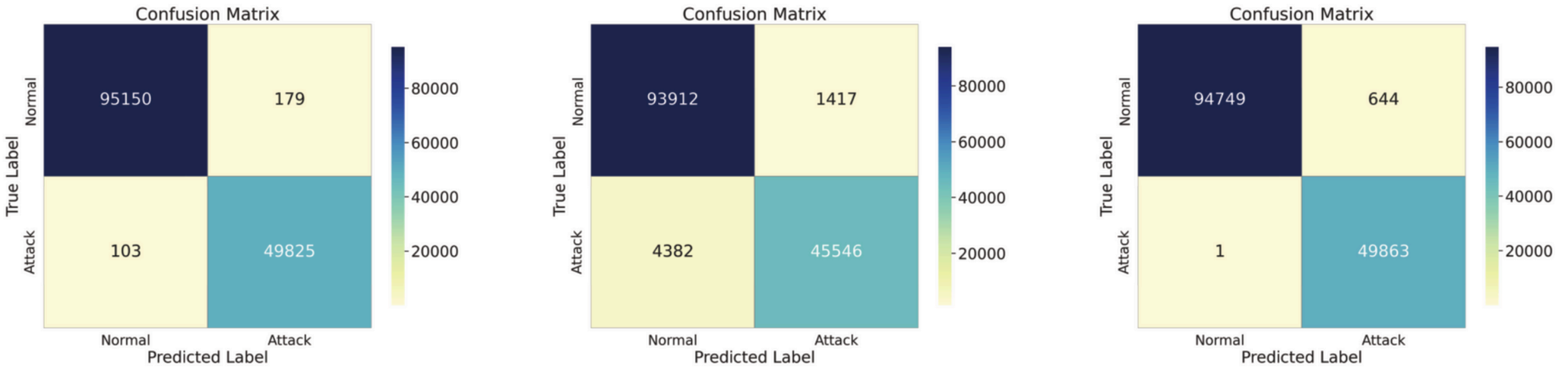


Fig. 8. The comparison of three models on TOW-IDS.



(a) BERT (b) CNN-BiLSTM (c) KDBC

Fig. 7. The confusion matrix of three models on TOW-IDS.

05 Experimental Evaluation

C) Experimental Results

2) **TOW-IDS + Can-Train-and-Test**

BERT - accuracy : 0.9999 / F1-Score : 0.9999

KDBC - accuracy : 0.9996 / F1-Score : 0.9997

CNN-BiLSTM - accuracy : 0.9658 / F1-Score : 0.9696

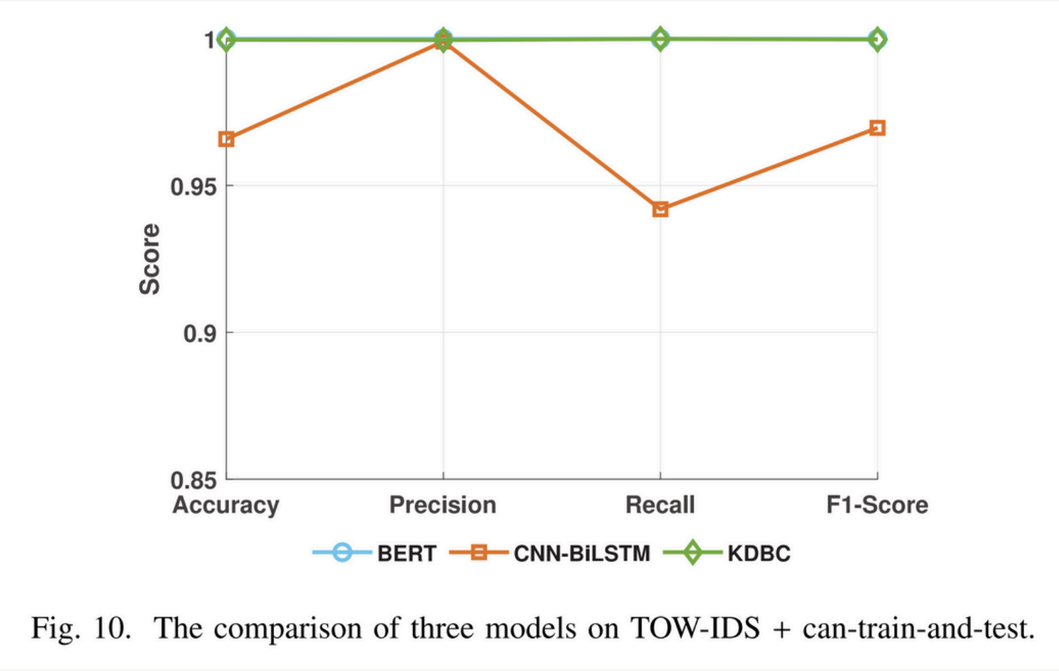
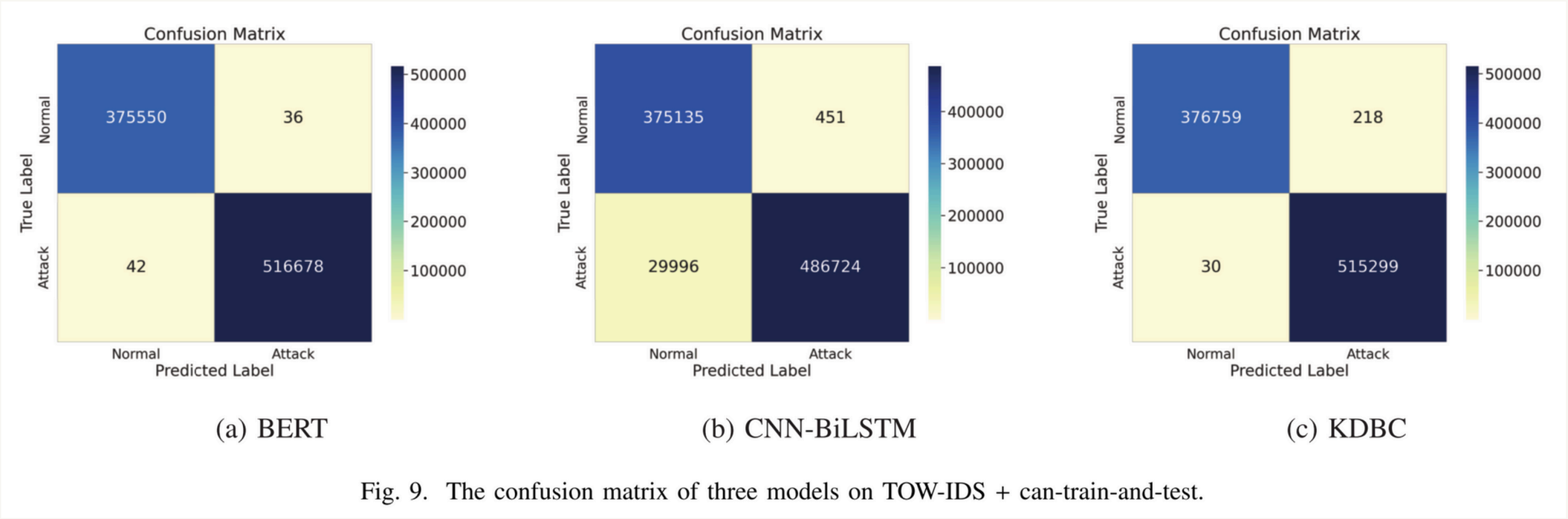


Fig. 10. The comparison of three models on TOW-IDS + can-train-and-test.



(a) BERT (b) CNN-BiLSTM (c) KDBC

Fig. 9. The confusion matrix of three models on TOW-IDS + can-train-and-test.

05 Experimental Evaluation

C) Experimental Results

3) The Cost of KDBC

TABLE II

THE COST OF THREE MODELS ON TOW-IDS + CAN-TRAIN-AND-TEST

Methods	Parameters	Model size (MB)
BERT	109483009	417.64
CNN-BiLSTM	614913	2.35
KDBC	**264833**	**1.01**

05 Experimental Evaluation

C) Experimental Results

4) The comparison with existing research Methods

(1) utilized the TOW-IDS or can-train-and-test datasets in their original form.
(2) adopted deep learning-based architectures, such as CNN-LSTM and BERT variants, which are directly comparable to KDBC.

TABLE III
THE COMPARISON OF KDBC AND OTHER LITERATURE UNDER TOW-IDS DATASET

Methods	Accuracy	F1-Score	Model size (MB)
TOW-IDS (with Wavelet transform) [6]	0.9965	**0.9974**	11.3
ResNet50 [6]	0.9651	0.9785	270.3
EfficientNetB0 [6]	0.9603	0.9776	46.8
Multi-stage IDS [19]	0.9962	0.9960	-
Semi-supervised model [38]	0.9700	0.9500	-
KDBC	**0.9966**	0.9935	**1.01**

TABLE IV
THE COMPARISON OF KDBC AND OTHER LITERATURE UNDER CAN-TRAIN-AND-TEST DATASET

Methods	Accuracy	F1-Score	Model size (KB)
Gaussian Naive Bayes [14]	0.8839	0.9361	0.16
Logistic Regression [14]	0.9565	0.9539	**0.044**
Support Vector Machine [14]	0.9976	0.0018	0.86
Multi-Layer Perceptron [14]	0.9666	0.9594	4.7
KDBC	**0.9986**	**0.9978**	17.7

05 Experimental Evaluation

C) Experimental Results

4) The comparison with existing research Methods

TABLE V

THE COMPARISON OF KDBC AND OTHER LITERATURE UNDER TOW-IDS + CAN-TRAIN-AND-TEST DATASET

Methods	Accuracy	F1-Score	Model size (MB)
ECF-IDS [15]	0.9997	0.9999	108.3
CC-IDPS [39]	0.9998	0.9999	108.1
CNN-LSTM [11]	0.9458	0.9123	1.86
CNN-LSTM (with attention) [11]	0.9679	0.9542	2.42
KDBC	0.9996	0.9997	**1.01**

05 Experimental Evaluation

D) Real scenarios test

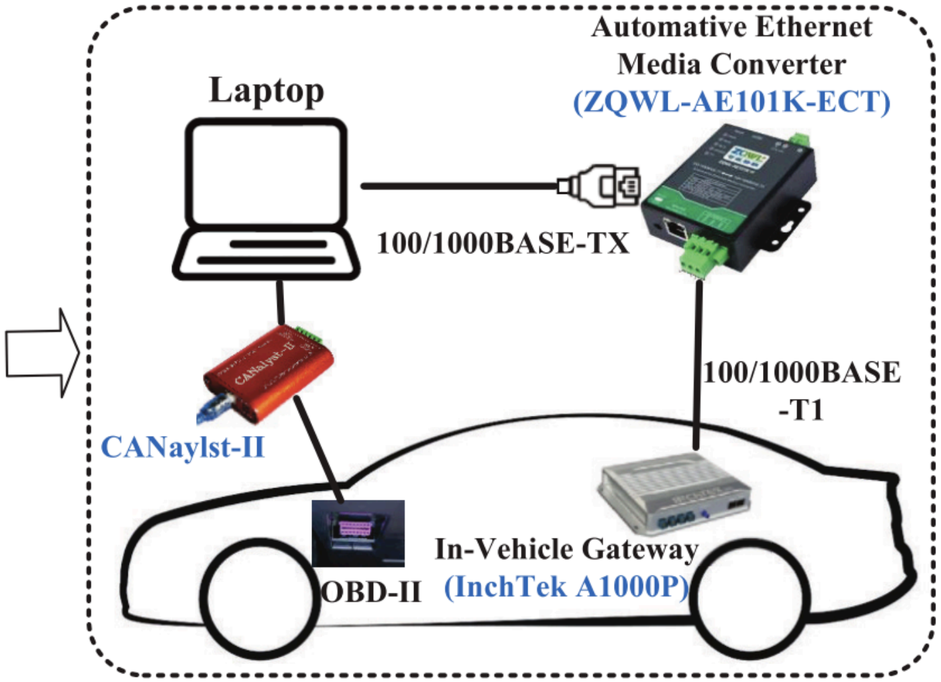


Fig. 11. Test of real scenarios and data.

05 Experimental Evaluation

D) Real scenarios test

TABLE VI

THE COMPOSITION OF IN-VEHICLE GATEWAY TEST DATASET

Type	Amount
Normal	405071
Ethernet DoS	60274
Ethernet Frame Injection	41057
Ethernet Switch MAC flooding	31319
TSN PTP Synchronization	26022
CAN DoS	31000
CAN RPM Spoof	17800
CAN Gear Spoof	19500
CAN Replay	13600
CAN Fuzzy	12600
Sniffing	17577
Scanning	18978

novel network attacks!!

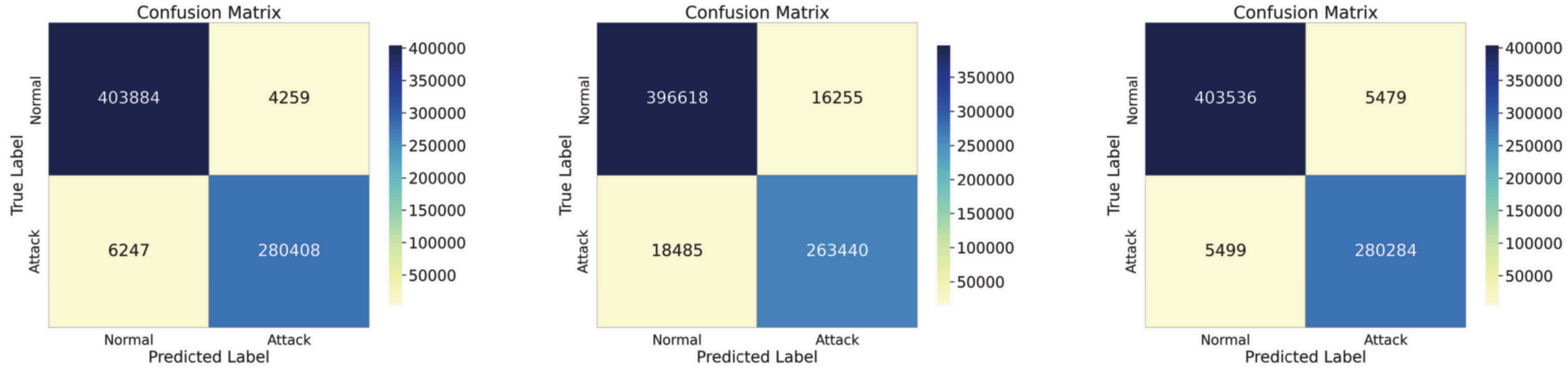
TABLE VII

THE COMPARISON OF KDBC AND OTHER LITERATURE UNDER REAL SCENARIOS

Methods	**Accuracy**	**F1-Score**	**Model size (MB)**
ECF-IDS [15]	0.9849(↓ 0.0148)	0.9825(↓ 0.0174)	108.3
CC-IDPS [39]	0.9852(↓ 0.0146)	0.9827(↓ 0.0172)	108.1
CNN-LSTM [11]	0.9018(↓ 0.0440)	0.8732(↓ 0.0391)	1.86
CNN-LSTM (with attention) [11]	0.9283(↓ 0.0396)	0.9176(↓ 0.0366)	2.42
KDBC	0.9842(↓ 0.0154)	0.9816(↓ 0.0181)	**1.01**

05 Experimental Evaluation

D) Real scenarios test



(a) BERT (b) CNN-BiLSTM (c) KDBC

Fig. 12. The confusion matrix of three models on real vehicle test.

06 Strengths & Limitations

1. **Unified pipeline for heterogeneous IVNs**
   - MI-based 42 features, zero-padding to 42 bytes, Z-score, masking all in one step
   - → CAN/Ethernet/TSN can be trained with a single input format
2. KD achieves **near-teacher accuracy** while being **ultra-light**
   - Model size: BERT 417.64 MB/1e8 params ↔ KDBC 1.01 MB/264,833 params (highly suitable for vehicle deployment)
3. **Data efficiency:** 0.97 accuracy with only 30% of the training data
4. **Real-vehicle gateway test** with diverse attacks (CAN & Ethernet incl. PTP). Real-world F1 ≈ 0.982
5. The KD procedure is clearly documented (easily reproducible)
   - The algorithm/hyper-specific approach is response-based KD (teacher/student logits → temp-sigmoid → KL + BCE, α=0.5, Adam 2e-4 / wd 1e-5)
6. Introduced masking Knowledge Distillation strategy in IVN field, which was mainly used in NLP fields (e.g., DistilBERT)
7. Since BiLSTM is built into the student model, it can learn packet sequences without implementing a sliding window

06 Strengths & Limitations

1. **TSN data scarcity**
   - TOW-IDS (gPTP) was used as a **TSN proxy** due to the lack of publicly available TSN attack data
   - richer actual TSN-specific datasets may be needed
2. Preprocessing risks
   - **Zero-padding** to unify length into 42 bytes may blur protocol semantics and dilute structural differences
   - **Masking augmentation** rule is **NLP-generic** → may require further IVN-domain optimization (e.g., reflecting mask position, field sensitivity, and temporal structure)
3. **Generalization decline** to unseen attacks (sniffing/scanning) in real scenarios
4. **Absence of latency/throughput metrics** for on-gateway inference
   - The deployment structure is explained, but real-time metrics such as inference latency, CPU usage, and power consumption are not provided in numerical form (it's important for practical decision-making)
5. KD format primarily limited to **"response-based" approaches**
   - Intermediate representation distillation (e.g., features, attention maps, contrasts) is not addressed
   - → may need further enrichment of the teacher's structural knowledge of representations...?
6. Description absence of Hyperparameter sweeping for fine-tuning

07 Possible Extensions

1. **Enhance Data/Protocol Generalization**
   - Collect or simulate TSN-specific datasets and report protocol-/attack-wise metrics
2. Enhance “response-based” KD strategy….? (feature/attention/contrastive distillation?)
3. Develop preprocessing strategies
   - **Replace naive zero-padding with Protocol/Field Aware Embedding** : Instead of simply accepting 42 bytes as input, add tag embedding that indicates “which field of which protocol”
   - (e.g., input sum of [byte value embedding] + [protocol embedding] + [field position embedding])
   - **Masking Augmentation in an IVN-like way** : The paper used NLP-style 70/20/10 masking, but IVN may be better when considering field sensitivity and temporality (e.g., Field-weighted masking, time-block masking, etc.)
4. Publish real-time KPIs (latency, throughput, power) on multiple gateways
5. Hyperparameter sweeping
6. Other lightweight model frameworks to apply?

The End.