POLITECNICO DI TORINO
Repository ISTITUZIONALE

The Sweet Danger of Sugar: Debunking Representation Learning for Encrypted Traffic Classification

Original
The Sweet Danger of Sugar: Debunking Representation Learning for Encrypted Traffic Classification / Zhao, Y., Dettori, G., Boffa, M., Vassio, L., Mellia, M.. - ELETTRONICO. - (2025), pp. 296-310. (SIGCOMM '25: ACM SIGCOMM 2025 Conference Coimbra (PT) September 8 - 11, 2025) [10.1145/3718958.3750498].

Availability:
This version is available at: 11583/3003539 since: 2025-10-01T09:13:57Z

Publisher:
Association for Computing Machinery

Published
DOI:10.1145/3718958.3750498

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

18 November 2026

The Sweet Danger of Sugar: Debunking Representation Learning for Encrypted Traffic Classification

Yuqi Zhao
Politecnico di Torino
Torino, Italy
yuqi.zhao@polito.it

Giovanni Dettori
Politecnico di Torino
Torino, Italy
giovanni.dettori@polito.it

Matteo Boffa
Politecnico di Torino
Torino, Italy
matteo.boffa@polito.it

Luca Vassio
Politecnico di Torino
Torino, Italy
luca.vassio@polito.it

Marco Mellia
Politecnico di Torino
Torino, Italy
marco.mellia@polito.it

Abstract

Recently we have witnessed the explosion of proposals that, inspired by Language Models like BERT, exploit Representation Learning models to create traffic representations. All of them promise astonishing performance in encrypted traffic classification (up to 98% accuracy). In this paper, with a networking expert mindset, we critically reassess their performance. Through extensive analysis, we demonstrate that the reported successes are heavily influenced by data preparation problems, which allow these models to find easy *shortcuts* – spurious correlation between features and labels – during fine-tuning that unrealistically boost their performance. When such shortcuts are not present – as in real scenarios – these models perform poorly. We also introduce *Pcap-Encoder*, an LM-based representation learning model that we specifically design to extract features from protocol headers. *Pcap-Encoder* appears to be the only model that provides an instrumental representation for traffic classification. Yet, its complexity questions its applicability in practical settings. Our findings reveal flaws in dataset preparation and model training, calling for a better and more conscious test design. We propose a correct evaluation methodology and stress the need for rigorous benchmarking.

CCS Concepts

• **Networks** → **Packet classification**; *Network measurement*; • **Computing methodologies** → **Machine learning**.

Keywords

Traffic Classification, Representation Learning, Reproducibility, Language Models

ACM Reference Format:
Yuqi Zhao, Giovanni Dettori, Matteo Boffa, Luca Vassio, and Marco Mellia. 2025. The Sweet Danger of Sugar: Debunking Representation Learning for Encrypted Traffic Classification. In *ACM SIGCOMM 2025 Conference (SIGCOMM '25), September 8–11, 2025, Coimbra, Portugal.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3718958.3750498



This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.
SIGCOMM '25, Coimbra, Portugal
© 2025 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-1524-2/2025/09
https://doi.org/10.1145/3718958.3750498

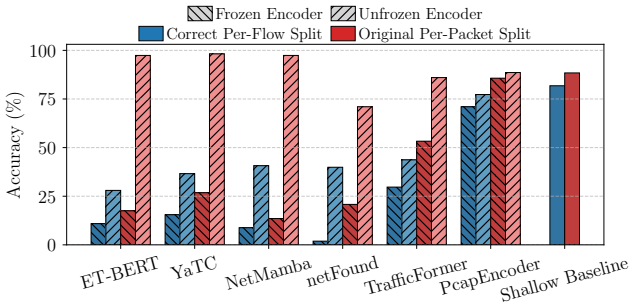


Figure 1: Accuracy of classifiers evaluated (TLS-120 dataset, packet classification task). Models' performance collapses when properly tested. *Pcap-Encoder* is the only model that maintains good performance. However, a simple shallow baseline surpass all representation learning-based methods.

1 Introduction

We are witnessing the success of Artificial Intelligence (AI), with Deep Neural Networks (DNN), Large Language Models (LLM) and multimodal models empowering applications in several fields. Pre-training using self-supervised tasks is the driving factor behind this 'AI Boom' [23]. Self-supervision trains models to solve pretext tasks – such as next-word (for text) or patch (for images) predictions [7, 19] – on humongous unlabelled datasets. Through this first *representation learning* [34] phase, pre-trained models learn to broadly master the nuances of the input data, learning how to turn texts or images into meaningful *embeddings*, i.e., compact yet highly informative numerical representations of the data. Later, such embeddings become useful to solve real-world tasks, called *downstream tasks*. This two-stage approach has proven extremely successful when solving a specific task given a few or even no examples (few-shot [30] or zero-shot learning [28]).

The allure of AI has captivated many, leading to a surge in the adoption of AI-based solutions to address network traffic classification [12, 39–41, 46], where encryption makes deep packet inspection ineffective [38, 48]. Like 'Bidirectional Encoder Representations from Transformers' (BERT) [7] learns to represent text, 'Encrypted Traffic BERT' (ET-BERT) [27] learns to represent packets, even if encrypted.

Most prior work adopts well-established AI pre-training pipelines and architectures. When fine-tuned for traffic classification, they reach up to 98% accuracy on VPN or TLS-encrypted traces. Some of the authors – quite controversially – claim that pre-training allows the model to extract patterns from the encrypted payloads [27].

As network experts, we must assess whether these proposals live up to their astonishing performance. What information can these approaches extract in an everything-encrypted setup? Do these models extract a meaningful traffic representation, or do they simply exploit *shortcuts* – spurious correlations between features and labels [15] – to predict in the downstream task?

This paper presents a systematic critical view of the adoption of representation learning for traffic classification. To ensure a fair comparison between different approaches, we define benchmarks based on open datasets, where these models face increasingly complex tasks, up to identifying traffic from 120 websites from TLS traces. Following ML principles and with a network expert mindset, we introduce a pipeline to properly assess the models' performance. We pay particular attention to possible pitfalls during cleaning, splitting, sampling and training. In this common playground, we compare the performance of state-of-the-art representation learning models. Among contenders, we include *Pcap-Encoder*, our new proposal based on the Text-to-Text Transfer Transformer (T5) [45] that we train specifically to extract the *format* and *semantics* of packet headers and to ignore any (encrypted) payload.

We highlight pitfalls that previous works underestimated and that, with promising sweet results, greatly poison the evaluation. Figure 1 summarises our findings:

• The *per-packet split* policy adopted by most of the previous work does not properly separate training samples from testing ones. This creates a huge data leak that creates shortcuts these complex models immediately exploit. Using *per-flow split*, i.e., including all packets from the same flow either in the training or testing set as in a real-world setup, suffices to remove the most prominent shortcuts. The classification accuracy barely reaches 40%.

• In the downstream tasks, all previous works train the entire classification architecture (*unfrozen* encoder) with hundreds of thousands of samples. This 'destroys' the pre-trained information and basically re-trains from scratch the entire model. When the embedders are frozen (*frozen* encoder), the accuracy of the models drops below 30%. This questions the representation learning abilities and confirms the intuition that training a model to learn patterns from the encrypted payload makes little sense.

• *Pcap-Encoder* is the only model that provides a meaningful and robust representation. By design, it exploits packet headers' information and ignores payload. However, the shallow baseline performs on par or better, with much less complexity. This calls into question representation learning at large for its practical applicability.

In summary, the pre-trained models presented in the literature fall short of producing an informative representation for traffic classification. The results we present call for a more cautious and critical view of representation learning in traffic analysis. When results appear surprisingly strong – especially where domain knowledge suggests limited learnability (e.g., encrypted traffic) – it is essential to examine why the model performs well.

We recommend the following practices:

- **Control for shortcut learning** — Be aware that deep models are particularly prone to leveraging unintended patterns instead of genuinely learning the task of interest.
- **Verify data integrity** — Ensure that the dataset is free from leakage, artefacts, or spurious correlations that the ML model would immediately exploit as shortcuts.
- **Stress representation learning capabilities** — Assess whether the model is truly learning useful representations, i.e., freeze the encoder during downstream training.
- **Consider cost-benefit trade-offs** — Compare the proposed approach against simple baselines to determine whether the added complexity of deep learning models is justified.

We believe our lesson extends to all AI-based solutions for computer networks. To this end, we provide the code, benchmark datasets and methodology to the community to establish a shared environment for development and testing[1].

2 Representation Learning: Core Principles

We introduce the fundamentals and intuitions of representation learning for readers who are not experts in the field.

Representation learning. Machine learning models, as computational systems, inherently work with numerical vectors. The primary goal of *representation learning* is to learn 'meaningful' mappings that encode the real properties of an input into a numerical space called the *embedding space*. The component in charge of learning this mapping is named *Encoder* which is *pre-trained* using pretext and self-supervised tasks. An embedding space is meaningful when it respects and captures the initial data properties. One way of measuring such alignment is to challenge a model to take advantage of the learned embeddings and solve, possibly with additional supervision, tasks that require an understanding of real-world properties. Such tasks, often of practical interest, are defined *downstream tasks*.

Pre-training for robust representation learning. In recent years, *pre-training* proved a compelling way to learn meaningful embeddings. Especially when dealing with non-numerical inputs like images [24] and text [20], pre-training demonstrated that it is possible to learn embeddings that automatically capture generic features and relationships of the raw data and can be exploited to solve many downstream tasks, with few (or even no) extra supervision [19, 28, 30]. Representation learning eliminates the need for *feature engineering*, i.e., to manually select or create relevant features. Instead, the model autonomously defines features directly from the raw input data.

Pre-training through pretext tasks. Pre-training involves training the model on a series of *self-supervised pretext tasks*. Unlike traditional end-to-end learning, this approach does not rely on externally provided labels. Examples of pretext tasks include predicting the next word or phrase in a document [11] or reconstructing a masked patch of an image [19]. In both cases, the 'correct label' is inherently derived from the input data itself. The component in charge of mapping the embedding space to the pre-text output is named *Decoder*. The whole Encoder/Decoder architecture

[1] https://github.com/SmartData-Polito/Debunk_Traffic_Representation

is trained on these self-supervised pretext tasks so to minimize the decoder error when compared to the correct data. Notice that, unlike traditional feature engineering – where one explicitly models what they consider to be relevant aspects of the data – pretext tasks encourage the model to independently uncover and understand which information to use to solve the task. The fundamental assumption here is that these features are indeed present, making it possible, for instance, to reconstruct a missing portion of an image given the remaining parts. Ultimately, the literature agrees that the size and diversity of the pre-training dataset are key factors for its successful [7, 22]. This is logical, as the embeddings shall capture generalizable aspects of the input data and avoid focusing on overly specific scenarios.

Leveraging the embeddings for downstream tasks. A *downstream task* refers to a specific problem a model is designed to solve, e.g., the classification of samples into classes, as we consider here. Leveraging pre-trained architectures is an efficient approach for solving downstream tasks. The pre-trained encoder is first used to extract embeddings from the input data. A classification head is then added to use the knowledge captured by the encoder and perform the final classification. The classification head, which ranges from a shallow model (e.g., Random Forest (RF) or simple K-NN classifier) to a Neural Network (e.g., Multi-Layer Perceptron (MLP)), works alongside the pre-trained encoder to form an overall *classification model*. Training this classification model for a specific downstream task requires labelled datasets for both train and test.

Frozen and unfrozen representation. There are coarsely two options for training the classification model: (i) train only the classification head while keeping the pre-trained encoder *frozen*; (ii) train the entire architecture end-to-end, with an *unfrozen* pre-trained encoder. The latter is often referred to as *fine-tuning*, as it involves tailoring the general representations learned by the pre-trained model to address the particular task. Although fine-tuning the entire model often yields better results, this is more computationally and memory-expensive [26]. Additionally, when performed on a large amount of supervised data, end-to-end fine-tuning can significantly alter the encoder representation, potentially causing the model to forget its pre-trained knowledge and 'overfit' to the downstream task. Sometimes, this can also lead the model to rely on tailored signals or *shortcuts*. Shortcuts are decision rules that perform well on standard benchmarks but do not transfer to more challenging testing conditions [15] rather than robust features that truly represent the underlying task [32].

3 Representation Learning: Network Traffic

Researchers are exploring representation learning adoption to automatically learn meaningful representations of network traffic for tasks like traffic classification or QoE estimation. This section provides an overview of the most cited and recent approaches. The key characteristics of these solutions are summarized in Table 1.

To adopt representation learning strategies, all proposed solutions follow a set of common steps which we analyse below, highlighting any potential pitfalls in the proposals[2].

3.1 Choice of Model Architecture

The first choice to make is whether to design a new encoder model or select an architecture previously presented in other areas. For traffic representation, all previous work builds on neural architectures presented in NLP or CV fields.

Literature choices: The motivation for using NLP-style models comes from the parallel between text, i.e., sequences of characters organised in words, sentences, etc., and network traffic, i.e., sequences of bytes organised in fields, packets, flows, etc. *PacRep* [33], *PERT* [18], *ET-BERT* [27], *TrafficFormer* [54], *netFound* [17] and *PTU* [42] use BERT-like models[7, 25] borrowed from NLP and define network traffic specific pretext tasks for their training.

Other approaches draw inspiration from the field of image processing. They represent packets in the same flow as rows in a matrix to build an image. With this, they leverage tools such as *Vision Transformer* (ViT) [8] or *Mamba* [16] to encode image-like inputs into the embedding space. *YaTC* [53] and *NetMamba* [49] fall into this class.

3.2 Pre-training Dataset

For pre-training, collecting a large volume of unlabelled data is crucial to ensure comprehensive coverage of the main protocols. The commonly adopted strategy is to passively collect traces through network sniffers, and to leverage large publicly available datasets. Best practice suggests data used for pre-training to be much larger than those used for downstream task training – as *few-shot* learning should suffice if the learned representation is effective[3]. Additionally, the upstream and downstream datasets shall be different to limit model overfitting and data leakage.

Literature choices: Different works can freely use different datasets for pre-training. However, some studies employ the same datasets – or part of them – for both the pretraining and downstream tasks. For example, *ET-BERT* uses the *ISCX-VPN* [9] and (likely) *CSTNET-TLS1.3* [27] datasets for both tasks. Also, *YaTC* and *NetMamba* rely on the same datasets for upstream and downstream tasks training.

Associated pitfalls: Dataset reuse is uncommon and discouraged in the CV and NLP domains. While for pre-training large datasets are strongly suggested, in the downstream task a large amount of data (likely) leads to forgetting – especially when the representation model is unfrozen. Indeed, the encoder could override its pre-training knowledge and instead memorize task-specific patterns that can deceptively enhance the performance on downstream tasks. We will discuss this later in Sec. 4.2.

3.3 Choice of Pre-training Tasks

With pre-training, the model learns some generic and task-agnostic data patterns from the data themselves. Usual tasks require the encoder to develop predictive or reconstruction skills, often by masking part of the data or leveraging the data's temporal nature to predict future properties.

Literature choices: A common pre-training in networking involves *reconstructing masked bytes* in a packet: The intuition behind

[2]We based our discussion on the information in the papers and verified missing details using the authors' code and models when available.

[3]For BERT, the ratio between supervised and self-supervised data samples ranges between 1:1,000 and 1:1,000,000.

Model	Pre-training				Downstream Classification			
	Architecture	Embedding Size	Task Types	Dataset	Cleaning	Split	# Tasks	Datasets
PacRep [33]	BERT	768	None	Not needed	Partial	Packet	6	A, B, +
PERT [18]	ALBERT	768	MAE	≠	No	Flow	2	A, +
ET-BERT [27]	BERT	768	MAE, SBP	∩	Partial	Packet	7	A, B, C, +
PTU [42]	BERT	768	MAE, SSP, HIP, FIP	≠	No	Packet	7	A, B, C, +
TrafficFormer [54]	BERT	768	MAE, SODF	∩	Partial	Packet	6	A, B, C, +
netFound [17]	BERT	1024	MAE	≠	Partial	Flow	5	A, +
YaTC [53]	ViT	192	MAE	=	No	Unknown	4	A, B, +
NetMamba [49]	Mamba	256	MAE	=	Partial	Flow	6	A, B, +
Pcap-Encoder	T5	768	Autoencoder, Q&A	≠	Full	Flow	6	A, B, C

Datasets: A=ISCX-VPN, B=USTC-TFC, C=CSTNET-TLS1.3, +=other

Table 1: Summary of representation learning models for traffic classification. Pitfalls are highlighted in red.

this task is to encourage the model to identify correlations within the unmasked input to reconstruct missing parts. *NetMamba*, *YaTC*, *netFound* and *PERT* adopt this pre-training strategy, named *Masked Autoencoder* (MAE) [19].

ET-BERT uses the original BERT pretext tasks of Masked Language Model (MLM) and Next Sentence Prediction (NSP). In *ET-BERT* they call them *Masked Burst Modelling* (MBM) – a MAE-style task – and *Same-origin Burst Prediction* (SBP): given two packets, the model is queried whether the packets are part of the same burst[4].

TrafficFormer keeps the first MAE task from *ET-BERT*, but further complicates the second into *Same Origin-Direction-Flow* (SODF): the model is not required to solve just a binary problem as in SBP, but also has to guess the direction, order, and corresponding flow of the packet.

PTU also builds on *ET-BERT* MAE, but adds a *Same Session Prediction* (SSP) task, where the model has to predict whether two packets belong to the same session, and the *Historical and Future Interval Prediction* (HIP and FIP) tasks to predict the time of arrival of previous and future packets in a flow.

netFound uses the header information (e.g., packet length, *TTL*, etc.) and the first 12 bytes of information in the payload, converts them into tokens, and then employs the standard MAE method for pre-training.

Finally, different from the others, *PacRep* uses the off-the-shelf BERT model trained on text and does not design any network-specific pretext task.

Associated pitfalls: Both linguistic and vision studies showed that words in a sentence and patches in an image exhibit significant correlations [3, 5, 31]. Contrarily, supposing a robust encryption algorithm, there is no correlation between the encrypted bytes in a packet payload. Hence, MAE tasks on encrypted payloads make little or no sense in the networking scenario. This is why more recent works, such as *netFound*, only focus on unencrypted content during pre-training [17].

3.4 Mitigating the pitfalls: *Pcap-Encoder*

We propose a representation learning architecture that we explicitly design and train to automatically extract information from the protocol headers only – that we assume still carry plain-text information. We call our proposed model architecture *Pcap-Encoder*. *Pcap-Encoder* leverages two sequential pre-training phases that aim at capturing the contextual relationship among bytes and automatically extract the semantic of some packet header fields.

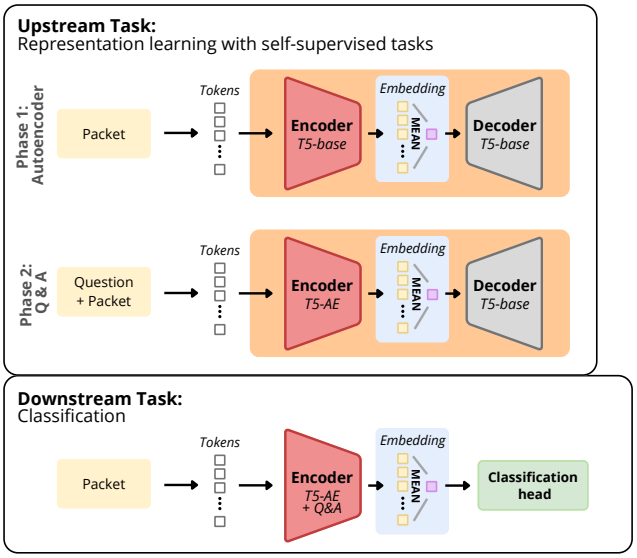


Figure 2: Schema of *Pcap-Encoder*, our proposal.

The overall architecture is shown in Figure 2. *Pcap-Encoder* is based on the well-known *T5* (base) architecture [45], a sequence-to-sequence transformer-based model. T5 solves question-answering tasks: prompted a question and offered a context, the model learns how to answer. For more details about the design of *Pcap-Encoder*, check Appendix A.1 and our technical report [6].

Phase 1: Encoder update. We first update the original T5 encoder using raw packet traces to adapt it to the new data format and semantics. The encoder's goal is to map the original data into a numerical space, possibly removing redundant or useless (e.g., constant) information. For this, we train the (base) T5 to reconstruct the original packet from the internal representation.

[4] *Burst*: sequence of consecutive packets that belong to the same flow.

T5 works using tokens to represent the input text. We convert each 2-byte long word into a hexadecimal number, separating each word by space. We feed this textual input to the original T5 tokenizer to generate tokens. The encoder receives the packet divided into tokens and obtains a representation vector for each token. We modify the T5 encoder and add a bottleneck to obtain a single representation for the entire packet from the representations of its tokens. We test different architectures for this bottleneck and a simple *mean pooling* layer suffices (see Appendix A.1). Finally, the decoder reconstructs the original packet tokens[5].

We start from the pre-trained *T5-base* encoder and continue the training on traffic data with a standard cross-entropy loss function based on the difference between the predicted tokens and the actual ones. At the end, the encoder gives us a single representation of the input packet. We call this pre-trained model *T5-AE*. For this pre-training, we use MAWI [2], UNSW-NB15 [37] traces and a freshly collected trace from our campus[6]. This ensures both spatial and time diversity in the samples. Traces include IPv4 and IPv6 traffic, and mostly TCP, UDP and some ICMP traffic. In total, we use ≈ 1GB of data or 500k packets.

Phase 2: Question-answering. Next, we fine-tune the *T5-AE* model to extract the semantics of protocol headers. We task the model to answer questions related to some specific packet header or fields. In total, we define types of 8 questions, and create 50,000 instances of questions. We included (see Appendix A.1) retrieval questions applied across different protocols (TCP-IPv4/6, UDP-IPv4, and ICMP) and more complex questions that involve the computation on different fields. Two examples of question prompts are 'What is the destination IP address of the packet?' and 'Is the packet's IP checksum correct?'. We stress that we avoid pretext questions on the application payload (except its size), assuming encryption prevents any possible answer on the content.

This simple Q&A training model allows one to pre-train *Pcap-Encoder* on specific protocol information. For instance, it can be tasked to find the SNI in the TLS handshake or learn where to find the A or AAAA record in DNS queries.

We start from the *T5-AE* encoder trained at Phase 1 using the mean pooling bottleneck to represent each packet. For the question-answering task, the input consists of two parts: the query and the context. The query is the task we ask the model to solve. The context is the packet. Queries are in plaintext, so we use the original T5 tokenizer directly. For packets, we tokenize them as before. At last, we separate the query from the context by the special token `</s>` that ensures the model correctly interprets the boundaries between the query and the context. For example: *What is the time to live of the packet?*`</s>4500 4000 F7C6 ... CD19`.

We use the same traces as before for this second phase. Ablation studies on the *T5-AE* and *Q&A* modules show that each component plays a beneficial role in enhancing model performance (details in Table 11). At the end, we have a *T5-AE+Q&A* pre-trained model.

Downstream classifiers: As in other representation learning models proposed for traffic classification, we add a classification head made by a two-layer MLP with a ReLU activation function. It takes as input the *T5-AE+Q&A* embedding computed from the input packet. We train separated classifiers, one for each task. Depending on the number of classes in the task, we use binary cross-entropy or softmax as a loss function for the MLP.

[5]To implement the encoder update, we use a dummy question and pass the packet over which the model operates.

[6]To ensure diversity and avoid the model memorising constant patterns, we randomize IP addresses and TTL values.

4 Benchmark for Network Traffic Classification

In this section, we shift focus to the downstream task of the representation learning pipeline. As in the previous section, we explore the design choices of the most influential and recent approaches, highlighting potential pitfalls. Additionally, to provide a common ground for comparing results, we introduce a fair and effective benchmark pipeline to evaluate the representation learning capabilities of different models for network traffic classification tasks. Our benchmark standardizes the critical steps of trace gathering, cleaning, splitting, and sampling – processes that are often inconsistently or incorrectly handled in the literature, making it difficult to fairly compare different methods.

4.1 Dataset Preparation: Collecting, Cleaning, Splitting

We first discuss the choices on the datasets that previous authors used to train and evaluate their solutions on downstream tasks. The picture is extremely heterogeneous. Most authors use publicly available datasets, while a few autonomously collect traces [17, 18, 27] – some sharing (part of) them [27]. Each paper proposes a *custom cleaning process* that removes spurious traffic (e.g., ARP, DHCP, LAN-related protocols, etc.). Some remove flows or packets shorter than a given threshold [17, 27, 49, 54]. Some perform *careless train-test splits*, ignoring the fact that packets of the same flow might leak information on the classification class [27, 33, 42]. In general, even when starting from the same dataset, all papers end up with a custom collection: even the number of classes per task often differs.

We hereby call for a standardization in the following. The process we propose here can be extended to include other datasets and classification tasks.

Choice of dataset: Instead of setting up specific data collection campaigns, we rely on previously used mainstream datasets and classification tasks created by the research community. These labelled datasets were generated through experiments conducted in controlled testbeds and offer a large data collection of encrypted traffic. We select three datasets among those commonly used in previous works, for a total of six tasks that we summarise in Table 2.

- *ISCX-VPN* [9]: This dataset contains traffic related to 6 different types of services (Web browsing, VoIP, Video Streaming, Chat, Email, P2P File transfer) using different applications (e.g., Chat with Skype or Hangouts), over plain or VPN-encrypted connection. We define three tasks: determining whether traffic is VPN-encrypted or not (*VPN-binary*); Service classification (*VPN-service*); and application classification (*VPN-app*).
- *USTC-TFC* [50]: This dataset contains a total of 20 applications, 10 are benign (BitTorrent, FaceTime, Gmail, Skype, ...) and 10 are malicious (malware run in controlled environments). We formulate two classification tasks: Malicious or not (*USTC-binary*); and application classification (*USTC-app*).
- *CSTN-TLS1.3* [27]: This dataset contains a total of 120 classes, each referring to visits to a different TLS1.3-enabled website. The

Dataset	Task	#Class	#Train	#Test	Description
ISCX-VPN	VPN-binary	2	100,000	110,594	Encrypted?
ISCX-VPN	VPN-service	6	120,000	111,368	Voip, Chat, ...
ISCX-VPN	VPN-app	16	33,088	111,678	Gmail, Vimeo, ...
USTC-TFC	USTC-binary	2	100,000	609,332	Malware?
USTC-TFC	USTC-app	20	69,680	609,477	Gmail, Skype, ...
CSTN-TLS1.3	TLS-120	120	98,640	553,994	120 Websites

Table 2: Downstream datasets and tasks.

task here is to output the visited website (*TLS-120*). The authors share only TCP flows from which they remove the TCP 3-way-handshake and the initial client TLS-Hello – thus removing the plain-text SNI if present. This results in an 'everything encrypted' payload scenario[7].

Data cleaning: Not supervising the trace collection process, data cleaning becomes a crucial step to ensure the quality and reliability of the datasets [13]. We summarize our interventions in the following four cases:

• *Extraneous protocol filters*: Given the constraints of network data collection, certain extraneous protocols inevitably make their way into the datasets. Some traces include ARP, DHCP, broadcast protocols, etc. that question the definition of the classification task (e.g., making predictions on ARP requests, which are not related to any classes). Some of the previous works [18, 33, 42, 53] did not clean (or did not report how they cleaned) the traces, blindly trusting the data collection. Besides, *netFound* retains traffic data related to the TCP, UDP and ICMP. For our benchmark, *we define a superset of filters that we report in Appendix A.4 to filter out the irrelevant protocols to the classification task.* ISCX and USTC traces contain 5% and 10% of spurious packets, respectively. CSTN is already filtered.

• *Minimum size filters*: Some of the previous work filtered packets shorter than a minimum size [17, 27, 49, 54]. For example, in *ET-BERT*, the authors remove all packets shorter than 80B[8]. In *TrafficFormer*, the authors remove flows shorter than 2kB or than three packets. In *netFound*, the authors exclude flows with fewer than six packets and bursts containing two or fewer packets. Filtering based on packet or flow size alters the classification tasks since, for instance, all TCP signalling and acknowledgement packets could be ignored. Hence, *we do not adopt and support filters based on minimum size/number.*

• *Classes support filters*: Some works limit the number of packets per class [27], the number of flows per class [27, 49, 54], or directly drop a class if the minimum support is not reached [27, 49, 54]. For example, *TrafficFormer* discards classes with less than 10 flows and limits to 500 flows the others; *ET-BERT* selects at most 5000 packets or 500 flows per class; *NetMamba* discards rare classes and limits common ones, but authors do not report thresholds. Since these filters alter the original data distribution, they change the nature of the problem compared to the initial dataset. The original datasets should reflect real-world conditions, while artificially modifying the underlying distributions introduces deviations that may impact the performance in practical deployments. *We refrain from applying any class removal during testing. Differently, during training, researchers could use techniques that change the original class distribution, such as balancing the class samples (see next).*

[7]In the original *ET-BERT* paper the authors state the SNI is present, but in the public dataset it is not.

[8]This filter is present in the code, but not mentioned in the paper.

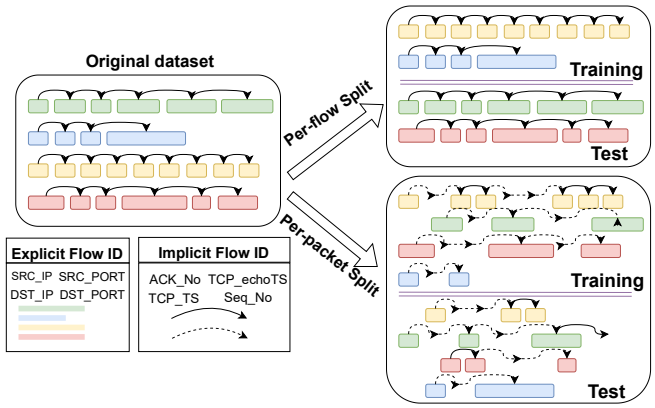


Figure 3: Per-flow and per-packet split.

• *Filters removing header information*: In an attempt to limit data leakage (i.e., identifiers the model could memorize), some works propose to anonymise specific fields like IP addresses and TCP/UDP ports [27, 33, 42, 49, 53, 54]. For example, *YaTC* randomises the IP address and sets the port number to zero; *PacRep* and *NetMamba* set both IP address and port to zero; *PTU* removes IP address, MAC address and checksum; *TrafficFormer* randomises IP address and ports, and some specific fields (such as timestamp) for data augmentation. *ET-BERT* removes the IP header entirely. *netFound* omits the explicit flow identifiers (e.g., IP address, port, SNIs, etc), but keeps other header's information.

For the pre-training task, we consider it incorrect to remove any information from the header. In the downstream task, removing selected header fields compels the model to generalize across diverse network conditions – for instance, it can no longer rely on memorizing that a specific IP range belongs to one server. *It is hence part of the downstream model and training task design to hide some information (and force the model to generalise) or to leave such features and create a more dedicated model.*

Dataset Splitting into Train and Test: The golden rule in any ML pipeline is to avoid any leakage of information from the test set into the training/validation set. For traffic classification, we consider two basic splitting processes:

• *Per-packet split:* Separate the packets based on their class; randomly split each class into train, validation and test sets.

• *Per-flow split:* Separate the flows based on their class; randomly put all packets from the same flow into either train, validation and test sets.

Fig. 3 sketches the differences between per-flow and per-packet split. Given flows, there are *explicit flow identifiers* (ID) like the flow 5-tuple (represented by colours), and *implicit flow IDs*, such as the TCP sequence (SeqNo) and ACK numbers (AckNo). For instance, TCP SeqNo and AckNo are randomly selected during the TCP three-way handshake, and all packets of the same flow then share values in a close range. Overall, the pairs (SeqNo, AckNo) create an implicit flow ID projecting all flow packets in a random space of ≈64 bits.

Similarly, the TCP timestamp option implicitly groups packets of the same session by a close-by timestamp.

All previous works adopt a per-packet split for packet classification tasks. Unfortunately, this leads to serious data leakage that allows the classifier to leverage implicit flow IDs to identify all packets of the same flow. Being the class of the flow available during training, the classifier can easily associate a packet to its flow, and then to its class. Therefore, *we propose the adoption of the per-flow split to remove simple implicit flow IDs that the classifier would not be able to leverage in real deployments.*

Notice that more advanced splits are possible: per-session, per-client, per-location, per-time split, etc. Each stresses the ability of the model to generalise when transferred to other setups. Here, we limit our analysis to the basic per-packet and per-flow split.

Sampling a subset of the original dataset: Different network applications generate different amounts of flows and packets. This translates into a class imbalance where a chatty application may exchange more packets than non-verbose applications, possibly creating an important class imbalance that challenges the downstream model training. However, this imbalance is real: Limiting the number of packets or flows per class on the entire dataset, as in some of the previous works, introduces artifacts in the data distribution that could bias the model evaluation. *Therefore, for the test set, we suggest not altering the class sample distribution as obtained from the previous step.* If for some reason (e.g., computational or time constraints at inference time) the test set must be reduced, *we suggest stratified sampling, which preserves the distribution of classes as in the collected data.*

For the training (and validation) set, multiple valid choices can be made according to the used methodology. *Balanced sampling* through oversampling or undersampling makes the number of samples of every class similar. This enables the model to better learn minority class samples and avoid the majority class dominating the model training process. Alternatively, one could use a weighted loss function or other ML techniques to compensate for class unbalance.

4.2 Downstream classification

Downstream task - Packet or flow: Some work frames the final task as a packet classification problem [33, 42]; others focus on flow classification [17, 18, 49, 53, 54]; some does both [27]. Here, we face both packet-level and flow-level tasks: given a packet (a flow), identify which class it belongs to. We define flow as the sequence of packets with the same 5-tuple, and consider both packets sent by the client and the server (bi-flow). Finally, we consider all packets and flows to belong to the class the trace belongs to. For instance, when visiting a website, all cleaned packets and flows collected during such a visit inherit the same website label[9].

Downstream model and training: Each pre-trained encoder outputs an embedding, given some input sample. This tensor is the input to the classification head. In previous works, the classification head ranges from a simple MLP to a more complex transformer-based architecture (e.g. in *PacRep*). Given the resulting classification model, authors trained it with simple supervision or using contrastive learning as in *PacRep*. Given the complexity of the downstream task, *one is free to choose the classification head model, but should keep in mind that the main learning effort lies in the self-supervised encoder – so excessively complex heads are typically unnecessary and may not provide significant gains.*

Pre-Trained Encoder - Frozen or unfrozen: During the training of the final classifier, all previous works perform training in an end-to-end manner, i.e., the encoder architecture is *unfrozen*. While legitimate, this is in contrast with the idea that the representation produced by the encoder is actually representative. By freezing the encoding part of the model, the classification head should leverage the obtained generic representation. Therefore, to check how meaningful the representation is, *we advocate the usage of frozen encoder during the training of the downstream classifier.*

Performance metrics: Accuracy and macro F1-Score. *Accuracy* measures the number of correct predictions. It is the ratio between the total number of correct predictions and the total number of predictions. Accuracy treats all samples as equally important. In an unbalanced situation, it underweights the performance of minority classes.

Macro-averaged F1-Score is the arithmetic mean (i.e., unweighted mean) of all F1 scores per class. This metric equally weights errors across all classes, regardless of support.

All previous works but *PacRep* and *netFound* report the accuracy. Correctly, some[18, 27, 42, 54] present the macro F1-Score too. *YaTC*, *NetMamba* and *netFound*[10] misleadingly use the micro F1-Score – which favours majority classes; *PacRep* only reports micro F1 and macro F1 scores.

To evaluate the performance of the classifier, we suggest using both the accuracy and the macro-averaged F1 score.

5 Experimental setup

Here we describe the experimental setup to compare and understand the real potential of representation learning for the traffic classification tasks.

Downstream models: For our experiments, we select five representative models: *ET-BERT*, *YaTC*, *NetMamba*, *TrafficFormer* and *netFound*. They use different representation learning models: BERT, ViT and Mamba. We download the pre-trained models from each original repository. We add *Pcap-Encoder*, and compare against shallow models (without representation learning) as baselines.

For each model, we follow the data preparation and hyperparameters suggested by the original papers when available. Refer to Appendix A.2 for a detailed overview. When facing *packet classification* with the flow-embedders, we *Repeat* the same packet 5 times to form an artificial flow and get the resulting embeddings in output[11]. For *netFound*, we fill its maximum input (72 packets) with the same packet and pad the multimodal information (packet direction, packet interval, and etc.) with the same value or zero. We use the original flow-embedder models for the *flow classification task*. For *Pcap-Encoder*, we consider a simple majority voting on the classification of the first 5 packets of each flow.

[9] Even if questionable, this is the same formulation previous works used.

[10] Detail inferred from the code.

[11] We also test a *Padding* strategy where 4 padding packets with all zeros follow the packet. The Repeat strategy offers better results.

Model (Per-flow split)	VPN-binary (2)		VPN-service (6)		VPN-app (16)		USTC-binary (2)		USTC-app (20)		TLS-120	
	AC	F1	AC	F1	AC	F1	AC	F1	AC	F1	AC	F1
ET-BERT	84.7	84.6	71.7	64.2	59.2	43.7	100.0	100.0	84.9	79.6	10.9	6.7
YaTC	83.9	83.9	69.2	60.1	60.9	44.3	99.5	99.5	85.2	78.0	15.5	9.6
NetMamba	75.0	74.5	56.9	49.0	39.6	28.4	97.6	97.5	72.5	57.7	8.8	4.5
TrafficFormer	90.9	90.9	76.5	69.4	67.7	54.4	100.0	100.0	72.0	65.0	29.7	24.0
netFound	76.0	61.9	47.3	36.5	32.9	15.3	99.4	99.4	58.0	30.7	1.9	0.5
Pcap-Encoder	**99.9**	**99.9**	**92.1**	**89.8**	**83.5**	**71.0**	**100.0**	**100.0**	**91.0**	**87.1**	**71.0**	**63.7**

Table 3: Results of *Pcap-Encoder* and the three SoA models for packet classification. Per-flow split, Frozen encoders. We report accuracy (AC) and macro F1-Score (F1). Results below 50% are highlighted in red, best in bold.

Downstream task evaluation: As reported in Section 4, we split each dataset into training and test partitions, according to a 7:1 ratio. We adopt the *per-flow split* strategy. We make sure that long flows are evenly distributed within the partitions. To stress the few-shot learning abilities of the models, in the training set, we *balance* classes by undersampling each class to the minority class. For flows longer than 1,000 packets, we randomly select 1,000 packets across the flow. Table 2 reports the number of samples in the training and testing sets. The proportion between the train and test samples changes for each downstream dataset due to the undersampling of the training part.

We perform a K-Fold cross-validation of the training partition with $K = 3$ (2/3 used for training, 1/3 for validation, 3 folds). In all experiments, we test all models and configurations with the exact same splits for a fair comparison.

To compare with the *per-packet split* scenario, we create a second split simply following an 8:1:1 random split into train, validation and test – as originally proposed in *ET-BERT*. In this case, packets from the same flow can end up in both trains and tests.

6 Results

We perform all the experiments HPC Cluster equipped with NVIDIA Tesla V100 SXM2 GPUs. We use Python and Pytorch for the implementation and make all datasets, models, and code available to the community for reproducibility and to foster further studies.

6.1 Packet-Level Traffic Classification

We start from the proposed flow-split-based scenario with frozen encoders on the packet classification tasks.

Per-Flow Split – Frozen encoder: We consider the six tasks we presented in Section 4 and test all representation learning models with the per-flow split and frozen encoders. We report results in Table 3. Surprisingly, the performance of all models is very poor, up to 80% lower than that reported in their respective papers. Only in the simplest binary classification tasks, *VPN-binary* and *USTC-binary*, all models offer solid results. On the most challenging *VPN-app* and *TLS-120* tasks, the performance is really disappointing.

Notice how *Pcap-Encoder* performs best in all tasks, significantly outperforming other methods. Yet, in the most complex tasks, it struggles to achieve excellent results.

Model (Per-flow split)	VPN-app (16)				TLS-120			
	Frozen		Unfrozen		Frozen		Unfrozen	
	AC	F1	AC	F1	AC	F1	AC	F1
ET-BERT	59.2	43.7	82.8	69.7	10.9	6.7	28.0	21.5
YaTC	60.9	44.3	79.1	65.2	15.5	9.6	36.6	31.4
NetMamba	39.6	28.4	80.4	65.9	8.8	4.5	40.7	35.3
TrafficFormer	67.7	54.4	73.1	61.0	29.7	24.0	43.7	38.9
netFound	32.9	15.3	70.4	57.3	1.9	0.5	39.9	35.2
Pcap-Encoder	**83.5**	**71.0**	**85.6**	**74.8**	**71.0**	**63.7**	**77.3**	**69.2**

Table 4: Per-flow split, frozen and unfrozen encoder. Results improve, but models still struggle in challenging setups.

Although it contrasts with previous studies, the poor performance in encrypted scenarios is justified by the debatable assumption made by previous work according to which some information can still be extracted from the encrypted payload. Since these models are designed to disregard information from packet headers, they rely on minimal data, primarily packet direction and size. In contrast, *Pcap-Encoder* provides an informative representation to the classification head, enabling it to distinguish the application that generated a packet from the network and the transport headers summarised by *Pcap-Encoder*.

Given that some tasks are very simple, in the following, we dismiss three and focus only on the *VPN-app* and *TLS-120* tasks.

Per-Flow Split – Unfrozen Encoder: To investigate the root cause of poor performance, we let the classification model fine-tune the embedder part as well, i.e., we unfreeze the encoder. We report results in Table 4 for *VPN-app* and *TLS-120* cases. As expected, all proposed models improve their performance. Still, unfreezing the encoder does not suffice to reach a satisfactory result for none of them. *Pcap-Encoder* still achieves the best performance: Notice that the unfreezing boost is less significant than the counterparts – about 5% improvement only. This confirms that *Pcap-Encoder* still relies on its pre-training knowledge and does not require re-learning with end-to-end training.

Per-Packet Split – Frozen vs Unfrozen Encoder: Wondering why the performance does not yet saturate to the promised good performance, we observe what happens with the per-packet split as originally adopted in all previous works. We present results in

Model (Per-packet split)	VPN-app (16)				TLS-120			
	Frozen		Unfrozen		Frozen		Unfrozen	
	AC	F1	AC	F1	AC	F1	AC	F1
ET-BERT	69.5	64.7	96.8	97.0	17.5	10.2	97.4	96.8
YaTC	73.2	67.7	**98.5**	**98.5**	26.8	17.7	**98.2**	**97.7**
NetMamba	53.5	45.1	98.4	98.4	13.5	5.3	97.4	96.8
TrafficFormer	87.5	85.6	95.6	95.2	53.3	48.2	86.0	83.3
netFound	35.3	18.7	89.6	89.0	10.8	2.3	71.0	67.4
Pcap-Encoder	**91.9**	**90.6**	94.3	93.9	**85.7**	**81.0**	88.6	80.3

Table 5: Per-packet split scenario. Eventually, in this wrong settings and unfrozen encoder, performance reaches the promised > 90% accuracy.

Table 5 for both frozen and unfrozen encoder setups on the two most challenging tasks.

First, focus on the frozen encoder scenario. The performance that all previous models achieve is still far from satisfactory. In particular, in the *TLS-120*, we obtain very poor results. Compare now with an unfrozen encoder: The accuracy finally largely improves, reaching up to 98%.

Two main take-home messages arise: *First, per-packet split enables some data leakage that allows the classifier to finally find exploitable patterns. Second, the representation offered by the frozen encoder is not useful for solving classification tasks.* Only when end-to-end training is enabled, the models learn patterns that allow them to shine.

To gauge the representativeness of the embeddings, we compute, for each packet, the number of neighbouring packets that have the same class, using their representation in the embedding space. The intuition is that if the embedder can project packets with the same class in the same portion of the embedding space, most of the packet's neighbours should be of the same class. For each point, we consider the 5-nearest neighbours (5-NN) and count how many samples are of the correct class (5-NN purity). Fig. 4 shows the analysis considering *ET-BERT* embeddings in frozen (left) and unfrozen (right) setups, in the *TLS-120* case. Given a packet of class c, in the former case, 71% of packets have no neighbour of class c. After fine-tuning, the embedding changes drastically, so that now 97% of packets have all the 5 neighbours of class c. The original embeddings lack meaningful information, and it is only during end-to-end fine-tuning that the classification model adapts them to specifically address the downstream tasks. The same holds for the embeddings of other models, not reported here for the sake of brevity. In a nutshell, the models need to modify all their weights to solve the classification task, as their original representations are uninformative.

Unfortunately, the patterns activated by the per-packet split are misleading and impractical for real-world use. The per-packet split strategy suffers from severe data leakage, allowing packets from the same flow to appear in both the training and test sets. As a result, the model learns information it should not rely on, i.e., it relies on shortcuts. Specifically, some implicit flow IDs enable the model to link a test packet to its corresponding flow and, ultimately, its class.

Removing implicit flow IDs: We systematically analyse the cause of the performance increase with the per-packet split and

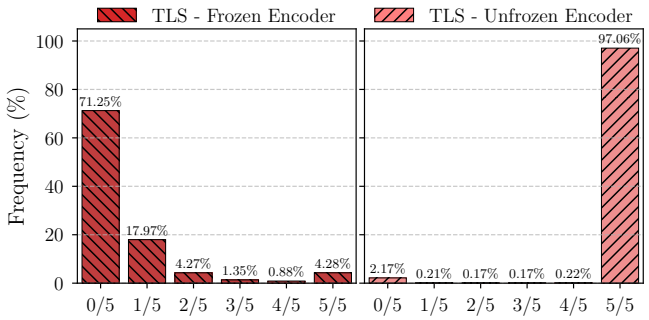


Figure 4: 5-NN purity of embeddings for *ET-BERT*. With a frozen encoder, 71% of points do not have a sample of the same class as TOP-5 neighbour. Situation changes only when the encoder is unfrozen.

Scenario	Dataset	AC	F1
Per-packet Split	Original	**97.4**	**96.8**
	w/o SeqNo/AckNo w/o Timestamp (only test)	19.5	15.4
	w/o SeqNo/AckNo w/o Timestamp (train + test)	52.2	48.2
	w/o Pre-training	97.1	96.4
Per-flow Split	Original	28.0	21.5

Table 6: Impact of implicit flow ID on the unfrozen *ET-BERT* and ablation study on the pre-training strategy.

unfrozen model. We focus on *ET-BERT* as a case study, with the *TLS-120* task. Table 6 summarises these experiments. The first row reports (as in Table 5) the excellent performance with the per-packet split and unfrozen encoder.

Now, consider the same model tested on the same data but with randomized SeqNo, AckNo, and TCP timestamps (second row). The results drop by approximately 80%. This suggests that the model relies on these shortcuts during training, leading to an abrupt performance decline when those shortcuts are unavailable.

Train the same model now on a training set where the implicit flow IDs are removed – and no easy shortcuts are present. Test this model on a test set without shortcuts too. Results improve. That is, the model – trained unfrozen – looks for other patterns that still allow it to solve the classification task, even if in an unsatisfactory manner. Some data leakage is still present.

To check the effectiveness *ET-BERT* pre-training strategy, we run an experiment in which we destroy the pre-training knowledge by initialising the ET-BERT model weights with random values. We then fine-tune this untrained model to solve the TLS-120 downstream task. We report results in Table 6 in the w/o Pre-training row. Results are on par with those of the pre-trained *ET-BERT* model. This strongly suggests that the *ET-BERT* pre-training is mostly useless.

Model (Per-flow split)	VPN-app (16)	TLS-120
w/o IP addr.	52.5	13.0
w/o header	16.4	1.5
w/o payload	66.7	63.6
base	71.0	63.7

Table 7: Ablation Study on *Pcap-Encoder* in the flow-based split scenario when removing the IPs, headers and payloads (Macro F1-Scores).

Model	VPN-app (16)		TLS-120	
(Per-flow split)	base	w/o IP addr	base	w/o IP addr
RF	81.1	72.4	78.0	39.4
XGBoost	82.1	73.2	82.0	41.3
LightGBM	82.6	74.5	82.4	40.6
MLP	65.1	52.5	68.8	30.5

Table 8: Macro F1-Scores of ML baselines. We try the baseline with and without the IP information.

Lastly, in the last row we report the results in a per-flow split (the same as Table 4). Here, packets are naturally and consistently separated into train or test sets, and the model has a hard time finding easy patterns.

In short, the per-packet split introduces dangerous information leaks, allowing the model to find easy patterns that will not be available during deployment. It is crucial to carefully split the data to avoid this issue: A per-flow split helps mitigate the problem.

Ablation study on *Pcap-Encoder*: we perform an ablation study on *Pcap-Encoder* in the per-flow split scenario when (i) removing the IPs, (ii) removing the TCP/IP header entirely, or (iii) removing the application payload. Changes are applied in both the train and test sets, and the encoder is kept frozen during training.

We report these results in Table 7. The performance reduces by removing the IP Addresses, and collapses by removing the entire IP and TCP headers. This is expected given that *Pcap-Encoder* is designed to ignore the payload. In fact, removing the application layer payload has a limited (*VPN-app*) or no impact (*TLS-120* – everything encrypted scenario). By design – and in practice – the encrypted payload cannot make any significant contribution to the classification task.

***Pcap-Encoder* vs Shallow models:** So far, *Pcap-Encoder* has proved to offer the best performance in the realistic per-flow split scenario. But how does it compare with traditional ML approaches?

In Table 8, we compare the performance of some shallow models with that of *Pcap-Encoder*. We report the F1 scores when provided the same input as *Pcap-Encoder* (*base* settings). We manually select which packet header field to use as features for training and testing (details are in Appendix A.3). The results show that the shallow ML models perform better than *Pcap-Encoder*. This happens because *Pcap-Encoder* relies on pre-training tasks to autonomously extract useful features from the raw byte stream, whereas we provide shallow models with custom features extracted from significant protocol fields pre-selected by networking experts.

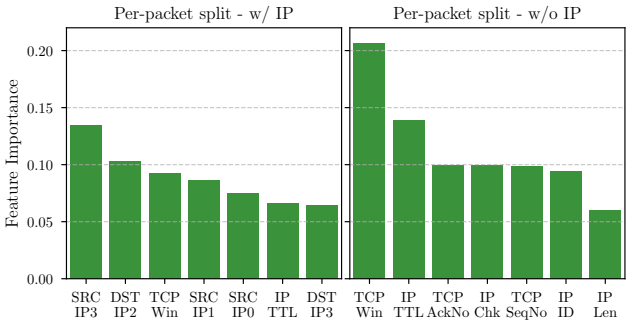


Figure 5: Feature importance for the Random Forest model on the per-packet split for the TLS-120 problem. We both examine the case with and without IP among input features.

Focus on the *w/o IP addr* column in Table 8 that shows what happens when we remove the IP address from the data. The performance of shallow models drops, particularly in the TLS-120 task. With no access to IP addresses, the Random Forest relies on other fields, which overall provide less information. However, even in this configuration, the use of handcrafted features still enables the shallow models to outperform *Pcap-Encoder*.

Shallow Models – Feature Importance: We leverage the feature importance scores provided by the Random Forest to identify the features the model relies on for its decisions. Fig. 5 reports the scores for the TLS-120 problem. We consider the per-packet split scenario to highlight the shortcuts on explicit and implicit flow IDs.

Observe the left plot: this is the base scenario. All headers' features are available for training. IP addresses are explicit flow IDs and the model reaches an accuracy of 98.9%. The most relevant features are the different octets of the source and destination IP addresses (see *SRC IP3*, *DST IP2*, *SRC IP1*, *SRC IP0* and *DST IP3*)[12].

In the right plot, we report results after removing IP addresses from the input features. Here, sequence and acknowledgment numbers – implicit flow identifiers – become the most important features. Despite the removal of explicit identifiers, the model still achieves high performance with an accuracy of 92.6%, reflecting the flaws of the per-packet split setup.

6.2 Flow-Level Traffic Classification Results

At last, given that *YaTC*, *NetMamba*, *TrafficFormer* and *netFound* models were designed for flow representation, we compare how they perform in a flow-level network traffic classification task. We take the same two challenging datasets used in the previous section and keep all flows that have at least 5 packets (the best case for encoders). Only per-flow split is viable in this case. We train the classification model using both frozen and unfrozen encoders using the same 3-fold approach.

When training the models, we follow the original papers' indications. For *netFound*, we select the median burst for each flow and the median packet for each burst based on the distribution of the data sequence lengths, with a maximum of 12 bursts and 6 packets

[12]Recall that we use bi-flows, and the source/destination IP addresses can correspond to both the client and server.

Model (Per-flow split)	VPN-app (16)				TLS-120			
	Frozen		Unfrozen		Frozen		Unfrozen	
	AC	F1	AC	F1	AC	F1	AC	F1
ET-BERT	42.0	38.9	59.2	54.3	20.5	13.8	55.3	51.5
YaTC	25.5	25.1	**60.0**	**54.8**	34.0	27.8	77.3	74.8
NetMamba	15.6	13.6	52.4	48.6	16.9	11.3	78.3	76.0
TrafficFormer	39.2	36.9	53.7	49.2	46.3	42.3	71.4	69.2
netFound	22.9	18.8	56.6	52.4	28.0	22.9	**90.8**	**89.7**
Pcap-Encoder	**69.2**	**62.2**	–	–	**71.3**	**68.1**	–	–

Table 9: Flow-based classification tasks. Only per-flow split is possible. Similarly to the per-packet case, results improve only when the encoder is unfrozen.

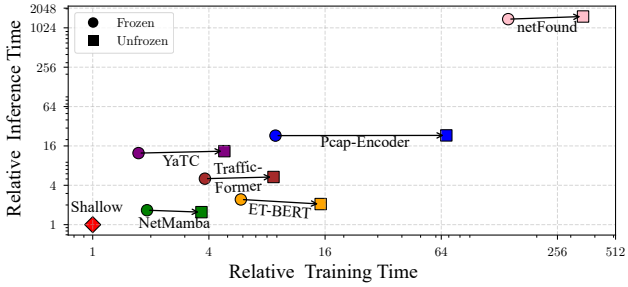


Figure 6: Relative training and inference times. All models are much slower than the shallow baseline, with larger models having the worst ratio (up to 2048× slower at inference).

per burst. For the other models, we choose the first five packets of each flow. For *Pcap-Encoder*, being it a packet-level encoder, we adopt a simple majority vote scheme: Without additional training, we classify the first 5 packets of each flow and directly assign the flow class based on the majority of the labels of these 5 packets. We only use the encoder in a frozen manner.

We adopt the same balanced split for training so that each class has a similar number of samples. This restricts the number of training samples per class to match the class with the fewest samples, stressing the models' few-shot learning capabilities.

We compare the results in Table 9. The same conclusions as in the previous experiment hold: First, all representation learning models struggle to classify the flow when the encoder is frozen. Once more, the representation learned during pre-training fails to effectively capture key features. Performance improves when the encoder is unfrozen and allowed to update all its weights freely. However, even in this case, the results remain comparable to those of *Pcap-Encoder*, which achieves similar performance despite having a frozen encoder and relying on a simple majority-voting scheme. *netFound* emerges as the best classifier in the TLS-120 task. This is due to its complex architecture (see next), and to specific pretext tasks that aim at including specific header fields (e.g., packet size) and flow-level multimodal information (e.g., time interval).

We noticed a large difference concerning the claimed results of *ET-BERT* on *TLS-120* (macro F1-Score of 97.5% [27]). This is due to: (i) the difference balanced strategy we adopt for training, which stresses the few-shot learning properties of the models; and (ii) the possible presence of the TLS Client Hello packet with plain text SNI (the authors do not confirm they removed it while in the public dataset it is not present) that would make the task simpler. We ran an additional experiment with the original unbalanced 8:1:1 per-flow split, and the performance increased accordingly.

6.3 Model efficiency

We compare the inference and training time for the different models with frozen or unfrozen encoders. We measure the time to complete the 3-folds in the *VPN-app*, per-flow split setup. Fig. 6 reports the results normalized to the Random Forest which is the fastest [13]. At training time, all representation learning models require from 2 to 500 times more time than the RF. When comparing frozen versus unfrozen training, the time grows by a 2x - 8x factor, depending on the model. At inference, the efficiency only depends on the complexity of the model. *NetMamba* is the most efficient, while *netFound* is the most costly model as it relies on the BERT Large architecture.

Overall, *Pcap-Encoder* good representation and classification performance is counterbalanced by it being the second slowest models, both at training and inference time. While the resource consumption of *Pcap-Encoder* is higher than that of shallow models, in scenarios where network data is becoming increasingly complex and protocols are becoming more diverse, this self-supervised model can extract information with stronger semantics, allowing it to better adapt to and generalize across different network environments and requirements.

7 Related Works

Traffic classification has been a traditional problem since the Internet's birth. Initially solved by DPI [12], after the adoption of encryption, researchers started using machine learning [39], deep learning [40, 41, 46], and recently representation learning to face it.

Along the way, several works explicitly questioned some approaches and suggested best practices to follow. Regarding the problem of shortcut and 'spurious correlations', Arp et al. [1] studied top-tier security conference papers from the past decade, confirming that pitfalls are widespread and demonstrating how they can lead to unrealistic performance and interpretations. Similarly, Jacobs et al. [21] showed that models trained for Network Security can be shortcut learners, and propose a decision tree-based methodology to explain the model's predictions. More recently, Willinger et al. [52] introduced the *Credibility Crisis* currently impacting ML for networking, and discussed the *underspecification* problem that affects standard supervised ML pipelines. Prior end-to-end works primarily address models trained with standard supervision; here we draw attention to what we call the *sweet danger of sugar*. Self-supervision – and specifically representation learning – represents a methodological advancement over end-to-end supervised learning and it can therefore be a way-out to the credibility problem. However, as we show, poor application of self-supervised methods can lead to pitfalls similar to those found in supervised approaches.

Focusing on one cause of shortcut learning, i.e., data issues, Dainotti et al. [4] provided recommendations, including rigorous data

[13] 22 seconds for training and validation, and 5 seconds for testing.

collection and common benchmark definition. Authors do not explicitly mention the need for proper and careful data splitting. More recently, on the problem of spurious correlations, Flood et al. [13] pinpointed issues in datasets commonly used for cybersecurity that can induce bias in intrusion detection systems. Similarly, Wickramasinghe et al. [51] identified severe limitations in commonly used datasets (e.g., *substantial portions of public datasets contain unencrypted traffic*). We are the first to identify how critical the dataset splitting is and the importance of focusing on frozen training.

Broadening the perspective, numerous studies have identified general pitfalls in machine learning, including shortcut learning [15], sampling bias and dataset shift [35, 44, 47], biased parameter selection [43], cherry picking [36], and flawed training [32] or evaluation practices [14]. Our work builds on this foundation, offering recommendations for the specific challenges of representation learning in traffic classification and supporting the networking community in developing AI-based solutions and shared best practices.

8 Conclusion

In this paper, we presented a detailed analysis of state-of-the-art representation learning works for traffic classification. We highlighted major pitfalls that previous works ignored, likely intoxicated by the 'sugar' of the results falsely close to perfection. After showing that the representation produced by the previous models is not informative, we advocate a frozen testing setup. In fact, *Pcap-Encoder* is the only model that produces a useful representation for downstream traffic classification tasks. However, its complexity and performance, on par with shallow models, question its practicality for current problems.

Our findings offer relevant lessons for two communities: (i) networking researchers applying AI techniques, and (ii) AI researchers developing models for networking problems. Some insights—such as the effects of freezing strategies and dataset splitting—are broadly applicable and may generalize to adjacent fields like cybersecurity. Others, like distinguishing between packet- and flow-level analysis, removing extraneous protocols, and designing networking-specific pre-training tasks, are more domain-specific.

We hope that our work sheds light on the debunking and understanding of the correct usage of AI-based solutions in the context of traffic classification and the networking field.

Ethics

This work does not raise ethical issues.

Acknowledgments

This work has been funded by Huawei Technologies France under the project "AISN – AI Secured Networks: Novel Approaches for Concept-Constrained multi-modal Learning for generalizable task-specific Language Models", which proved fundamental to the genesis of the work. Yuqi Zhao has been supported by the China Scholarship Council (Grant No. 202306470001). Matteo Boffa has been supported by the AI4CTI FISA project #FISA-2023-00168 funded by the Italian Ministry of University and Research (MUR). Marco Mellia has been supported by the project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU. Computational resources were provided by HPC@POLITO (https://hpc.polito.it).

References

[1] Daniel Arp, Erwin Quiring, Feargus Pendlebury, Alexander Warnecke, Fabio Pierazzi, Christian Wressnegger, Lorenzo Cavallaro, and Konrad Rieck. 2022. Dos and don'ts of machine learning in computer security. In *31st USENIX Security Symposium (USENIX Security 22)*. 3971–3988.

[2] Kenjiro Cho, Koushirou Mitsuya, and Akira Kato. 2000. Traffic data repository at the {WIDE} project. In *2000 USENIX Annual Technical Conference (USENIX ATC 00)*.

[3] Noam Chomsky. 1957. *Syntactic Structures*. De Gruyter Mouton, Berlin, Boston.

[4] Alberto Dainotti, Antonio Pescape, and Kimberly C Claffy. 2012. Issues and future directions in traffic classification. *IEEE network* 26, 1 (2012), 35–40.

[5] Ferdinand De Saussure. 2004. Course in general linguistics. *Literary theory: An anthology* 2 (2004), 59–71.

[6] Giovanni Dettori. 2024. *Designing and engineering a Q&A LLM for network packet representation*. MSc thesis. Politecnico di Torino. Available at https://webthesis.biblio.polito.it/33158/.

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[8] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *International Conference on Learning Representations*.

[9] Gerard Draper-Gil, Arash Habibi Lashkari, Mohammad Saiful Islam Mamun, and Ali A. Ghorbani. 2016. Characterization of Encrypted and VPN Traffic using Time-related Features. In *International Conference on Information Systems Security and Privacy*.

[10] Nick Erickson, Jonas Mueller, Alexander Shirkov, Hang Zhang, Pedro Larroy, Mu Li, and Alexander Smola. 2020. AutoGluon-Tabular: Robust and Accurate AutoML for Structured Data. *arXiv preprint arXiv:2003.06505* (2020).

[11] Amir Feder, Katherine A Keith, Emaad Manzoor, Reid Pryzant, Dhanya Sridhar, Zach Wood-Doughty, Jacob Eisenstein, Justin Grimmer, Roi Reichart, Margaret E Roberts, et al. 2022. Causal inference in natural language processing: Estimation, prediction, interpretation and beyond. *Transactions of the Association for Computational Linguistics* 10 (2022), 1138–1158.

[12] Michael Finsterbusch, Chris Richter, Eduardo Rocha, Jean-Alexander Muller, and Klaus Hanssgen. 2013. A survey of payload-based traffic classification approaches. *IEEE Communications Surveys & Tutorials* 16, 2 (2013), 1135–1156.

[13] Robert Flood, Gints Engelen, David Aspinall, and Lieven Desmet. 2024. Bad design smells in benchmark nids datasets. In *2024 IEEE 9th European Symposium on Security and Privacy (EuroS&P)*. IEEE, 658–675.

[14] George Forman and Martin Scholz. 2010. Apples-to-apples in cross-validation studies: pitfalls in classifier performance measurement. *Acm Sigkdd Explorations Newsletter* 12, 1 (2010), 49–57.

[15] Robert Geirhos, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard Zemel, Wieland Brendel, Matthias Bethge, and Felix A Wichmann. 2020. Shortcut learning in deep neural networks. *Nature Machine Intelligence* 2, 11 (2020), 665–673.

[16] Albert Gu and Tri Dao. 2023. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752* (2023).

[17] Satyandra Guthula, Roman Beltiukov, Navya Battula, Wenbo Guo, and Arpit Gupta. 2023. netFound: Foundation model for network security. *arXiv preprint arXiv:2310.17025* (2023).

[18] Hong Ye He, Zhi Guo Yang, and Xiang Ning Chen. 2020. PERT: Payload encoding representation from transformer for encrypted traffic classification. In *2020 ITU Kaleidoscope: Industry-Driven Digital Transformation (ITU K)*. IEEE, 1–8.

[19] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. 2022. Masked autoencoders are scalable vision learners. In *2022 IEEE/CVF conference on computer vision and pattern recognition*. 16000–16009.

[20] Jeremy Howard and Sebastian Ruder. 2018. Universal language model fine-tuning for text classification. *arXiv preprint arXiv:1801.06146* (2018).

[21] Arthur S Jacobs, Roman Beltiukov, Walter Willinger, Ronaldo A Ferreira, Arpit Gupta, and Lisandro Z Granville. 2022. Ai/ml for network security: The emperor has no clothes. In *2022 ACM SIGSAC Conference on Computer and Communications Security*. 1537–1551.

[22] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361* (2020).

[23] Will Knight. 2023. Google's Gemini Is The Real Start of the Generative AI Boom. https://www.wired.com/story/google-gemini-generative-ai-boom/.

[24] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems* 25 (2012).

[25] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942* (2019).
[26] Dongze Lian, Daquan Zhou, Jiashi Feng, and Xinchao Wang. 2022. Scaling & shifting your features: A new baseline for efficient model tuning. *Advances in Neural Information Processing Systems* 35 (2022), 109–123.
[27] Xinjie Lin, Gang Xiong, Gaopeng Gou, Zhen Li, Junzheng Shi, and Jing Yu. 2022. Et-bert: A contextualized datagram representation with pre-training transformers for encrypted traffic classification. In *2022 ACM Web Conference*. 633–642.
[28] Shayne Longpre, Le Hou, Tu Vu, Albert Webson, Hyung Won Chung, Yi Tay, Denny Zhou, Quoc V Le, Barret Zoph, Jason Wei, et al. 2023. The flan collection: Designing data and methods for effective instruction tuning. In *International Conference on Machine Learning*. PMLR, 22631–22648.
[29] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. 2015. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025* (2015).
[30] Ben Mann, Nick Ryder, Melanie Subbiah, J Kaplan, P Dhariwal, A Neelakantan, P Shyam, G Sastry, A Askell, S Agarwal, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.
[31] David Marr. 2010. *Vision: A computational investigation into the human representation and processing of visual information*. MIT press.
[32] R. Thomas McCoy, Ellie Pavlick, and Tal Linzen. 2019. Right for the Wrong Reasons: Diagnosing Syntactic Heuristics in Natural Language Inference. In *Annual Meeting of the Association for Computational Linguistics*.
[33] Xuying Meng, Yequan Wang, Runxin Ma, Haitong Luo, Xiang Li, and Yujun Zhang. 2022. Packet representation learning for traffic classification. In *2022 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 3546–3554.
[34] Sam Meredith. 2023. A 'thirsty' generative AI boom poses a growing problem for Big Tech. https://www.cnbc.com/2023/12/06/water-why-a-thirsty-generative-ai-boom-poses-a-problem-for-big-tech.html.
[35] Jose G Moreno-Torres, Troy Raeder, Rocío Alaiz-Rodríguez, Nitesh V Chawla, and Francisco Herrera. 2012. A unifying view on dataset shift in classification. *Pattern recognition* 45, 1 (2012), 521–530.
[36] Janice M Morse. 2010. "Cherry picking": Writing from thin data. 3–3 pages.
[37] Nour Moustafa and Jill Slay. 2015. UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In *2015 military communications and information systems conference (MilCIS)*. IEEE, 1–6.
[38] David Naylor, Alessandro Finamore, Ilias Leontiadis, Yan Grunenberger, Marco Mellia, Maurizio Munafò, Konstantina Papagiannaki, and Peter Steenkiste. 2014. The cost of the" s" in https. In *2014 10th ACM International on Conference on Emerging Networking Experiments and Technologies*. 133–140.
[39] Thuy TT Nguyen and Grenville Armitage. 2008. A survey of techniques for internet traffic classification using machine learning. *IEEE communications surveys & tutorials* 10, 4 (2008), 56–76.
[40] Fannia Pacheco, Ernesto Exposito, Mathieu Gineste, Cedric Baudoin, and Jose Aguilar. 2018. Towards the deployment of machine learning solutions in network traffic classification: A systematic survey. *IEEE Communications Surveys & Tutorials* 21, 2 (2018), 1988–2014.
[41] Eva Papadogiannaki and Sotiris Ioannidis. 2021. A survey on encrypted network traffic analysis applications, techniques, and countermeasures. *ACM Computing Surveys (CSUR)* 54, 6 (2021), 1–35.
[42] Lingfeng Peng, Xiaohui Xie, Sijiang Huang, Ziyi Wang, and Yong Cui. 2024. PTU: Pre-trained Model for Network Traffic Understanding. In *2024 32nd IEEE International Conference on Network Protocols (ICNP)*.
[43] Jing Qin. 2017. *Biased sampling, over-identified parameter problems and beyond*. Vol. 5. Springer.
[44] Joaquin Quiñonero-Candela, Masashi Sugiyama, Anton Schwaighofer, and Neil D Lawrence. 2022. *Dataset shift in machine learning*. Mit Press.
[45] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research* 21, 140 (2020), 1–67.
[46] Shahbaz Rezaei and Xin Liu. 2019. Deep learning for encrypted traffic classification: An overview. *IEEE communications magazine* 57, 5 (2019), 76–81.
[47] Tatiana Tommasi, Novi Patricia, Barbara Caputo, and Tinne Tuytelaars. 2017. A deeper look at dataset bias. *Domain adaptation in computer vision applications* (2017), 37–55.
[48] Martino Trevisan, Danilo Giordano, Idilio Drago, Marco Mellia, and Maurizio Munafo. 2020. Five Years at the Edge: Watching Internet From the ISP Network. *IEEE/ACM Transactions on Networking* 28, 02 (2020), 561–574.
[49] Tongze Wang, Xiaohui Xie, Wenduo Wang, Chuyi Wang, Youjian Zhao, and Yong Cui. 2024. NetMamba: Efficient Network Traffic Classification via Pre-training Unidirectional Mamba. In *2024 32nd IEEE International Conference on Network Protocols (ICNP)*.
[50] Wei Wang, Ming Zhu, Xuewen Zeng, Xiaozhou Ye, and Yiqiang Sheng. 2017. Malware traffic classification using convolutional neural network for representation learning. In *2017 International Conference on Information Networking (ICOIN)*. 712–717.
[51] Nimesha Wickramasinghe, Arash Shaghaghi, Gene Tsudik, and Sanjay Jha. 2025. SoK: Decoding the Enigma of Encrypted Network Traffic Classifiers. In *2025 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1825–1843.
[52] Walter Willinger, Ronaldo A Ferreira, Arpit Gupta, Roman Beltiukov, Satyandra Guthula, Lisandro Z Granville, and Arthur S Jacobs. 2025. When Something Looks Too Good To Be True, It Usually Is! AI Is Causing A Credibility Crisis In Networking. *ACM SIGCOMM Computer Communication Review* 55, 1 (2025), 10–15.
[53] Ruijie Zhao, Mingwei Zhan, Xianwen Deng, Yanhao Wang, Yijun Wang, Guan Gui, and Zhi Xue. 2023. Yet another traffic classifier: A masked autoencoder based traffic transformer with multi-level flow representation. In *2023 37th AAAI Conference on Artificial Intelligence*, Vol. 37. 5420–5427.
[54] Guangmeng Zhou, Xiongwen Guo, Zhuotao Liu, Tong Li, Qi Li, and Ke Xu. 2024. TrafficFormer: An Efficient Pre-trained Model for Traffic Data. In *2025 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 102–102.

A Appendix

Appendices are supporting material that has not been peer-reviewed.

A.1 *Pcap-Encoder* details

A.1.1 Packet representation learning task formalization. We use the same problem formalization as the one in [33]. Consider a packet set $\mathcal{X} = \{\boldsymbol{x_1}, \boldsymbol{x_2}, ..., \boldsymbol{x_n}\}$ where $\boldsymbol{x_i}$ is the i-th packet, and each packet $\boldsymbol{x_i}$ is represented as an input sequence $\boldsymbol{x_i} = \{t_{i,1}, t_{i,2}, ..., t_{i,n}\}$ where $t_{i,j}$ is the j-th token derived from the split of the tokenizer. The length of the vector $\boldsymbol{x_i}$ can vary since the size of packets is unfixed. In addition, each packet $\boldsymbol{x_i}$ can assume a number n of labels $\boldsymbol{y_i}$ depending on the number of downstream tasks we want to solve. So, for a downstream classification task j, $y_{i,j} \in C_j$.

To obtain the classification from the packet, we need the latent vector (packet representation) $\boldsymbol{r_i} \in \mathbb{R}^d$ where d is the hidden dimension of the model. The latent vector is obtained by combining the columns of the latent matrix $\boldsymbol{H_i} \in \mathbb{R}^{d \times L}$ where L is the number of tokens in each packet and d the dimension of a single token.

The formalization of the packet representation learning task becomes: Given the input $\mathcal{X}$ and the corresponding label set $\mathcal{Y}$ of multiple classification tasks, the goal is to learn a single packet representation encoder $f : \boldsymbol{x_i} \rightarrow \boldsymbol{r_i}$ that obtain accurate $\boldsymbol{y_i}$ on downstream tasks by a function $g : \boldsymbol{r_i} \rightarrow \boldsymbol{y_i}$.

A.1.2 Bottleneck. T5 provides a single representation of size 768 for each of the L tokens of the packet. Let's call $\boldsymbol{e_{i,j}}$ the representation of token j for packet i. However, we want to obtain a single representation $\boldsymbol{r_i}$ of dimension 768 for the entire packet $\boldsymbol{x_i}$ from the L representations of the tokens. Hence, we need to use an aggregator or bottleneck. This dimensionality reduction process necessarily discards a part of the information set. We tried different architectures:

(1) **First pooling**: a dummy solution that takes as packet representation the embedding of the first token, that is always the initial part of the question needed by T5. The representation vector of packet i becomes:

$$\boldsymbol{r_i} = \boldsymbol{e_{i,0}}$$

(2) **Mean pooling**: performs the average over the hidden vectors $\boldsymbol{e_{i,j}}$ of the hidden matrix. The representation vector of packet i becomes:

$$\boldsymbol{r_i} = \frac{\sum_{j=1}^{L} \boldsymbol{e_{i,j}}}{L}$$

	Questions on Packets
Retrieval	*Which is the TCP checksum?* *Which is the destination IPv4/IPv6 of the packet?* *Which is the source IPv4/IPv6 of the packet?* *Which is the id of IPv4/IPv6?* *Which is the time to live of IPv4/IPv6?*
Computational	*Is the packet's IPv4/IPv6 checksum correct?* *Which is the last byte of the header in the third layer?* *Which is the length of the payload in the third layer?*

Table 10: Retrieval and computational questions for the Q&A pre-training task.

(3) **Luong attention** [29]: performs a weighted average of the embeddings. The weights, computed for each $e_{i,j}$, must be positive and the sum is 1. The representation vector of one packet becomes:

$$w_j = \frac{\exp\left(\boldsymbol{e_j}^\top \boldsymbol{q}\right)}{\sum_{z=1}^{L} \exp\left(\boldsymbol{e_z}^\top \boldsymbol{q}\right)} \qquad \boldsymbol{r_i} = \sum_{j=1}^{L} w_j \boldsymbol{e_j}$$

where $\boldsymbol{q}$ is a learnable query vector and w_j is the weight associated with the embedding vector $\boldsymbol{e_j}$.

The bottleneck is part of the trained model T5 (encoder+decoder). So, even if the bottleneck is very simple, the underlying layers can adjust their weights to create a meaningful representation. Therefore, having a computationally expensive bottleneck is redundant.

A.1.3 Question answering dataset and results. We created a dataset with multiple tasks for the Q&A phase starting from the datasets already described in Subsection 3.4.

Table 10 shows the 8 questions selected in the Q&A dataset. Some of the questions are retrieval tasks that need to find the answer in the context. Others consist of more complex tasks, such as the computation of the checksum, or the payload length.

On this question dataset, we obtain an average accuracy of 98.2% on the test, averaging over different tasks.

A.1.4 Ablation study on Pcap-Encoder pre-training. Our ablation study examined the impact of different components in *Pcap-Encoder* pipeline across the two tasks VPN-App (16) and TLS (120). Table 11 shows a clear performance hierarchy starting from the highest with the complete model. Removing the autoencoder component led to a moderate decrease in performance, particularly noticeable in the TLS task with an accuracy drop of ≈ 8%. Most strikingly, using only the base T5 model without any pre-training resulted in severely degraded performance, especially on the more complex TLS dataset where the accuracy plummeted to 8.5. These results strongly suggest that both pre-training components contribute meaningfully to the model's effectiveness, with the Q&A component appearing to be particularly crucial for maintaining strong performance

A.2 Models Hyperparameters

In the following, we report the chosen hyperparameters for the seven models under analysis:

• *ET-BERT*: We remove the Ethernet and IP header and TCP ports. We set the learning rate to $2 \cdot 10^{-5}$ with 20 epochs for fine-tuning the unfrozen model, and $2 \cdot 10^{-3}$ for 60 epochs for the frozen model.

Model (Flow Split)	VPN-app (16)		TLS (120)	
	AC	F1	AC	F1
Autoencoder + Q&A	83.5	71.0	71.0	63.7
Q&A only	82.6	72.1	63.6	57.2
T5-base	54.5	39.8	8.5	2.5

Table 11: Results on the per-Flow Split scenario by freezing and removing the pre-training phases of *Pcap-Encoder*.

Protocol	Packet fields
IPv4	*Source and Destination addresses, Type of service, Internet Header Length, ID, Checksum, Flags, Length, Protocol, Version, TTL, Fragmentation*
IPv6	*Source and Destination addresses, Flow label, Version, Payload Length, Hop Limit, Traffic Class, Next Header*
UDP	*Source and Destination ports, Checksum, Length*
TCP	*Source and Destination ports, Timestamp, Window, Urgent pointer, Data offset, Flags, Checksum, Sequence and Acknowledgment numbers, Options*

Table 12: Packet fields selected for the Shallow model training. Features are extracted from raw traces using the Python Scapy package (https://scapy.net).

• *YaTC*: We anonymize IP addresses and ports, and group the first five packets to construct the input matrix by padding or truncating packets if necessary, as in the original paper. We set the learning rate at $2 \cdot 10^{-3}$ and the batch size to 64 over 200 epochs, both for the frozen and unfrozen tests.

• *NetMamba*: We use the same learning rate and data processed as *YaTC*.

• *TrafficFormer*: We randomize the IP address and TCP ports and follow the same training-set augmentation proposed in the paper. We set the learning rate to $2 \cdot 10^{-5}$ with 20 epochs for fine-tuning the unfrozen model, and $1 \cdot 10^{-4}$ for 60 epochs for the frozen model. At the same time, we set an early stop of 5 epochs, that is, if the model performance does not improve in 5 consecutive epochs, the model will be terminated early.

• *netFound*: We generate tokens and extract flow metadata as in the original paper. We set the learning rate to $2.5 \cdot 10^{-6}$ for a single GPU or $1 \cdot 10^{-5}$ for four GPUs and fine-tune the unfrozen and frozen models for 100 epochs with early stopping for 6 epochs.

• *Pcap-Encoder*: We load the T5-base model with pre-trained weights. For the encoder adaptation, we use the AdamW optimizer, a learning rate of $5 \cdot 10^{-4}$, a linear rate scaling, a batch size of 8, and we train for 15 epochs. For question-answering tasks, we keep the same learning rate and scaling. We train for 20 epochs with a batch size of 24.

• *Shallow model*: We use 4 different ML models (i.e., Random Forest, XGBoost, MLP, LightGBM) and we use AutoGloun [10] to automatically select the best hyperparameters. As features, we construct a vector for each packet by extracting the values of selected protocol fields (padding missing fields).

Type	Protocols	ISCX-VPN	USTC-TFC	CSTN-TLS1.3
link-local protocols	llmnr, nbns, mdns, lsd	922347 (3.45%)	227413 (3.93%)	0
network management protocols	icmp, icmpv6, dhcp, dhcpv6, igmp, snmp, arp, cops	437020 (1.63%)	373641 (6.46%)	0
nat protocols	nat-pmp, rsip, stun	205660 (0.77%)	68 (<0.01%)	0
route management protocols	db-lsp, db-lsp-disc, pathport, stp, bfd echo, bgp, ecmp, asap	21914 (0.08%)	656 (0.01%)	0
service management protocols	ssdp, lldp, srvloc, opa, cbsp	8696 (0.03%)	3812 (0.07%)	0
real time protocols	rtcp	2763 (<0.01%)	0	0
network time protocols	ntp	2386 (<0.01%)	35 (<0.01%)	0
link management protocols	llc, ipxsap	1582 (<0.01%)	0	0
distributed protocols	thrift, dcerpc, rmi	182 (<0.01%)	5 (<0.01%)	0
security protocols	ocsp, pkix-cert, egd, chargen, tpm, knet	170 (<0.01%)	134 (<0.01%)	0
industrial protocols	r-goose, dcp-pft, dcp-af, vicp nxp 802154 sniffer, enip, c1222, ax4000	107 (<0.01%)	61 (<0.01%)	0
remote access protocols	vnc, x11, msnms	75 (<0.01%)	6 (<0.01%)	0
file protocols	lanman, bjnp, spoolss, ndps, laplink, bzr, cvspserver	62 (<0.01%)	129 (<0.01%)	0
quake protocols	quake, quake2, quake3, quakeworld	0	4 (<0.01%)	0
mobile protocols	gsm, ipa, gtp	0	18 (<0.01%)	0
iot management protocols	bat.vis, tplink-smarthome, coap,mqtt	0	11 (<0.01%)	10 (<0.01%)
others protocols	tds, bitcoin	0	7 (<0.01%)	0

Table 13: The protocols we filter and number and percentage (in parenthesis) of removed packets for each dataset.

A.3 Shallow models details

Table 12 lists the fields selected for the training of shallow models. The specific fields vary across datasets based on the presence or absence of the different datalink and transport protocols.

A.4 Filter Details

In Table 13 we list all protocols we filter using Tshark filters. We detail the total number of packets each filter removes from each trace. The main protocols involved are closely related to network management, link-local communication (link-local), and NAT (especially STUN). The CSTN trace was already cleaned, while the other contains from 5 to 10% of unrelated protocols.