Computer Networks 280 (2026) 112090

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

A multimodal and perturbation-aware learning approach for robust traffic classification

Idio Guarino [a,*], Giampaolo Bovenzi [b], Alfredo Nascita [b], Domenico Ciuonzo [b], Damiano Carra [a], Antonio Pescapè [b]

[a] University of Verona, Italy
[b] University of Napoli Federico II, Italy

ARTICLE INFO

Keywords:
Deep learning
Encrypted traffic
Multimodal techniques
Robust design
Traffic classification

ABSTRACT

Traffic Classification (TC) is pivotal for network management, cybersecurity, and Quality of Experience (QoE) monitoring. However, while Deep Learning (DL) has significantly advanced TC, most existing works assume static, idealized conditions, overlooking key challenges of real-world deployments—such as traffic variability, routing asymmetries, out-of-order packet arrivals, and partial visibility at the Vantage Points (VPs). This motivates the need for robustness evaluations under such scenarios.

In this work, we investigate the robustness of state-of-the-art (SOTA) TC models under realistic, yet controlled, perturbation scenarios. Specifically, we introduce novel, model-agnostic traffic perturbations—simulating time jitter, retransmissions, and partial visibility—to reflect conditions commonly encountered in live network traffic. We evaluate our approach on three public datasets—i.e., VPN-16, MIRAGE-19, and MIRAGE-24—and show how MIMETIC-ENHANCED, a multimodal model, tends to outperform two representative single-modal counterparts both in terms of TC effectiveness on clean traffic and robustness under perturbations. Nonetheless, our analysis also reveals that multimodal models remain vulnerable under specific perturbation settings. To address this limitation, we propose a *model-agnostic perturbation-aware training framework* based on Supervised Data Augmentation (Aug) and Contrastive Learning (CL)—considering both self-supervised and supervised variants. Unlike architecture-specific solutions, our approach operates at the *learning strategy level*, allowing it to be seamlessly applied to diverse classifiers without requiring structural modifications. Adopting MIMETIC-ENHANCED as a primary multimodal case study, we integrate the proposed strategies into its two-stage training pipeline. Experimental results demonstrate that perturbation-aware training not only improves TC effectiveness on clean (i.e., unperturbed) traffic—particularly when applied across both training stages—but also significantly strengthens the model's robustness under diverse and realistic perturbation scenarios. Furthermore, we investigate Out-of-Distribution (OOD) detection, model calibration, and TC effectiveness in low-data regimes. Finally, we explicitly demonstrate the framework's *generalizability* by validating it on other SOTA architectures, spanning both single- and multi-modal approaches.

1. Introduction

TC plays a central role in network operations, supporting traffic monitoring and management, cybersecurity enforcement, and QoE monitoring. The field is well-established, with a rich body of research and numerous comprehensive surveys, e.g., [1,2].

Over the years, TC has evolved from traditional Machine Learning (ML) with handcrafted features to DL models that extract patterns directly from raw network data. This shift unfolded in two waves [3]. The first wave, starting in the early 2000s, used ML with per-packet (e.g., packet size, inter-arrival time) and per-flow statistics (e.g., total bytes, number of packets, ports) to classify a small set of applications–often with high accuracy and deployability (e.g., supporting early classification by relying only on a few initial packets) [4–7]. However, these methods struggled to keep pace with rising traffic complexity driven by (*a*) encryption, (*b*) proliferation of mobile devices, and (*c*) diversity of the tasks/activities performed by services running over the Internet [8,9]. The second wave has embraced DL, leveraging raw payloads or statistical features in end-to-end architectures to model traffic in an "expert-free" fashion [8,10–14]. In this context, more sophisticated

* Corresponding author.
E-mail addresses: idio.guarino@univr.it (I. Guarino), giampaolo.bovenzi@unina.it (G. Bovenzi), alfredo.nascita@unina.it (A. Nascita), domenico.ciuonzo@unina.it (D. Ciuonzo), damiano.carra@univr.it (D. Carra), pescape@unina.it (A. Pescapè).

https://doi.org/10.1016/j.comnet.2026.112090
Received 29 July 2025; Received in revised form 30 January 2026; Accepted 4 February 2026
Available online 25 February 2026

efforts have explored *multimodal DL architectures* [9,15,16], using different (but complementary) facets of traffic data, such as packet payloads and statistical features developed in the first wave [8,10–14]. Indeed, recent literature demonstrates that combining heterogeneous views effectively tackles challenging operational conditions, such as mitigating the impact of noisy annotations [17].

Yet, a key limitation remains: *most SOTA models are trained and tested in static, closed-world settings with clean data from limited VPs*. In contrast, real-world deployments—especially at intermediate network locations such as ISP routers or enterprise middleboxes—are characterized by noisy measurements, asymmetric flows, and partial visibility. This gap highlights *the need to evaluate and improve model robustness* [3] under realistic (perturbed) conditions–especially for encrypted traffic.

Accordingly, this paper tackles a critical shortcoming in today's traffic classifiers: *their fragility under real-world, dynamic network conditions*. We propose a framework to systematically evaluate model robustness by introducing "black-box" perturbations—transformations applied directly to traffic traces that simulate common network phenomena such as jitter, retransmissions, asymmetric paths, and delayed monitoring. These perturbations are model-agnostic and reflect challenges faced by classifiers operating beyond the client edge, where access to clean data and model internals is precluded. Our emphasis is on *early* TC [4], where decisions must be made based on the initial packets of a flow—this is crucial for maximizing the reactivity of, and minimizing the latency of, network management tools [4].

To improve robustness under such conditions, we thoroughly explore *perturbation-aware training*. These techniques help the model learn invariant representations that are resilient to timing distortions and visibility gaps. While our experiments use MIMETIC-ENHANCED [16]—a multimodal traffic classifier that fuses packet sequences and payloads–the proposed training pipeline *is broadly applicable to any multimodal TC model*. Accordingly, the **contributions** of this work are:

- We propose a taxonomy of *realistic network perturbations*—spanning timing, retransmissions, directional asymmetries, and changes in VP—and their systematic application to evaluate the robustness of DL-based encrypted traffic classifiers *without relying on model internals*.
- We systematically benchmark the *robustness* of SOTA multimodal TC models, investigating their susceptibility to network-induced perturbations. While these models perform well on clean data [16], we investigate the potential benefit of multimodality (i.e., the fusion of packet-sequence features and payload content) in close-to-real-world TC. By analyzing how different input modalities respond to these conditions, we assess whether (and to what extent) multimodal DL can improve resilience by compensating for modality-specific weaknesses.
- Using MIMETIC-ENHANCED as a primary case of study, we investigate *perturbation-aware* training pipelines by injecting realistic traffic transformations into the learning process. Our approach integrates *both* `Aug` and `CL`—spanning self-supervised and supervised variants—into the *pre-training* and *fine-tuning* stages of model training. This design helps models learn robust representations, which are essential in real-world network environments.
- We assess whether traffic transformations can be leveraged not only to improve model robustness but also to enhance generalization across network conditions. In this respect, we extend the evaluation beyond accuracy—analyzing latent space structure, model calibration, out-of-distribution detection, and sample-scarcity conditions—to capture TC effectiveness in practical deployments.
- Finally, we demonstrate that the proposed learning framework is effectively *model-agnostic*. By explicitly validating its *broad generalizability* on representative classifiers—spanning both single-modal [11,13] and multimodal [15,18] paradigms—we prove that it enhances robustness regardless of the specific model design.

Table 1
List of acronyms and abbreviations in alphabetical order.

Acronym	Definition
AI	Artificial Intelligence
Aug	Supervised Data Augmentation
AUROC	Area Under the Receiver Operating Characteristic Curve
CCE	Categorical Cross-Entropy
CL	Contrastive Learning
DBI	Davies-Bouldin Index
DHiding	Direction Hiding
DHidingloss	Direction Hiding with Packet Loss
DIR	Packet Direction
DL	Deep Learning
ECE	Expected Calibration Error
FPR	False Positive Rate
FT	Fine-Tuning
GenAI	Generative Artificial Intelligence
IAT	Inter Arrival Time
ID	In-Distribution
LSTM	Long Short-Term Memory
ML	Machine Learning
NT-Xent	Normalized Temperature-scaled Cross-Entropy
OOD	Out-of-Distribution
PAY	initial raw bytes transmitted by the application
PHiding	Packets Hiding
PL	Transport-level Payload Length
PS	Packet Size
PSQ	sequence of header fields from the initial packets
PT	Pre-Training
QoE	Quality of Experience
RTO	Retransmission Timeout
RTT	Round-Trip Time
SDU	Service Data Unit
SIL	Silhouette score
SimCon	Self-supervised Contrastive Learning
SOTA	state-of-the-art
SupCon	Supervised Contrastive Learning
TC	Traffic Classification
TCP	Transmission Control Protocol
TCPWIN	TCP Window Size
TJitter	Directional Time Jittering
TLS	Transport Layer Security
TO	Traffic Object
TPR	True Positive Rate
VAE	Variational Autoencoder
VP	Vantage Point
VPC	Vantage Point Centralization
VPCloss	Vantage Point Centralization with Packet Loss
XAI	eXplainable Artificial Intelligence

Our study is validated on *three public datasets*: VPN-16 [19], MIRAGE-19 [8], and MIRAGE-24 [20], showing practical relevance and broad applicability.

The remainder of this paper is organized as follows. Section 2 reviews related work on TC robustness. Section 3 introduces our multimodal architecture, defines the proposed perturbations, and details the perturbation-aware learning methodologies. Section 4 outlines the experimental setup. Results are discussed in Section 5, and conclusions follow in Section 6.

Tables 1 and 2 list the acronyms and symbols used in this paper, together with their definitions, to aid readability.

2. Related works

Recent advances in TC have applied DL to curated datasets, yielding high accuracy under *static, idealized conditions*. Yet these models often assume full visibility, balanced classes, and stable patterns—assumptions that break down in real-world deployments, especially with encrypted or incomplete traffic. This section surveys how prior work addresses robustness to input perturbations, focusing on how TC models respond to deviations from ideal conditions. Table 3 summarizes the landscape, also detailing the type of **Traffic Object (TO)** used (e.g., flows vs. biflows),

Table 2
List of symbols and their definition.

Symbol	Definition
α	RTT fraction attributed to Client-to-VP path
β	Temperature parameter for contrastive loss
$\mathcal{B}$	Mini-batch of samples
$C \rightarrow S$	Client-to-Server traffic direction
$C \leftarrow S$	Server-to-Client traffic direction
δ	Percentage of retained training data (Data-Constrained)
e	Embedding dimension size ($e = 10$)
$\mathcal{F}_\phi(\cdot)$	Fusion function parameterized by ϕ
$f_\theta(\cdot)$	Feature extractor (backbone) parameterized by θ
$f_\varphi(\cdot)$	Classifier head parameterized by φ
$\mathcal{L}$	Generic loss function (e.g., $\mathcal{L}_{CCE}$)
LP	Number of lost initial packets
$L_{\min}, L_{\max}$	Minimum and maximum number of retransmissions
M	Number of modalities ($M = 2$)
N	Total number of samples in the dataset
N_b	Number of initial bytes in PAY ($N_b = 576$)
N_p	Number of initial packets in PSQ ($N_p = 10$)
p	Packet loss probability
$p(x)$	Soft-output probability distribution
$p(\hat{x})$	Predicted confidence
r	Retransmission rate
$sim(\cdot)$	Similarity function (e.g., cosine similarity)
σ_d	Standard deviation for directional time jitter
$\mathcal{T}$	Predefined set of transformations
T_{out}	Retransmission Timeout duration
$\tau(\cdot)$	A single transformation function sampled from $\mathcal{T}$
X	Set of original training samples $\{x_i\}_{i=1}^N$
$\hat{X}$	Set of transformed training samples
$\overline{X}$	Final augmented training set ($X \cup \hat{X}$)
x	Generic input sample (biflow)
$\tilde{x}$	Transformed input sample
$y, \hat{y}$	True class label and predicted label
z	Latent feature vector ($z = f_\theta(x)$)
$\emptyset$	Empty set (indicating no transformations, $\mathcal{T} = \emptyset$)
$\uparrow, \downarrow$	Direction of improvement (Higher/Lower is better)

Input type(s) adopted (e.g., raw bytes, packet sequences), **Datasets**, and TC **Task** considered.

We categorize prior efforts by the **Perturbation Type**: Section 2.1 covers methods that introduce deliberate, adversarial modifications, while Section 2.2 focuses on techniques that simulate network fluctuations or changes in operating conditions. Finally, Section 2.3 positions our work with respect to the existing state of the art.

2.1. Robustness to intentional manipulations

A first line of research investigates **intentional perturbations** ("I" in column **Perturbation Type**) and applies adversarial ML [39] with the goal of degrading model performance through carefully crafted input perturbations. These adversarial examples exploit the vulnerability of Artificial Intelligence (AI) models to small input manipulations that can lead to incorrect predictions. Adversarial attacks are typically categorized based on the attacker's knowledge of the "target" model, typically falling into *three categories*: (*i*) *white-box* attacks assume full access to the model's architecture and gradients; (*ii*) *black-box* attacks operate without internal model access, relying only on observed outputs; (*iii*) *hybrid* attacks exploit access to a "surrogate" model during the perturbation generation phase and apply the resulting perturbations in a model-agnostic fashion.

In the *white-box* setting, Sadeghzadeh et al. [22] craft universal perturbations (named "pad", "payload", and "burst" attacks) targeting payloads and packet sequences, while Jin and Apostolaki [36] *PANTS* framework uses satisfiability modulo theory solvers to generate valid yet misleading inputs.

Black-box strategies—e.g., *Tantra* [23] and time-based evasion Zhang et al. [24]—reshape temporal patterns to evade malicious traffic detection without altering original content. *Prism* approach [27] goes further by injecting asymmetric perturbations that mask traffic signatures. Such perturbations are driven by the temporal transitions of the application behaviour.

Other works adopt a *hybrid strategy* where adversarial perturbations are generated with white-box access to a surrogate model and then directly reused against the target model under black-box conditions. Lu et al. [37] introduce *TSAF*, a framework that uses white-box access to craft time series-based adversarial perturbations via gradient optimization. Then, these are stored as generic patterns and injected into clean traffic at runtime through a masking mechanism. Ding et al. [31] propose a method for generating universal, transferable adversarial perturbations by optimizing a loss function based on internal network activations. Crafted offline with white-box access, these perturbations generalize across models and datasets, enabling efficient, model-agnostic attacks with high success rates and low overhead. Xie et al. [28] introduce *Cactus*, a client-side defense mechanism that applies protocol-compliant perturbations–modifying packet size, timing, and direction–to obfuscate encrypted Transmission Control Protocol (TCP) traffic. Although not tailored to a specific model, these transformations are intentionally designed to disrupt DL-based traffic analysis, effectively reducing classification accuracy while preserving application semantics.

Collectively, these studies highlight the fragility of traffic classifiers when exposed to adversarial manipulations and motivate the need for robust training techniques. Works such as [22,28,31,37] primarily assess the robustness of state-of-the-art classifiers, revealing their susceptibility to carefully crafted perturbations, even when such perturbations are limited or protocol-compliant (column **Base Classifier Robustness**). Other contributions, including [23,24,27,36], extend the analysis by proposing enhanced models or defense mechanisms and evaluating their behavior under adversarial conditions (i.e., perturbed inputs). They evaluate how these strengthened designs behave when exposed to perturbed inputs (subcolumn ◐), thus providing a more complete picture of robustness under attack. These efforts offer a more holistic view of robustness, combining attack scenarios with mitigation strategies. Still, *adversarial approaches differ in goals and assumptions from studies focused on natural, deployment-induced perturbations—a gap this paper aims to bridge*.

2.2. Robustness to network changes

Beyond adversarial attacks, a substantial line of work examines how network traffic classifiers respond to **non-adversarial perturbations** ("NC" in column **Perturbation Type**)—i.e., variations that *occur naturally in operational settings*. These perturbations, while not malicious, can degrade classification performance or, conversely, serve as a form of data augmentation to improve model generalization and robustness in the face of real-world variability. These include jitter, delay, packet loss, reordering, and incomplete sessions, which can all affect the structure and timing of traffic flows. Unlike adversarial inputs, these perturbations are not crafted to induce misclassification, but they nonetheless pose challenges for models trained under idealized conditions.

Some studies focus on training-time perturbations and assess their impact on the TC effectiveness, that is, the model's performance with unperturbed input. These perturbations are applied during training to promote invariance to minor input variations and to reduce overfitting to specific traffic characteristics.

For instance, Guarino et al. [25] compare contrastive, transfer, and meta-learning strategies, showing that contrastive learning leads to more transferable representations, even when labeled data is limited. Wang et al. [30] benchmark 18 augmentation strategies for Inter Arrival Time (IAT), Transport-level Payload Length (PL), and Packet Direction (DIR), finding that structural changes in TOs (e.g., masking, reordering) yield a stronger generalization than amplitude-based perturbations. Along similar lines, Yang et al. [32] propose *TrafCL*, a framework that combines CL with session-level augmentations to produce robust representations that remain effective even when sessions are incomplete or labeled samples are scarce. Horowicz et al. [35] explore

Table 3
Overview of related work. The last row summarizes our proposal.

Reference	Year	Perturbation Type	Base Classifier Robustness	Strengthened Design		Classification Details					
						Early	MM	TO	Input	Datasets	Task
Liu et al. [21]	2019	NC	●	●	●	○	○	BF	F	Private	TC
Sadeghzadeh et al. [22]	2021	I	●	○	○	●	○	BF	B, PS	VPN-16	TTC
Sharon et al. [23]	2022	I	●	○	●	○	○	F	PS	CIC-IDS2017 Kitsune	AC
Zhang et al. [24]	2022	I	●	○	●	○	○	BF	F	CIC-IDS2018	AC
Guarino et al. [25]	2023	NC	○	●	○	●	○	BF	PS	MIRAGE-19 AppClassNet	TC
Xie et al. [26]	2023	NC	●	○	●	○	○	BF	PS	VPN-16 CICDoHBrw-2020	TC
Li et al. [27]	2023	I	●	○	●	●	○	BF	PS, B, F	USTC2016 VPN-16	ATC
Xie et al. [28]	2024	I	●	○	○	○	○	BF	PS	Private	ATC, WF
Deng et al. [29]	2024	NC	○	○	●	–	○	BF	PS, B, F	SJTU-AN21 VPN-16 ISCXTor2016	TC
Wang et al. [30]	2024	NC	○	●	○	●	○	BF	PS	MIRAGE-19 MIRAGE-2022 Private	TC
Ding et al. [31]	2024	I	●	○	○	●	○	P	B	VPN-16 USTC2016 CIC-IoT2023	TC
Yang et al. [32]	2024	NC	○	●	○	–	○	BF	B, PS	CICDoHBrw-2020 RAT-PN USTC-VPN	TC
Hajaj et al. [33]	2024	NC	○	●	○	○	○	F	PS	UC-Quic-Davis FLASH	SC ATC
Jiang et al. [34]	2024	NC	●	○	●	●	●	BF	B, PS	PHAM21 REN19 Private	TC
Horowicz et al. [35]	2024	NC	○	●	○	○	○	F	B	UC-Quic-Davis VPN-16 + ISCXTor2016	TC
Jin and Apostolaki [36]	2025	I	●	○	●	○	○	BF	F	VPN-16 UTMobileNetTraffic2021 VCAML	ENC ATC QoE-I
Lu et al. [37]	2025	I	●	○	○	○	○	BF	F, PS	CIC-IDS2017	TC
Wang et al. [38]	2025	NC	◐	○	◐	●	●	F	B, PS	USTC2016 Malicious-TLS	AC
This work	2025	NC	●	●	●	●	●	BF	B, PS	VPN-16 MIRAGE-19 MIRAGE-24	TC

Perturbation Type: Network Changes (NC) , Intentional (I) . **Base Classifier Robustness**: Present (●), Absent (○).
Strengthened Design: TC Effectiveness on "unperturbed" traffic ( ), Robustness to "perturbed" traffic ( ).
Early Classification (e.g., $N_p \leq 20$): Present (●), Absent (○), Not inferable from the original work (–). **Multi-modal architecture (MM)**: Present (●), Absent (○).
Traffic Object (TO): Biflow (BF), Flow (F), Network Channel (NC). **Traffic Input:** Packet Sequences (PS), Raw Bytes (B), Statistical Features (F).
Eval. Task: Traffic Classification (TC), Attack Classification (AC), App Traffic Classification (ATC), Traffic Type Classification (TTC), Website Fingerprint (WF), Encapsulation (ENC), Service Classification (SC), QoE Inference (QoE-I).

generalization in a few-shot learning setting using CL. Their approach, based on FlowPic-based inputs, demonstrates that domain-specific perturbations, such as Packet Loss or Round-Trip Time (RTT) changes, outperform generic computer vision augmentations (e.g., rotate, flip, and color jitter). Differently, Hajaj et al. [33] devise a Long Short-Term Memory (LSTM)-based data augmentation technique to produce synthetic traffic that enriches the training dataset and improves the effectiveness of encrypted traffic classifiers on unperturbed inputs.

Other works evaluate robustness under perturbed inputs, without measuring improvement in TC effectiveness. Deng et al. [29] introduce *AN-Net*, a classifier exposed to two types of synthetic noise: (*i*) irrelevant packet noise, simulating traffic from unrelated flows, and (*ii*) per-packet attribute noise, reflecting fluctuations in network environments, leading to discrepancies between training and test conditions. Additionally, they propose a fusion mechanism that combines attributes from different modalities to mitigate the effects of per-packet attribute noise and improve the robustness of the learned representations. Some studies evaluate models across both perturbed and unperturbed data, offering a more comprehensive robustness evaluation. Focusing on Transport Layer Security (TLS) traffic, Xie et al. [26] propose *Rosetta*, a self-supervised training strategy with TCP-aware augmentations. Their approach aims to teach models the implicit semantics of encrypted traffic flows to obtain robust generalization, validated across diverse network setups by replaying traffic across diverse network setups with hosts in different geographic locations. Jiang et al. [34] design a packet permutation-aware model that handles out-of-order packet arrival within flows and uses payload information to enrich the input representation. Their evaluation on both clean and perturbed inputs shows consistent improvements under noise. Conversely, Liu et al. [21] present a more complete evaluation. Their work addresses feature drift by selecting more stable traffic representations and evaluates model robustness under three dimensions: (*i*) unperturbed test data, (*ii*) perturbed test data, and (*iii*) training-time augmentation. This threefold analysis provides one of the most complete assessments of robustness in current TC literature.

Finally, a recent trend explores the use of Generative Artificial Intelligence (GenAI) to improve the quality of learned representations in complex environments. For instance, Wang et al. [38] propose ER-CMGI, a framework combining Variational Autoencoders (VAEs) and Diffusion Models. By generating consistent cross-modal features (e.g., aligning packet lengths with byte sequences), this architectural approach achieves high classification performance on complex traffic patterns. However, such methods rely on computationally-intensive components during inference to reconstruct or align features, aiming to solve robustness through *architectural complexity* rather than *training dynamics*.

Collectively, these works highlight the diversity of strategies used to account for real-world traffic variability, ranging from data augmentations to architectural adaptations.

2.3. Literature gap and positioning

While prior work has explored both intentional attacks and network perturbations, *important gaps remain*, particularly concerning network

-specific transformations ("NC" in the column **Perturbation type**). We address these challenges by introducing *five* categories of traffic perturbations that *emulate realistic network conditions without requiring internal model access*. Our black-box approach highlights model blind spots that clean inputs may conceal, aligning with the goals of adversarial learning. Nonetheless, unlike typical adversarial ML methods ("I" in the column **Perturbation type**) that depend on white-box assumptions or latent feature manipulations, our perturbations operate directly on network traffic and reflect plausible variations, avoiding artificial techniques borrowed from unrelated domains like computer vision [25,30,35].

Only a few prior works, e.g., Horowicz et al. [35], employ comparable black-box, network-level perturbations to assess the robustness of TC models. However, *state-of-the-art approaches typically restrict their analysis to a single dimension/analysis*: they either evaluate how baseline models behave under perturbed inputs (**Base Classifier Robustness**), or they assess robust models (**Strengthened Design**) in terms of *effectiveness* on clean data (sub-column (🏎)) or their *robustness* on perturbed data (sub-column 🛡). When robust models are designed in the same context [35], *a limited number of perturbations and a smaller set of design strategies are explored*. In contrast, our work offers a *comprehensive robustness evaluation by covering all these scenarios using three different representative datasets*. Equally important, our model combines byte-level content and packet sequence patterns ("B" and "PS" in the column **Input**, respectively) *using a multimodal architecture* (**MM**), addressing the relevant challenge of defining network perturbations for traffic payload. To the best of our knowledge, alongside Jiang et al. [34], only the recent generative framework by Wang et al. [38] adopts a comparable multimodal design. However, while the latter pursues feature alignment through computationally intensive diffusion models, our framework prioritizes deployment efficiency, embedding robustness directly into the training dynamics. Furthermore, although their evaluation covers traffic with intrinsic perturbations (e.g., jittering and packet loss), such data is also employed during training [38]. Consequently, their analysis does not assess robustness against *unseen* or varying network conditions.

Furthermore, *we explicitly support early TC* (**Early**), which remains under-explored in the literature [22,25,27,30,31,34]. In this way, *our work offers a holistic perspective on robustness in TC and fills several key gaps in the existing literature*. The last row of Table 3 summarizes the characteristics of our approach to facilitate direct comparison with the literature.

3. Methodology

Section 3.1 describes the multimodal DL classifier used for TC. Then, Section 3.2 introduces the set of perturbation scenarios considered in this work. Last, Section 3.3 describes the learning approaches used to leverage the defined perturbations.

3.1. Multi-modal traffic classification

TC seeks to assign meaningful labels to entities observed in network traffic, which we refer to as TOs, corresponding to logically grouped sets of packets, as seen from a VP. In this study, as the fundamental TO, we treat bidirectional flows—or *biflows*—each consisting of all packets exchanged between a pair of endpoints, identified by a the tuple: $\langle \texttt{IP}_{\texttt{src}}, \texttt{port}_{\texttt{src}}, \texttt{IP}_{\texttt{dst}}, \texttt{port}_{\texttt{dst}}, \texttt{L4}_{\texttt{proto}} \rangle$, where the source and destination are interchangeable [40].

Our goal is to classify each biflow based solely on its early packets—a strategy known as *early* classification. By making decisions at the outset of a flow, the system can promptly adapt to changing network conditions and enforce traffic management policies, which is vital in real-time scenarios. A biflow can be characterized through *multiple modalities*, each offering a *distinct perspective* for classification [15].

The two most direct inputs are:

1. the initial raw bytes transmitted by the application (PAY)—namely, the transport-layer Service Data Unit (SDU), and
2. the sequence of header fields from the initial packets (PSQ), such as Packet Size (PS), DIR and IAT.

Table 4
Impact of each perturbation on PAY and PSQ traffic inputs.

	PAY	PSQ
TJitter	⊗	✓
RTOaft	?	✓
RTObef	?	✓
PHiding	✓	✓
DHiding	✓	✓
$\texttt{DHiding}_{\texttt{loss}}$	✓	✓
VPC	⊗	✓
$\texttt{VPC}_{\texttt{loss}}$	?	✓

✓: Affected.
?: Possibly Affected.
⊗: Unaffected.

Formally, a generic DL classifier can be represented by a function $\boldsymbol{f}_{\theta,\varphi} : \mathcal{X} \to C$ that transforms input data into output labels. Specifically, $\boldsymbol{f}_{\theta,\varphi}(\cdot)$ can be decomposed into two parts, the model backbone $\boldsymbol{f}_{\theta} : \mathcal{X} \to \mathcal{Z}$—the *feature extractor*—and the model head $\boldsymbol{f}_{\varphi} : \mathcal{Z} \to C$—the *classifier*. The feature extraction process maps input data $\boldsymbol{x}$ into a latent representation $\boldsymbol{z} = \boldsymbol{f}_{\theta}(\boldsymbol{x}) \in \mathcal{Z}$, whereas the classifier maps the latent representation to a per-class probability distribution $\boldsymbol{p}(\boldsymbol{x}) = \boldsymbol{f}_{\varphi}(\boldsymbol{z}) \in [0,1]^{|C|}$ (i.e., *soft-output*), typically via a "softmax" activation in the final layer, where each component $p_c(\boldsymbol{x})$ indicates the confidence of the model that the input $\boldsymbol{x}$ belongs to class c. The predicted label is obtained as $\hat{y}(\boldsymbol{x}) = \arg\max_{1,\cdots,|C|} p_i(\boldsymbol{x})$.

In a standard *supervised learning* setting, model training seeks to optimize parameters $(\boldsymbol{\theta}, \boldsymbol{\varphi})$ to minimize a classification loss $\mathcal{L}(y, \hat{y})$, where $y \in C$ is true class label. The typical loss is the *Categorical Cross-Entropy (CCE)*: $\mathcal{L}_{\texttt{CCE}}(y, \hat{y}) = -\sum_{i=1}^{|C|} \mathbf{1}_{[y=i]} \log(p_i)$, which compares the soft-output provided by the model with the true label, penalizing lower probabilities assigned to the true class. Since the classifier $f_{\varphi}(\cdot)$ is typically a single linear layer, minimizing the CCE loss implicitly encourages the feature extractor $f_{\theta}(\cdot)$ to structure the latent space so that classes become linearly separable, thereby improving accuracy and generalization.

In a multimodal setting, the input data $\boldsymbol{x}$ consists of M distinct modalities $\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_M$, derived from the same biflow. Each modality $\boldsymbol{x}_m$ is processed by a dedicated feature extractor $f_{\theta_m}(\cdot)$—i.e., a *branch*—, resulting in a latent representation $\boldsymbol{z}_m = f_{\theta_m}(\boldsymbol{x}_m)$, for $m = 1, \cdots, M$.

The resulting feature vectors are then combined through a fusion function $\mathcal{F}_{\boldsymbol{\phi}}$ modeling interactions across modalities and producing a joint (latent) representation $\bar{\boldsymbol{z}} = \mathcal{F}(\boldsymbol{z}_1, \boldsymbol{z}_2, \cdots, \boldsymbol{z}_M)$, which is passed to the classifier $\boldsymbol{f}_{\varphi}(\cdot)$. This enables an integrated decision-making process based on complementary modality-specific features.

Notably, *different modalities can be affected differently by network changes*, due to their heterogeneous nature. For example, during congestion, IATs (captured by PSQ) may increase, while raw bytes (from PAY) remain largely unchanged. Accordingly, we hypothesize that the *synergistic combination of these modalities through multimodal learning yields a more comprehensive and robust representation*, enhancing resilience to variations in network conditions.

3.2. Perturbation scenarios overview

To evaluate the robustness of traffic classifiers, we designed traffic perturbation scenarios that reflect realistic deployment conditions. While traffic classifiers are typically trained on traffic captured in controlled environments near the network edges, real-world monitoring points—such as middleboxes—may operate in contexts where the view of traffic is biased or distorted (e.g., middleboxes are mainly deployed in Tier2/ISP networks [41]) due to factors such as asymmetric routing, delayed observations, packet loss, or timing variations [42,43].

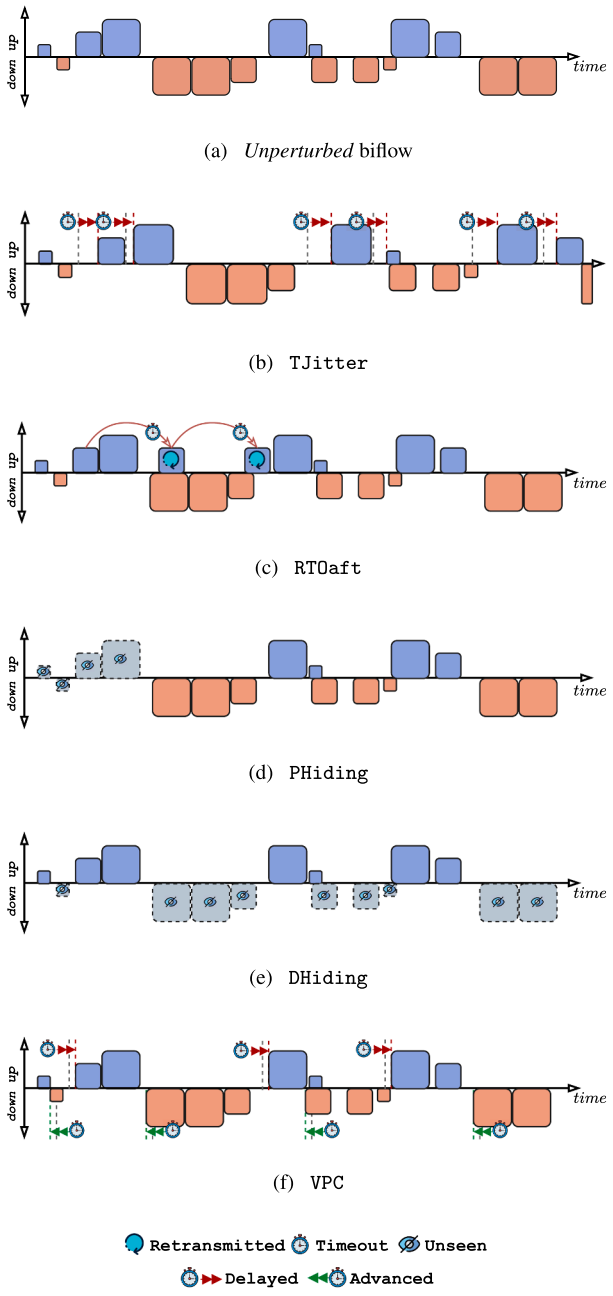


Fig. 1. Visual effects of TJitter (b), RTOaft (c), PHiding (d), DHiding (e), and VPC (f) on a biflow (a). The *x-axis* represents time, while the *y-axis* indicates the traffic direction (e.g., upstream or downstream). Each box corresponds to a packet, and its size reflects the actual payload length.

Specifically, we introduce a set of perturbations that selectively alter the packet sequence and/or the payload content observed at the VP to simulate these effects. Because our goal is to enable early biflow-level TC using two widely-adopted inputs—PAY and PSQ—we systematically explain each perturbation: its underlying motivation, how it reshapes the biflow structure, and which traffic inputs it influences (see Table 4).

Importantly, although these perturbations emulate real-world effects, they are applied synthetically to traffic originally captured in controlled environments. To support reproducibility, we include pseudocode for each perturbation in Appendix A. In the following, we describe the perturbations, whose effects are illustrated in Fig. 1.

Directional Time Jittering (TJitter) (Fig. 1b): since network conditions are inherently dynamic and subject to transient variations, packet timing—particularly IAT—can be strongly affected by network environmental changes.

To assess model robustness to timing fluctuations, TJitter selectively alters the IAT of packets of a given biflow based on their direction (i.e., upstream or downstream). Specifically, a small delta-time is added (*delaying*) to the IAT of packets traveling in a given direction, with these values sampled from direction-specific probability distributions, allowing for asymmetric timing distortions.

This transformation shares similarities with the jittering proposed in [44], but differs by explicitly modeling distinct delays per direction, better reflecting the real-world condition of asymmetric path routing. For simplicity, to model jitter in the IAT of the ith packet in a biflow, $\texttt{IAT}_i$, we sample two independent terms $\lambda_{a,i} \sim \mathcal{N}^+(\mu, \sigma_{\texttt{DIR}_i})$ and $\lambda_{b,i} \sim \mathcal{N}^+(\mu, \sigma_{\texttt{DIR}_i})$, with $\texttt{DIR}_i \in \{upstream, downstream\}$. The perturbed $\texttt{IAT}_i$, denoted as $\widehat{\texttt{IAT}}_i$, is computed as:

$$\widehat{\texttt{IAT}}_i = \begin{cases} \texttt{IAT}_i + |\lambda_{a,i} - \lambda_{b,i}|, & \text{if } \texttt{DIR}_i = \texttt{DIR}_{i-1} \\ \texttt{IAT}_i + |\lambda_{a,i} + \lambda_{b,i}|, & \text{if } \texttt{DIR}_i \neq \texttt{DIR}_{i-1} \end{cases} \tag{1}$$

which introduces jitter based on the direction of the current packet, $\texttt{DIR}_i$, and that of the preceding one, $\texttt{DIR}_{i-i}$. Notably, since TJitter affects only packet timing, it does not alter the packet content (i.e., the PAY input) or the arrival order of packets as observed from the VP.

Retransmission Timeout (RTO) (Fig. 1c) emulates the retransmission of packets within a biflow, where both the packets to be retransmitted and the number of retransmissions are randomly selected. Packet retransmissions are modeled through a **retransmission rate** r, with each retransmitted packet being re-sent after a preset **Timeout**.

In this context, we devised two realistic sub-scenarios to model different packet retransmission conditions: (*i*) RTOaft, where the VP observes both the original and retransmitted packets—i.e., the loss occurs *after* the VP; (*ii*) RTObef, where the VP observes only retransmitted copies—i.e., the original packet was lost *before* the VP.

The proposed transformation is a generalization of the simplified version of RTObef suggested by Xie et al. [26]. Notably, although retransmissions do not alter the content of packets, they can indirectly affect the construction of the PAY input since it is built by concatenating the payloads of the first packets until N_b bytes are reached, and the number of packets required depends on their payload sizes.

As a result, retransmissions can substantially affect PAY, particularly when they involve the earliest packets. This effect is pronounced in the RTObef setting, where the VP misses the original transmissions. If the initial packet is lost before reaching the VP and only the retransmission is observed, its payload may not contribute to PAY as it would under lossless conditions—leading to a distorted view of the biflow. In addition, our RTO transformation—similarly to what is proposed in [26] and adopted in [30,35]—does not account for the potential impact on the IAT of the first response packet when the RTO involves a request packet. While such a retransmission could delay the server's response and thus alter the observed IAT, we intentionally avoid modeling this effect to maintain consistency with prior work and prevent the introduction of speculative timing artifacts [1] Moreover, RTObef shares some similarities to the packet loss applied by Yang et al. [32], where they randomly remove 10% to 50% packets from a given biflow. However, our proposal goes beyond the simple packet loss by also implementing a retransmission mechanism.

Packets Hiding (PHiding) (Fig. 1d) models the scenario where the VP misses a biflow's initial packets and must classify it based on the later

[1] We do not explicitly model request/response semantics in a biflow and preserve all original packet IATs for simplicity.

ones (*late-start classification*), for example, due to a start-up delay in the monitoring system or classifier.

The perturbation affects both PSQ and PAY, although with different intensity. Intuitively, PSQ may be less sensitive to missing initial packets because the classifier still sees a slightly shifted packet sequence that remains close to the original. PAY, instead, could be heavily affected, as it relies on the biflow's first bytes. The problem is amplified in TLS traffic, where initial packets carry plaintext metadata (e.g., *Server Name Indication*, *Cipher Suites*) while later packets are usually encrypted [16, 45].

Similar to PHiding, Yang et al. [32] apply packet loss, but our approach discards only the first packets of a biflow, starting from a random position.

Direction Hiding (DHiding) (Fig. 1e) models the scenario where the forward and backward paths between Client and Server are asymmetric, so only one direction crosses the observation point. This condition becomes more likely as this point is placed deeper within the network, i.e., farther from the endpoints. We also consider a variant, **Direction Hiding with Packet Loss ($\texttt{DHiding}_{\texttt{loss}}$)**, which assumes that even packets in the visible direction may be missing. Each packet is dropped independently with probability p, accounting for situations where packets in the same direction may follow different paths, possibly due to routing asymmetries or load balancing.

Assuming bidirectional communication where both endpoints generate traffic, both versions primarily affect PSQ by altering the observed packet sequence, making it unidirectional or even incomplete. Conversely, their impact on PAY mainly depends on the direction and the size of the earliest packets of the biflow, which are typically used to extract PAY.

Vantage Point Centralization (VPC) (Fig. 1f) models the case where a VP is placed far from the endpoints but still observes traffic in both directions. In such conditions, packets exchanged exhibit different timing characteristics due to asymmetric delays, primarily influenced by the RTT. We assume that the RTT between the Client and the Server can be decomposed into two segments [46]: *from-client-to-VP* and *from-VP-to-server*. We introduce a parameter $\alpha \in [0, 1]$ to represent the proportion of RTT attributed to the Client-to-VP path:

$$RTT = RTT_{C\to V} + RTT_{S\to V} \tag{2}$$

where $RTT_{C\to V} = \alpha \cdot RTT$ and $RTT_{S\to V} = (1-\alpha)\cdot RTT$, and α represents the relative position of the VP along the Client-to-Server path, with $RTT_{C\to V}$ (resp. $RTT_{S\to V}$) denoting the RTT between Client and VP (resp. Server and VP).

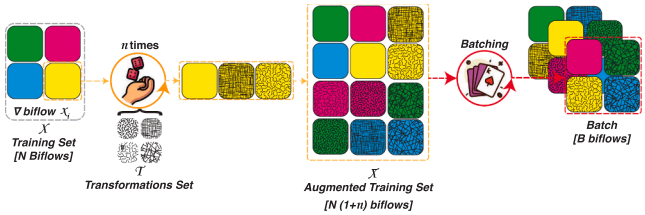
Based on this decomposition, the IAT between consecutive packets sent in opposite directions depends on the total RTT and reflects the asymmetry between upstream and downstream paths. Specifically, consider two consecutive packets with opposite directions: a packet p_c sent by the Client and a packet p_s sent by the Server. Whether p_c precedes p_s or p_s precedes p_c, the IATs of the later-arriving packet can be computed as:

$$\texttt{IAT}_{p_s} = T_{p_c} + T^k_{p_s} + T_{p_s} + (1-\alpha)\cdot RTT \tag{3}$$

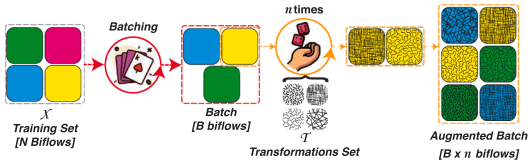
$$\texttt{IAT}_{p_c} = T_{p_s} + T^k_{p_c} + T_{p_c} + \alpha \cdot RTT \tag{4}$$

Here, $\texttt{IAT}_p$ is the IAT of the packet p, T^k_p denotes the thinking time at the sender before transmitting packet p, and T_p denotes the transmission time for the packet p, which typically depends on its PS.

In our experimental setup, traffic is usually captured near the client, so we assume $\alpha \approx 0$. To model a VP farther from the endpoints, we vary α and recompute the IATs to introduce the desired timing asymmetry. Notably, changes in the VP position do not affect the IATs between packets sent in the same direction, as the RTT components cancel out in the difference.



(a) Data augmentation



(b) Contrastive learning

Fig. 2. Transforming procedure in Aug (a) and CL (b). In Aug, transformations are applied before training to create a fixed augmented training set. Batches are then formed and may include both original and transformed samples. In CL, transformations are applied "on the fly" during training. Each batch contains only transformed samples. **N** is the number of original samples. **B** is the batch size. **T** is the number of independent transformations per sample.

We further defined the **Vantage Point Centralization with Packet Loss ($\texttt{VPC}_{\texttt{loss}}$)** variant with each packet independently dropped with probability p. This models the increased likelihood of partial visibility when the VP is located deeper in the network, due to routing asymmetries, and load balancing.

Regarding input perturbation, VPC affects only PSQ, as it alters IATs, without modifying packet order or payload content, whereas $\texttt{VPC}_{\texttt{loss}}$ may also impact PAY when early packets are lost. These scenarios help assess whether a model—typically trained on traffic captured near the Client—remains effective when deployed in more central, challenging network positions.

3.3. Perturbation-aware learning approaches

Herein, we describe the two strategies for integrating perturbation-based transformations into the training process: *supervised data augmentation* and *contrastive learning*. Both apply controlled transformations to original samples, but with different objectives. The former directly trains on transformed samples labeled as their corresponding originals to improve generalization [47], while the latter exploits transformations to create semantically-similar pairs, enabling the model to learn robust representations via instance discrimination [48].

Supervised Data Augmentation (Aug): is typically used to enrich supervised learning by increasing sample diversity or mitigating class imbalance. Since DL models require large datasets to generalize, Aug is particularly beneficial when labeled samples are limited. Hence, we applied perturbations to training samples to generate realistic variants and used within standard supervised learning.

The underlying idea is that, by exposing the model to a wider range of plausible input variations, Aug helps it learn invariant representations, i.e., feature embeddings $\boldsymbol{z} = \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x})$ that remain consistent across different perturbations of inputs belonging to the same class. Hence, since the classifier $\boldsymbol{f}_{\boldsymbol{\varphi}}(\cdot)$ is linear, the model is implicitly encouraged to cluster perturbed samples of the same class in the latent space, thereby improving inter-class separation generalization. Unlike augmentation techniques used in domains such as computer vision (e.g., rotations or flipping), we *do not apply generic, domain-agnostic transformations*. We instead apply domain-specific perturbations tailored to network biflows, using the realistic transformations defined in Section 3.2 to generate semantically plausible traffic variants.

Formally, given a set of training samples $\mathcal{X} \equiv \{\boldsymbol{x}_i\}_{i=1}^{N}$, Aug creates n perturbed samples for each $\boldsymbol{x}_i$ by independently applying n transformations sampled from a predefined set $\mathcal{T}$. We adopt a *pre-augmented* strategy (see Fig. 2a) where all perturbations are applied before training—i.e., prior to mini-batches construction—thus decoupling transformation sampling from the training loop.

For each $\boldsymbol{x}_i$, we randomly select n transformations $\tau_1(\cdot), \tau_2(\cdot), \cdots, \tau_n(\cdot) \in \mathcal{T}$ and apply them separately to obtain the augmented versions, resulting in a *transformed* training set:

$$\hat{\mathcal{X}} \equiv \{\hat{\boldsymbol{x}}_k = \tau_j(\boldsymbol{x}_i) | \boldsymbol{x}_i \in \mathcal{X}, j \in \{1, \dots, n\}\} \tag{5}$$

with $|\hat{\mathcal{X}}| = n \cdot N$. The final *augmented* training set is obtained by combining both the original and augmented samples:

$$\tilde{\mathcal{X}} = \mathcal{X} \cup \hat{\mathcal{X}} \tag{6}$$

This ensures that the training set remains representative of the original distribution while also promoting generalization through increased sample diversity.

Contrastive Learning (CL) is a representation learning approach that explicitly regularizes the latent space by teaching models to recognize similarity and dissimilarity among samples. Accordingly, it optimizes a feature extractor by comparing the projections of samples in the latent space using a contrastive loss to bring similar samples closer and push dissimilar ones apart. This is achieved by selecting a reference (ref. to as *anchor*) representation and comparing it to a *positive* (viz. similar) and one or more *negative* (viz. dissimilar) samples. CL methods can be either self-supervised or supervised, depending on how similarity is defined and whether labels are used.

Self-supervised Contrastive Learning (SimCon) [48] can be seen as a branch of unsupervised learning, as it does not involve any manual labeling [49]. It assumes each sample is only similar to its own augmented versions and dissimilar to all others. Unlike Aug, in CL the augmentations are applied independently at each training step, i.e., new pairs are generated for every batch (see Fig. 2b), exposing the model to constantly varying transformations and improving robustness and generalization.

Formally, given a mini-batch $\mathcal{B} = \{\boldsymbol{x}_i\}_{i=i}^{B} \sim \mathcal{X}$, for each $\boldsymbol{x}_i \in \mathcal{B}$, two transformed versions $\tilde{\boldsymbol{x}}_i = \tau_1(\boldsymbol{x}_i)$ and $\tilde{\boldsymbol{x}}_j = \tau_2(\boldsymbol{x}_i)$ are obtained by applying two distinctive transformations $\tau_1(\cdot), \tau_2(\cdot) \sim \mathcal{T}$. This results in an "augmented" batch $\tilde{\mathcal{B}} = \{\tilde{\boldsymbol{x}}_1, \tilde{\boldsymbol{x}}_2, \dots, \tilde{\boldsymbol{x}}_{2B}\}$ containing two transformed versions of each original input sample $\boldsymbol{x}_i$. Therefore, for each anchor $\tilde{\boldsymbol{x}}_i$, its corresponding counterpart $\tilde{\boldsymbol{x}}_j$ (i.e., the one originated from the same original sample), is treated as a positive, while the remaining $2 \cdot (B-1)$ samples $\tilde{\boldsymbol{x}}_k \in \tilde{\mathcal{B}} \backslash \{\tilde{\boldsymbol{x}}_i, \tilde{\boldsymbol{x}}_j\}$ are considered as negative.

The learning process penalizes cases where the latent representation of positive pairs $(\boldsymbol{f}_\theta(\tilde{\boldsymbol{x}}_i), \boldsymbol{f}_\theta(\tilde{\boldsymbol{x}}_j))$ are far apart, and those of negative pairs $(\boldsymbol{f}_\theta(\tilde{\boldsymbol{x}}_i), \boldsymbol{f}_\theta(\tilde{\boldsymbol{x}}_k))$ are too close. This is typically performed by minimizing the *Normalized Temperature-scaled Cross-Entropy (NT-Xent)* loss function [48] (ref. to as $\mathcal{L}_{\text{SimCon}}$ in the following), which aims to maximize the mutual information between the anchor and positive samples. Formally, this loss function is defined as:

$$\mathcal{L}_{\text{SimCon}}(\tilde{\boldsymbol{x}}_i) = -\ln \frac{e^{\text{sim}(\boldsymbol{f}_\theta(\tilde{\boldsymbol{x}}_i), \boldsymbol{f}_\theta(\tilde{\boldsymbol{x}}_j))/\beta}}{\sum_{k=1}^{2N} \mathbf{1}_{[k \neq i]} e^{\text{sim}(\boldsymbol{f}_\theta(\tilde{\boldsymbol{x}}_i), \boldsymbol{f}_\theta(\tilde{\boldsymbol{x}}_k))/\beta}} \tag{7}$$

where $\text{sim}(\cdot)$ is a similarity function (e.g., cosine similarity) and $\beta \in \mathbf{R}^+$ is a temperature parameter.

It is worth noting that the original framework includes a *projection head* that maps the latent representations into a "contrastive space" where $\mathcal{L}_{\text{SimCon}}$ is applied. In our setting, $\mathcal{L}_{\text{SimCon}}$ is directly applied to the latent representations as CL approaches are fully integrated into the supervised pipeline, and the resulting representations are directly used for the classification task.

Supervised Contrastive Learning (SupCon) [50] aims to overcome the limitations of SimCon by incorporating class labels directly into the learning process. In SimCon, samples from the same class as the anchor are treated as negatives, thus preventing the model from learning the relationship between samples of the same class. Thus, SupCon refines the latent space by ensuring that samples of the same class are grouped together while samples from different classes are well-separated.

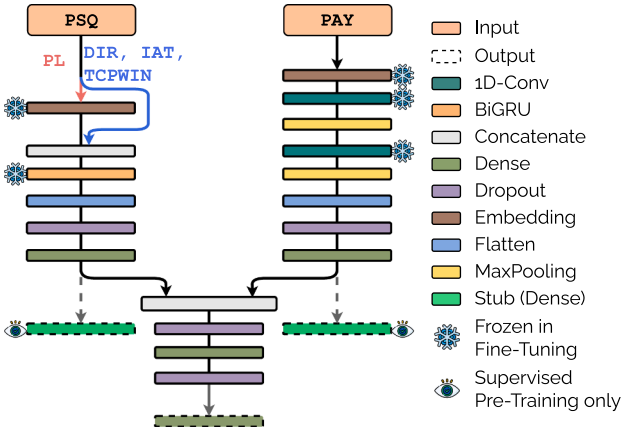


Fig. 3. MIMETIC-ENHANCED classifier. Colors indicate the layer type.

In this case, the learning process is driven by a *SupCon* loss (ref. to as $\mathcal{L}_{\text{SupCon}}$ in the following), which is an extension of the classical $\mathcal{L}_{\text{SimCon}}$. Formally, this loss is defined as:

$$\mathcal{L}_{\text{SupCon}}(\tilde{\boldsymbol{x}}_i) = -\frac{1}{|\mathcal{P}(i)|} \sum_{\tilde{\boldsymbol{x}}_j \in \mathcal{P}(i)} \ln \frac{e^{\text{sim}(\boldsymbol{f}_\theta(\tilde{\boldsymbol{x}}_i), \boldsymbol{f}_\theta(\tilde{\boldsymbol{x}}_j))/\beta}}{\sum_{k=1}^{2N} \mathbf{1}_{[k \neq i]} e^{\text{sim}(\boldsymbol{f}_\theta(\tilde{\boldsymbol{x}}_i), \boldsymbol{f}_\theta(\tilde{\boldsymbol{x}}_k))/\beta}} \tag{8}$$

where $\mathcal{P}(i) \equiv \{\tilde{\boldsymbol{x}}_k \in \tilde{\mathcal{B}} \backslash \{\boldsymbol{x}_i\} : y(\tilde{\boldsymbol{x}}_k) = y(\tilde{\boldsymbol{x}}_i)\}$—with $y(\cdot)$ denoting the actual label—, $\text{sim}(\cdot)$ is a similarity function (e.g., cosine similarity) and $\beta \in \mathbf{R}^+$ is a temperature parameter. In both SimCon and SupCon, the final loss is computed symmetrically across all positive pairs $(\tilde{\boldsymbol{x}}_i, \tilde{\boldsymbol{x}}_j)$ and $(\tilde{\boldsymbol{x}}_j, \tilde{\boldsymbol{x}}_i)$ within the batch.

4. Experimental setup

In this section, we describe the DL architecture (Section 4.1) and the proposed training procedures (Section 4.2). Then, Section 4.3 reports the datasets employed for the experiments, whereas Section 4.4 outlines the evaluation procedure and the metrics.

4.1. Reference multi-modal architecture

We use MIMETIC-ENHANCED [16] as our baseline model. The architecture and training methodology are described below.

Architecture Overview: Fig. 3 depicts the architecture of MIMETIC-ENHANCED, a multimodal DL model that combines two distinct traffic views (i.e., $M = 2$). Each modality is processed by a distinct branch, optimized to extract salient features from its corresponding traffic input. The first branch processes a sequence of $N_h = 4$ header fields extracted from the first $N_p = 10$ packets (viz. PSQ). These fields include the PL, DIR—represented as $\{0 = \text{upstream}, 1 = \text{downstream}\}$, TCP Window Size (TCPWIN), which is set to zero for UDP packets, and IAT. The second branch handles the first $N_b = 576$ bytes of payload data (viz. PAY), treated as a flat byte sequence. Each input is first passed through a *trainable embedding layer* that maps raw input elements into a vector of dimension $e = 10$. This results in an embedding matrix $\boldsymbol{E} \in \mathbb{R}^{N \times e}$, where N is equal to N_p and N_b for PSQ and PAY, respectively. Notably, for PSQ, only the sequence of PL is processed with the embedding layer. The resulting vectors are concatenated with the remaining header fields to form the full input to the PSQ branch. In other terms, after the embedding layer, the PSQ data has a resulting size $N_p \times (e + N_h - 1)$.

The PSQ branch comprises a Bidirectional Gated Recurrent Unit layer (BiGRU, with 64 units—returning sequences, followed by a 256-unit dense layer. The PAY branch is composed of two 1D convolutional

Table 5
Overview of the training variants used to integrate Aug and CL (i.e., SimCon or SupCon) into the two-phase training procedure of MIMETIC-ENHANCED. Each configuration applies perturbation-based strategies—either individually or in combination—across the PT and FT phases. The baseline configuration is highlighted in blue.

	Pre-Training					Fine-Tuning				
	M	S	O	T	$\mathcal{L}$	M	S	O	T	$\mathcal{L}$
Base	✓	✗	✓	✗	$\mathcal{L}_{\text{CCE}}$	✗	✓	✓	✗	$\mathcal{L}_{\text{CCE}}$
Aug_{PT}	✓	✗	✓	✓	$\mathcal{L}_{\text{CCE}}$	✗	✓	✓	✗	$\mathcal{L}_{\text{CCE}}$
$\text{SimCon}_{\text{PT}}$	✓	✓	✗	✓	$\mathcal{L}_{\text{SimCon}}$	✗	✓	✓	✗	$\mathcal{L}_{\text{CCE}}$
$\text{SupCon}_{\text{PT}}$	✓	✓	✗	✓	$\mathcal{L}_{\text{SupCon}}$	✗	✓	✓	✗	$\mathcal{L}_{\text{CCE}}$
Aug_{FT}	✓	✗	✓	✗	$\mathcal{L}_{\text{CCE}}$	✗	✓	✓	✓	$\mathcal{L}_{\text{CCE}}$
$\text{Aug}_{\text{PT}} \rightarrow \text{Aug}_{\text{FT}}$	✓	✗	✓	✓	$\mathcal{L}_{\text{CCE}}$	✗	✓	✓	✓	$\mathcal{L}_{\text{CCE}}$
$\text{SimCon}_{\text{PT}} \rightarrow \text{Aug}_{\text{FT}}$	✓	✓	✗	✓	$\mathcal{L}_{\text{SimCon}}$	✗	✓	✓	✓	$\mathcal{L}_{\text{CCE}}$
$\text{SupCon}_{\text{PT}} \rightarrow \text{Aug}_{\text{FT}}$	✓	✓	✗	✓	$\mathcal{L}_{\text{SupCon}}$	✗	✓	✓	✓	$\mathcal{L}_{\text{CCE}}$

Architectural Layers (): Per-modality Branches (**M**), Final Layers (**S**), Trained (✓), Not Trained (✗).—*Objective Loss* ($\mathcal{L}$): CCE ($\mathcal{L}_{\text{CCE}}$), NT-Xent/SimCon ($\mathcal{L}_{\text{SimCon}}$), SupCon ($\mathcal{L}_{\text{SupCon}}$).—*Training Data* (): Original (**O**), Transformed (**T**), Used (✓), Not Used (✗).

layers (1D−Conv, with 16 and 32 filters, kernel size of 25, and stride 1, each followed by a max-pooling layer with window size of 3 and unit stride, and a dense layer with 256 neurons. To model cross-modality correlations, outputs from both branches are concatenated and passed through a shared dense layer with 128 neurons, followed by a softmax-activated dense layer for classification. All layers use RelU activations. To regularize training and reduce overfitting, a dropout—with a rate of 0.2—is applied after each dense layer and at the end of both modality branches.

Details of the training process: MIMETIC-ENHANCED is trained using a *two-stage procedure*. In the **Pre-Training (PT)** stage, each single-modality branch is independently trained using a *softmax stub* to guide the branch-specific learning. This is followed by a **Fine-Tuning (FT)** stage, where the entire architecture is optimized after freezing the embedding, convolutional, and BiGRU layers (❄). Both training phases minimize a CCE loss using the SGD optimizer (batch size of 64), combined with a custom *ReduceLROnPlateau* scheduler that monitors the validation loss and decreases the learning rate by a factor of 3 after 20 consecutive epochs without improvement. [2]

4.2. Advanced training for multi-modal traffic classification

We integrated Aug and CL into the "supervised" training pipeline of MIMETIC-ENHANCED to assess their impact on both TC effectiveness and robustness. As summarized in Table 5, we devised several training variants by applying Aug and CL either individually or jointly across the different phases of the training process (cf. Section 4.1).

For each variant, the table details the setup adopted during the PT and FT phases, in terms of:

1. the architectural layers involved ()—i.e., per-modality branches (**M**) or final layers (**S**);
2. the training data used ()—i.e., original (**O**) or transformed (**T**); and the objective loss function used to optimize the model—i.e., $\mathcal{L}_{\text{CCE}}$, $\mathcal{L}_{\text{SimCon}}$, or $\mathcal{L}_{\text{SupCon}}$.

The baseline configuration (ref. to as Base in the following) relies on original data (**O**) and the CCE in both PT and FT. We then considered configurations where transformed data (**T**) is used only during PT (i.e., Aug_{PT}, $\text{SimCon}_{\text{PT}}$, and $\text{SupCon}_{\text{PT}}$) to investigate whether training only the early layers (**M**) improves representation learning. The classifier is always trained solely on original data, so it is never exposed to transformed inputs and is not explicitly trained to recognize them. We also assess the Aug_{FT} configuration where Aug is applied only during FT. In this setup, PT uses only original samples, while FT is performed on the augmented training set (**O** + **T**). This approach isolates the effect of exposing only the model's final layers (**S**) to transformed data, allowing us to assess whether this selective exposure improves generalization by refining the classifier's decision boundaries without altering feature extraction. We do not apply SimCon or SupCon during FT, since this phase specifically aims to refine the classifier using a supervised objective, whereas CL focuses on shaping the latent space rather than directly optimizing class separation.

In addition, we explore combined configurations in which Aug or CL is applied in both PT and FT. More in detail, $\text{Aug}_{\text{PT}} \rightarrow \text{Aug}_{\text{FT}}$ uses original and transformed samples (**O** + **T**) in both training phases, enabling us to assess the impact of consistent exposure to increased data variability throughout the entire learning process. Conversely, $\text{SimCon}_{\text{PT}} \rightarrow \text{Aug}_{\text{FT}}$ and $\text{SupCon}_{\text{PT}} \rightarrow \text{Aug}_{\text{FT}}$ adopt a hybrid strategy: the model is pre-trained with CL on transformed data (**T**) and then fine-tuned on an augmented dataset (**O** + **T**) using standard cross-entropy.

These setups aim to evaluate whether combining robust representation learning—enabled by CL—with augmented FT improves generalization by leveraging both latent space regularization and classifier adaptation to data variability. For contrastive variants, no output stub is used during PT, and the loss is applied directly to the latent representations. Additionally, before the final supervised FT, an intermediate contrastive FT step updates only the shared layers and the final dense layer (**S**), while all remaining layers (**M**) remain frozen.

In our setup, we consistently generated $n = 2$ transformed versions of each sample during all training stages involving Aug or CL. In both (training) strategies, perturbations are sampled independently for every sample: both the transformation type (e.g., RTOaft) and its hyperparameters (e.g., retransmission probability, $r\%$) are drawn from predefined distributions. This prevents the model from overfitting to fixed, deterministic transformations and encourages the learning of invariant features across diverse and realistic variations. Conversely, during robustness evaluation, the same transformation is applied deterministically to all test samples under fixed hyperparameter settings. This procedure is repeated across multiple configurations to enable controlled sensitivity analysis and to ensure reproducible results.

4.3. Datasets description

In this work, we employ three publicly available datasets, namely VPN-16 [19], MIRAGE-19 [8], and MIRAGE-24 [20].

VPN-16 [19] was collected at the Canadian Institute for Cybersecurity and provided in raw PCAP format. It includes human-generated traffic spanning various types, with app-related information collected from both regular and VPN-encapsulated sessions. The dataset comprises 6 traffic types and 15 apps, providing the ground truth at the trace level for the encapsulation type, service type, and app.

Given that VPN encapsulation can alter apps' traffic patterns, we consider both the app and the encapsulation type as joint labels. Fig. 4a depicts the biflow distribution across all ⟨app, encapsulation⟩ pairs. Since some classes contain only a few dozen biflows, we retained only those with at least 100 biflows (17 classes, highlighted in yellow) to ensure sufficient representation.

MIRAGE-19 [8] contains per-flow mobile traffic logs of 40 Android apps collected at the ARCLAB laboratory of the University of Napoli Federico II between May 2017 and May 2019. The data was collected by employing 3 *rooted* Android devices [3], which were used by ≈ 300 volunteers,

[2] The initial learning rate is set to 0.1, and training stops early if the learning rate falls below 0.0001. To avoid overfitting, we use a validation-based early-stopping technique, where we set the patience to 20 epochs without setting the min_delta parameter.

[3] These devices were running CyanogenMod v13.0 custom firmware, equivalent to Android 6.0.1

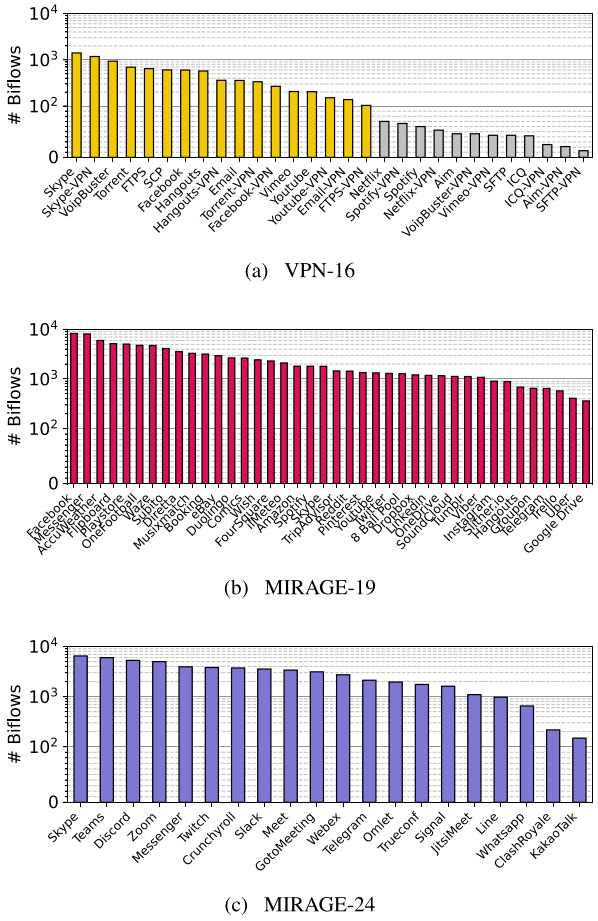


Fig. 4. Biflows distribution for (a)VPN-16, (b) MIRAGE-19, and (c) MIRAGE-24. Values on the y-axis are in *SymLog* scale. For VPN-16, the considered classes are colored in yellow. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

who interacted with the selected apps for 5 – 10 minute sessions. Each session produced a PCAP file and a `strace` log .[4] Fig. 4b shows the biflow distribution across all apps.

MIRAGE-24 [20] was collected between April 2021 and December 2023 at the University of Napoli Federico II by leveraging an improved version of the `MIRAGE` [8] collection framework. The dataset contains mobile traffic data from 20 popular Android apps, collected with four devices [5] by 240 volunteers. These apps were used to perform various activities, such as *Chat*, *Audio-call*, *Video-call*, *Video-streaming*, and *Online Gaming*. Fig. 4c shows the biflow distribution across all apps.

Dataset selection rationale: the selected datasets *complement each other*. Specifically, VPN-16 offers controlled VPN and non-VPN traces. Conversely, MIRAGE-19 adds mobile traffic from diverse devices, users, and network conditions. Finally, MIRAGE-24 updates this view with newer Android versions, updated apps, and labeled activities. Together, they span (*a*) varied environments, (*b*) periods, and (*c*) device generations, providing a solid basis for assessing robustness and generalization.

Data pre-processing: to ensure reproducibility, we apply a standardized pre-processing pipeline. First, packets with no payload (TCP signaling packets such as SYN, ACK, and FIN) are discarded. This ensures that the model focuses on application-layer behavior rather than transport-layer signaling, which is often application-agnostic. After filtering, we normalize IAT, DIR, and TCPWIN using *min-max* scaling. [6] Consistent with early TC, we extract the first $N_b = 576$ bytes for the PAY modality and the first $N_p = 10$ packets for PSQ. For biflows with fewer than N_b bytes or N_p packets, we apply zero-padding to maintain fixed input dimensions. We stress that perturbations are applied to the entire biflow *before* extracting the PAY and PSQ inputs.

4.4. Evaluation procedure and metrics

To rigorously assess TC performance, we employ a stratified ten-fold cross-validation strategy. In each fold, 90% of the biflows are utilized for training, while the remaining 10% is for testing. Within the 90% partition, we apply an 80/20 split for training and validation purposes. The validation set plays a key role in enabling early stopping, which helps mitigate overfitting during training. To thoroughly assess our models, we employ different evaluation metrics, described in the following.

Classification Performance: we used the (*macro*) `F1-score`, which is the average of the harmonic means of per-class precisions and recalls, defined as:

$$\texttt{F1-score} \triangleq \frac{1}{|C|}\sum_{i\in C}\frac{2\cdot \texttt{PREC}_i \cdot \texttt{REC}_i}{\texttt{PREC}_i + \texttt{REC}_i} \tag{9}$$

where $\texttt{REC}_i$ (Recall) is the ratio between correctly classified samples of class i and the total number of samples belonging to that class, and $\texttt{PREC}_i$ (Precision) is the ratio of correctly classified samples of class i to all samples predicted as belonging to that class.

Latent Space Structure: to evaluate representation quality, we analyze the latent space distribution of test biflows. A robust model should exhibit tight clustering for same-app biflows and distinct separation between different apps, ensuring clear decision boundaries and strong generalization. In detail, we employ two clustering-originated metrics, namely Davies-Bouldin Index (DBI) [51] and Silhouette score (SIL) [52], computed on the latent vectors $\boldsymbol{z}(\boldsymbol{x}) = f_\theta(\boldsymbol{x})$.

DBI captures the average similarity between each cluster and its most similar one, where similarity is defined as the ratio between intra-cluster dispersion and inter-cluster separation. Formally, the DBI is defined as:

$$\texttt{DB} = \frac{1}{|C|}\sum_{i\in C}\max_{j\neq i}\frac{\Delta_i + \Delta_j}{d_{i,j}} \tag{10}$$

where Δ_i and Δ_j are the mean intra-cluster distances of the classes i and j in the latent space, respectively, and $d_{i,j}$ is the distance between their centroids (viz. inter-cluster distance). Lower DBI values indicate better clustering—tight, well-separated groups—with 0 as the ideal case.

On the other hand, SIL combines the mean intra-cluster distance—i.e., the similarity between a sample and all other samples within the same cluster and the mean inter-cluster distance—i.e., the dissimilarity between the sample and the closest cluster it does not belong to. Formally, the SIL for the latent representation $\boldsymbol{z}(\boldsymbol{x})$ of a generic sample $\boldsymbol{x}$ is defined as:

$$s(\boldsymbol{z}(\boldsymbol{x})) = \frac{b(\boldsymbol{z}(\boldsymbol{x})) - a(\boldsymbol{z}(\boldsymbol{x}))}{\max\{a(\boldsymbol{z}(\boldsymbol{x})), b(\boldsymbol{z}(\boldsymbol{x}))\}} \tag{11}$$

where $a(\boldsymbol{z}(\boldsymbol{x}))$ is the mean distance between $\boldsymbol{z}(\boldsymbol{x})$ and all other samples in the same cluster in the latent space (viz. intra-cluster distance), and $b(\boldsymbol{z}(\boldsymbol{x}))$ is the smallest mean distance between $\boldsymbol{z}(\boldsymbol{x})$ and all samples of any other cluster in the latent space (viz. inter-cluster distance). The overall SIL is then computed as the sample-mean:

$$\texttt{SIL} = \frac{1}{N}\sum_{n=1}^{N} s(\boldsymbol{z}(\boldsymbol{x}_n)) \tag{12}$$

[4] https://man7.org/linux/man-pages/man1/strace.1.html

[5] All devices were equipped with the custom firmware LineageOS 17.1 corresponding to Android 10.

[6] Raw payload bytes and PL are mapped directly to dense vectors using learnable embedding layers (embedding size $e = 10$) without any pre-scaling.

where N is the total number of samples. SIL ranges in $[-1, 1]$, with 1 being the best value—indicating high compactness and clear separation—and -1 the worst, corresponding to samples assigned to the wrong cluster.

Model Calibration: we assess model calibration—i.e., how closely predicted confidence matches actual accuracy—using reliability diagrams [53] and the Expected Calibration Error (ECE) [54]. A model is *perfectly calibrated* if, whenever it predicts a confidence level $\hat{p}$, the true probability of correctness matches that confidence—that is, $\Pr\{\hat{y} = y \mid \hat{p}\} = \hat{p}$, where y and $\hat{y}$ are the true and predicted labels, respectively.

For each test sample n, we denote its true label as $y_{(n)}$, the predicted label as $\hat{y}_{(n)} \triangleq \arg\max_{1,\dots,|C|} p_i(n)$, and the predicted confidence as $\hat{p}(n) \triangleq \max_{1,\dots,|C|} p_i(n)$.

To estimate calibration, we partition predictions into M bins, evenly spaced across the confidence range from $[1/|C|, 1]$. For bin B_m, containing samples with confidence $\hat{p}$ in interval $I_m \triangleq (\frac{m-1}{M}; \frac{m}{M})$, we compute the "expected" accuracy:

$$\mathrm{acc}(B_m) = \frac{1}{|B_m|} \sum_{n \in B_m} 1(\hat{y}_{(n)} = y_{(n)}) \tag{13}$$

and the center-bin confidence $\hat{p}_m \triangleq \frac{m}{M} - \frac{1}{2M}$. A reliability diagram plots these two quantities for each bin and deviations from the diagonal indicate *over-* or *under-confidence*.

We summarize (mis)calibration using the ECE, defined as $E_{\hat{p}}\{|P\{\hat{\ell} = \ell \mid \hat{p}\} - \hat{p}|\}$, and approximated as:

$$\mathrm{ECE} \approx \sum_{m=1}^{M} \frac{|B_m|}{N} |\mathrm{acc}(B_m) - \mathrm{conf}(B_m)| \tag{14}$$

where N is the total number of test samples and $\mathrm{conf}(B_m) = \frac{1}{|B_m|} \sum_{n \in B_m} \hat{p}(n)$ represents the averaged confidence within mth bin. This metric succinctly captures how far the model's confidence diverges from reality.

Out-of-Distribution (OOD) Detection: to evaluate the model's ability to identify samples that do not belong to any of the known classes (i.e., seen during the training), we assess its performance in detecting OOD traffic. This capability is fundamental in open-set recognition, where the model is required to appropriately classify inputs from previously unseen classes as unknown [55], and is particularly relevant in open-world settings such as network traffic analysis, where novel or anomalous patterns may emerge.

In our case, the (anomaly) *scoring function* corresponds to the negated predicted confidence associated with the inferred class—i.e., $s(x) = -\hat{p}(x)$ where $\hat{p}(x) = \max_{c \in C} p_c(x)$. This score is justified since higher OOD scores (closer to zero) correspond to a lower confidence of the classifier for the known classes.

To measure OOD capabilities, we use the Area Under the Receiver Operating Characteristic Curve (AUROC) to discriminate between positive (viz. out-of-distribution) and negative (viz. in-distribution) samples. From a geometric standpoint, AUROC is defined as the area of the ROC curve in the TPR-FPR space:

$$\mathrm{AUROC} = \int_0^1 \mathrm{TPR}(\mathrm{FPR})\, d\mathrm{FPR} \tag{15}$$

where TPR(FPR) denotes the True Positive Rate (TPR) as a function of the False Positive Rate (FPR). The AUROC span over $[0, 1]$, with 1 indicating the perfect separation between In-Distribution (ID) and OOD samples, while 0.5 corresponds to random guessing.

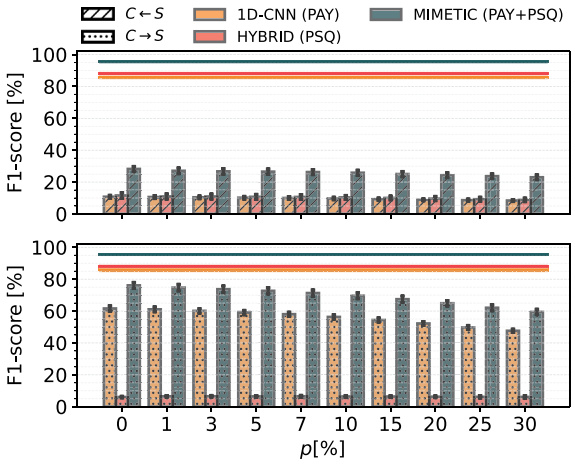


Fig. 5. Comparison of HYBRID, 1D-CNN, and MIMETIC-ENHANCED under DHiding (p=0%) and DHiding$_{\text{loss}}$ (p>0%) with packet-loss rate $p \in \{0, 1, 3, 5, 7, 10, 15, 20, 25, 30\}$. Each model is identified by its traffic type (i.e., PAY or PSQ), with performance on unperturbed samples shown by a solid line. Results are distinct for Client-to-Server ($C \rightarrow S$) and Server-to-Client ($C \leftarrow S$) directions of the biflow observed at the VP. Values refer to MIRAGE-24 and are reported as $mean \pm std.dev.$ over 10 folds.

5. Experimental results

In this section, we report our experimental evaluation [7] We start by comparing single-modal and multi-modal baseline architectures across unperturbed and perturbed scenarios (Section 5.1). Then, we evaluate the baseline multi-modal architecture by assessing: (*i*) its ability to generalize to perturbed inputs (Section 5.2); (*ii*) whether our training strategies improve generalization on clean (unperturbed) data (Section 5.3) and increase robustness to perturbations (Section 5.4). Next, we evaluate model calibration—how well predicted confidence matches actual accuracy (Section 5.5). We assess performance on OOD samples (Section 5.6) and analyze the impact of limited labeled data (Section 5.7). We further investigate the framework's generalizability across distinct model architectures (Section 5.8) and dissect the impact of individual perturbation types on overall TC performance (Section 5.9). Finally, we assess the framework's deployability (Section 5.10), offering insights into computational complexity and real-world feasibility Fig. 5.

5.1. Comparing the effectiveness and robustness of single- and multi-modal baselines

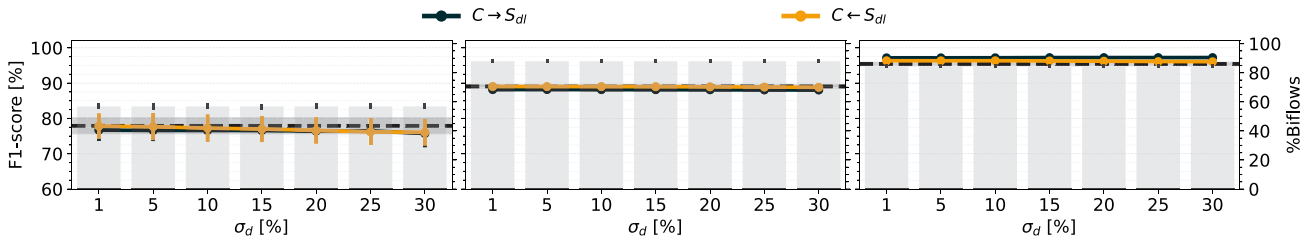
We first investigate the robustness of multimodal architectures compared to state-of-the-art single-modal models under limited traffic visibility. As a case study, we compare MIMETIC-ENHANCED with *two* representative *single-modal architectures*:

1. a 1D-CNN [11] fed with PAY input, and
2. HYBRID [13], a hybrid architecture composed by a 2D-CNN followed by a RNN and fed with PSQ input.

We focus on one representative perturbation from Section 3.2, noting that results are consistent across other cases but omitted for brevity.

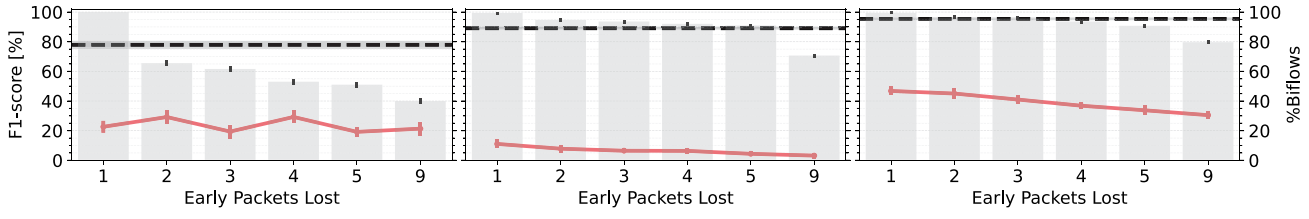
Section 5.1 shows the F1-score for DHidings scenarios when the VP is traversed solely by either the Client-to-Server ($C \rightarrow S$) or Server-to-Client ($C \leftarrow S$) path, across increasing packet-loss rates p. The results highlight *two main advantages* of MIMETIC-ENHANCED: (*i*) it outperforms

[7] Implementation uses PyTorch, numpy, and pandas, with scikit-learn for metrics (custom ECE) and the official SupCon code for contrastive losses [50]. Experiments ran on an Ubuntu 22.04 server with Intel i9-10900X, 256GB RAM, and 2×NVIDIA RTX A5000 (24GB) GPUs.



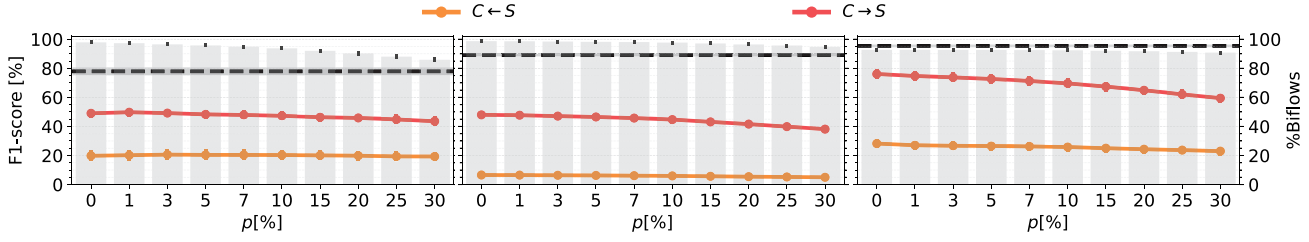
(a) VPN-16 (b) MIRAGE-19 (c) MIRAGE-24

Fig. 6. Performance evaluation of `Base` under `TJitter` as the standard deviation σ_d varies in $\{1, 5, 10, 15, 20, 25, 30\}\%$, with σ_d expressed as a percentage of the maximum IAT per dataset. Results are reported separately for scenarios where a delay is added only in the Client-to-Server ($C \rightarrow S_{dl}$) or Server-to-Client ($C \leftarrow S_{dl}$) direction. The *dashed-black line* reports the performance on the unperturbed test samples. Values are reported as $mean \pm std.dev.$ over 10 folds. *Gray bars* show the share of test samples on which performance has been computed.



(a) VPN-16 (b) MIRAGE-19 (c) MIRAGE-24

Fig. 7. Performance evaluation of `Base` under `PHiding` as the number of lost initial packets in the biflow varies in $\{1, 2, 3, 4, 5, 9\}$. The *dashed-black line* reports the performance on the unperturbed test samples. Values are reported as $mean \pm std.dev.$ over 10 folds. *Gray bars* show the share of test samples on which performance has been computed.



(a) VPN-16 (b) MIRAGE-19 (c) MIRAGE-24

Fig. 8. Performance evaluation of `Base` under `DHiding` (p=0%) and $\texttt{DHiding}_{\texttt{loss}}$ (p>0%) with packet-loss rate $p \in \{0, 1, 3, 5, 7, 10, 15, 20, 25, 30\}\%$.Results are reported separately for scenarios where only the Client-to-Server ($C \rightarrow S$) or Server-to-Client ($C \leftarrow S$) direction of the biflow is visible at the VP. The *dashed-black line* reports the performance on the unperturbed test samples. Values are reported as $mean \pm std.dev.$ over 10 folds. *Gray bars* show the share of test samples on which performance has been computed.

single-modal models on unperturbed data—up to a ≈10% `F1-score` gain over 1D-CNN—demonstrating the value of combining complementary inputs like PAY and PSQ; (*ii*) it maintains greater robustness under perturbations, with smaller performance drops as packet loss (p) increases. Among single-modal models, 1D-CNN and HYBRID behave similarly in the $C \leftarrow S$ scenario but diverge when $C \rightarrow S$, where 1D-CNN shows better resilience and a degradation trend closer to MIMETIC-ENHANCED.

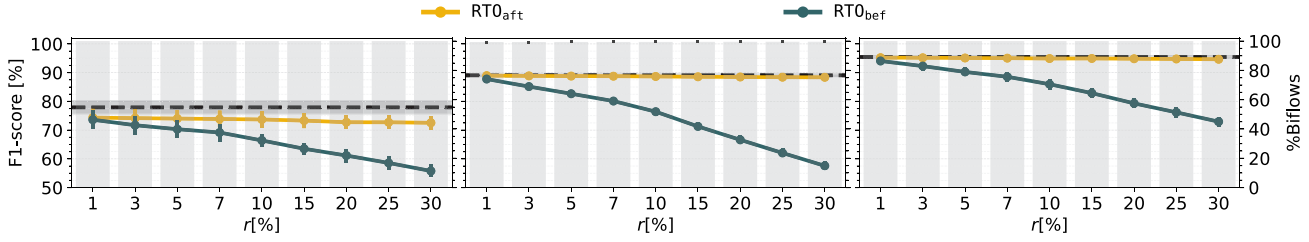
Takeaway:

Multimodal architectures outperform single-modal models by delivering higher classification accuracy in both ideal and perturbed settings, providing enhanced robustness when conditions degrade.

5.2. Baseline sensitivity to perturbations

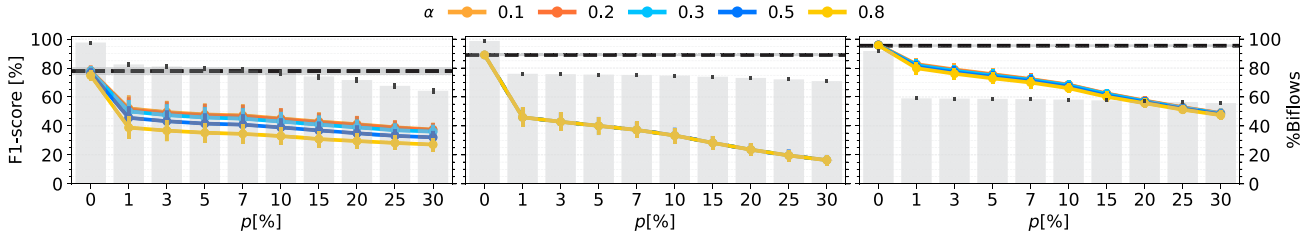
Herein, we assess the generalization ability of the baseline model (viz. `Base`)—trained on "clean" data—when exposed to perturbed test samples. To this end, in Figs. 6–10, we evaluated its performance—in terms of `F1-score`—under each of the perturbed scenarios described in Section 3.2. For each scenario, we defined a set of parameter values and applied the corresponding variants uniformly to all test samples. We also compare the resulting performance against that obtained on the unperturbed samples (highlighted with a dashed-black line). This analysis is useful for evaluating the sensitivity of the model to such controlled perturbations.

At a high level, we note that the model performance varies greatly depending on the type of perturbation and is almost consistent across all the datasets. On the one hand, perturbations that solely affect packet timing—such as `TJitter` (Fig. 6) and `VPC` (Fig. 10)—have a limited impact on classification performance, as they do not alter the sequence or structure of the biflows. Under `TJitter`, performance degradation remains limited as σ_d increases—e.g., up to −1% on MIRAGE-19. Interestingly, on MIRAGE-24, these perturbations lead to an improvement of up to +1.8%, particularly when the delay is introduced in the forward ($C \rightarrow S_{dl}$) direction. This suggests the robustness of the model against timing noise and that it likely leverages alternative, more stable traffic features. Similar findings are observed for `VPC` on MIRAGE-19 and MIRAGE-24 (Fig. 10b and c). Notably, performance remains stable across the entire variability range $\alpha \in [0, 1]$, which represents the fraction of RTT attributed to the Client-to-VP path. In contrast, the performance drop is more pronounced on VPN-16, indicating that the



(a) VPN-16 (b) MIRAGE-19 (c) MIRAGE-24

Fig. 9. Performance evaluation of `Base` under `RTOaft` and `RTObef` with retransmission rate $r \in \{1, 5, 10, 20, 30\}\%$ and timeout $T_{out} = 1$s. The *dashed-black line* reports the performance on the unperturbed test samples. Values are reported as $mean \pm std.dev.$ over 10 folds. *Gray bars* show the share of test samples on which performance has been computed.



(a) VPN-16 (b) MIRAGE-19 (c) MIRAGE-24

Fig. 10. Performance evaluation of `Base` under VPC (p=0%) and VPC$_{loss}$ (p>0%) with the packet-loss rate $p \in \{0, 1, 3, 5, 7, 10, 15, 20, 25, 30\}\%$ and $\alpha \in \{0.1, 0.2, 0.3, 0.5, 0.8\}$. The *dashed-black line* reports the performance on the unperturbed test samples. Values are reported as $mean \pm std.dev.$ over 10 folds. *Gray bars* show the share of test samples on which performance has been computed.

model may be more sensitive to variations in the relative position of the VP w.r.t. the endpoints on this dataset.

On the other hand, perturbations affecting biflow visibility through packet reordering, hiding, or duplication—e.g., `PHiding`, `DHiding`, `RTOaft`, and VPC$_{loss}$—result in more pronounced performance degradation, indicating that the model is more sensitive to structural than temporal variations.

Specifically, performance significantly drops even when only the first two packets of the biflow are lost (i.e., unseen) by the VP (Fig. 7). This holds especially on MIRAGE-19, where the drop is up to −81%. Conversely, on VPN-16 and MIRAGE-24, the drop—while still substantial—is smaller and increases more gradually as the number of unseen packets increases. These findings highlight that the model heavily relies on the earliest packets of the biflow, from which the PAY input is typically extracted. This is particularly critical in TLS traffic, where the initial packets often carry plaintext information (e.g., SNI, Cipher Suite, etc.) [16,45].

Similar findings are also observed when the VP is crossed only by one of the two paths connecting the client and the server (Fig. 8). Here, performance worsens mainly when the VP is crossed only by the backward path ($C \leftarrow S$), with drops up to −82% on MIRAGE-19. In this case, the impact remains relatively stable as the packet loss rate p increases up to 30%. This range was selected to rigorously cover the three severity classes identified in [56]: low (< 5%), middle (5–20%), and high (> 20%). When only the forward path ($C \rightarrow S$) is visible to the VP, a performance drop is observed across all datasets. The degradation is relatively stable on VPN-16, while it worsens more sharply with p on the others—e.g., from −19% to −36% on MIRAGE-24.

Finally, under RTO scenarios (Fig. 9), we observe that the impact strongly depends on where the loss occurs. In the `RTOaft` case, performance remains almost unchanged (i.e., <1%), even as the retransmission probability r increases across the considered range [1%, 30%] on MIRAGE-19 and MIRAGE-24. This is likely because all copies of a packet traverse the VP, preserving the expected packet sequence—especially for the initial packets of the biflow, which are particularly important for classification. Conversely, on VPN-16, a more noticeable degradation is observed, with `F1-score` dropping from −3.6% to −5.4% as r increases.

When losses occur *before* the VP (`RTObef`), performance drops significantly as r varies within [1%, 30%]. For instance, the `F1-score` drops to up to −32% on MIRAGE-19 as r reaches the maximum severity of 30%—a value consistent with retransmission rates observed for small flows in congested networks [57]. This finding is likely driven by the increased probability of observing out-of-order packets, since retransmitted packets may be observed at the VP in positions that differ from their expected order in the unperturbed biflow.

Takeaway:

The baseline model handles time-based perturbations fairly well, meaning it does not depend much on precise inter-packet timing. But structural perturbations—like dropping, hiding, or reordering packets—hurt performance more, especially if they affect the first few packets or if the VP sees traffic from only one direction. This shows the model is sensitive to how much of the biflow it can observe.

5.3. Improving effectiveness with augmentation and contrastive learning

Herein, we test whether the training strategies from Section 4.2 improve TC effectiveness under clean (unperturbed) conditions. We compare each variant to the baseline (`Base`) using both (*i*) TC performance and (*ii*) the structure of the learned latent space. For each variant, Table 6 reports the results in terms of macro `F1-score` (*i*), `DBI` (*ii*), and `SIL` (*ii*) on all datasets.

On the one hand, focusing on TC performance (i.e., `F1-score`), we observe that employing `Aug`, `SupCon`, or a combination of both in at least one of the training phases consistently improves results over `Base`.

On both MIRAGE-19 and MIRAGE-24, applying either `Aug` or `SupCon` in a single phase (viz. Aug$_{PT}$, Aug$_{FT}$, and SupCon$_{PT}$) consistently leads to performance improvements. Notably, using `Aug` only during FT (Aug$_{FT}$) typically results in greater improvements than applying it solely during PT (Aug$_{PT}$). The best performance is achieved when `Aug` is employed in both phases—e.g., +1.2% on MIRAGE-19. Interestingly, apply-

Table 6
Performance of $\texttt{Aug}_{\texttt{PT}}$, $\texttt{SimCon}_{\texttt{PT}}$, $\texttt{SupCon}_{\texttt{PT}}$, $\texttt{Aug}_{\texttt{PT}} \rightarrow \texttt{Aug}_{\texttt{FT}}$, $\texttt{SimCon}_{\texttt{PT}} \rightarrow \texttt{Aug}_{\texttt{FT}}$, and $\texttt{SupCon}_{\texttt{PT}} \rightarrow \texttt{Aug}_{\texttt{FT}}$ compared to Base on VPN-16, MIRAGE-19, and MIRAGE-24. Results are reported as *mean* ± *std.dev.* over 10 folds.↓ (resp. ↑): lower (resp. higher) is better. Baseline (Base) performance is shown in blue. For other variants, green (resp. red) highlights improvements (resp. degradations) w.r.t. Base, with darker shades indicating larger changes.

	VPN-16			MIRAGE-19			MIRAGE-24		
	F1-score [↑]	DBI [↓]	SIL [↑]	F1-score [↑]	DBI [↓]	SIL [↑]	F1-score[↑]	DBI [↓]	SIL [↑]
Base	77.91 ± 2.25	1.84 ± 0.09	0.13 ± 0.01	89.06 ± 0.44	1.92 ± 0.03	0.14 ± 0.00	95.42 ± 0.48	1.36 ± 0.04	0.27 ± 0.01
$\texttt{Aug}_{\texttt{PT}}$	79.82 ± 1.34	1.74 ± 0.05	0.14 ± 0.01	89.90 ± 0.39	1.78 ± 0.02	0.18 ± 0.01	96.16 ± 0.57	1.27 ± 0.03	0.33 ± 0.01
$\texttt{SimCon}_{\texttt{PT}}$	74.29 ± 1.06	2.16 ± 0.11	0.07 ± 0.01	88.72 ± 0.67	2.89 ± 0.08	−0.04 ± 0.01	92.40 ± 0.46	2.15 ± 0.1	0.05 ± 0.01
$\texttt{SupCon}_{\texttt{PT}}$	78.81 ± 1.62	1.51 ± 0.06	0.19 ± 0.01	88.95 ± 0.47	1.83 ± 0.31	0.08 ± 0.08	95.67 ± 0.55	1.11 ± 0.06	0.36 ± 0.02
$\texttt{Aug}_{\texttt{FT}}$	78.65 ± 1.21	1.91 ± 0.08	0.12 ± 0.01	90.20 ± 0.30	2.26 ± 0.03	0.09 ± 0.01	96.23 ± 0.59	1.61 ± 0.05	0.22 ± 0.01
$\texttt{Aug}_{\texttt{PT}} \rightarrow \texttt{Aug}_{\texttt{FT}}$	80.16 ± 1.93	1.70 ± 0.08	0.14 ± 0.01	90.26 ± 0.26	1.75 ± 0.03	0.19 ± 0.01	96.57 ± 0.49	1.18 ± 0.04	0.35 ± 0.01
$\texttt{SimCon}_{\texttt{PT}} \rightarrow \texttt{Aug}_{\texttt{FT}}$	77.47 ± 2.48	2.13 ± 0.11	0.07 ± 0.02	90.01 ± 0.42	3.17 ± 0.09	−0.05 ± 0.01	95.31 ± 0.54	2.54 ± 0.13	−0.00 ± 0.02
$\texttt{SupCon}_{\texttt{PT}} \rightarrow \texttt{Aug}_{\texttt{FT}}$	80.46 ± 1.16	1.47 ± 0.06	0.20 ± 0.01	90.08 ± 0.36	1.79 ± 0.08	0.12 ± 0.04	96.22 ± 0.41	1.01 ± 0.06	0.39 ± 0.02

ing Aug only during FT ($\texttt{Aug}_{\texttt{FT}}$) provides results comparable to those obtained when combining SupCon with Aug ($\texttt{SupCon}_{\texttt{PT}} \rightarrow \texttt{Aug}_{\texttt{FT}}$). Although these improvements may appear limited, it is important to note that the baseline already performs well under clean conditions (i.e., >89%), leaving little room for further improvements.

A different trend is observed on VPN-16, where the baseline performance is substantially lower (i.e., 77.9%). Here, the adoption of Aug and SupCon leads to more pronounced improvements (i.e., up to +2.6%). The best results are achieved when such strategies are applied consistently across both training phases ($\texttt{Aug}_{\texttt{PT}} \rightarrow \texttt{Aug}_{\texttt{FT}}$ and $\texttt{SupCon}_{\texttt{PT}} \rightarrow \texttt{Aug}_{\texttt{FT}}$), confirming the importance of combining these techniques in more challenging scenarios.

On the other hand, analysis of the latent space geometry shows that employing SupCon during PT *enhances both* the DBI and SIL, indicating better intra-class compactness and inter-class separation. These improvements are further amplified when Aug is applied during FT, suggesting that SupCon effectively structures the latent space while Aug helps refine class decision boundaries. Conversely, applying Aug in PT leads to only minor improvements, while using it in FT alone substantially worsens the representations learned during PT.

Interestingly, using SimCon during PT ($\texttt{SimCon}_{\texttt{PT}}$ and $\texttt{SimCon}_{\texttt{PT}} \rightarrow \texttt{Aug}_{\texttt{FT}}$) often results in marginal improvements or even performance drops on all datasets. This is likely due to its unsupervised nature, which promotes invariance across samples rather than class-discriminative features. These limitations are also reflected in the latent space geometry, underscoring its reduced effectiveness.

Takeaway:

Training strategies based on Aug and SupCon lead to consistent improvements in terms of both classification performance and latent space structure under clean (i.e., unperturbed) conditions. These gains are largest when the techniques are applied across both training phases—especially when combining SupCon in PT with Aug in FT. In contrast, SimCon consistently performs worse across all metrics.

5.4. Improving robustness with augmentation and contrastive learning

Herein, we investigate whether incorporating perturbation-based transformations into the training process—i.e., leveraging Aug and CL—can improve robustness to perturbed traffic conditions. Accordingly, we compare the performance of the best models identified in Section 5.3 (i.e., $\texttt{Aug}_{\texttt{PT}}$, $\texttt{SupCon}_{\texttt{PT}}$, $\texttt{Aug}_{\texttt{FT}}$, $\texttt{Aug}_{\texttt{PT}} \rightarrow \texttt{Aug}_{\texttt{FT}}$, and $\texttt{SupCon}_{\texttt{PT}} \rightarrow \texttt{Aug}_{\texttt{FT}}$) in terms of F1-score across the PHiding (Fig. 11), DHidings (Fig. 12), and RTOs (Fig. 13) conditions. We omit TJitter as it has a negligible impact on performance, and VPC for brevity, since its results are comparable to those of RTObef. We report results across all datasets and parameter settings, with the baseline (Base) performance under the same perturbations included for comparison. A solid black line shows the Base model's accuracy on unperturbed samples, serving as an upper bound.

Overall, we observe that the models trained leveraging Aug and SupCon exhibit significantly higher robustness—i.e., a lower gap against the upper-bound—compared to Base across all considered scenarios, with $\texttt{SupCon}_{\texttt{PT}} \rightarrow \texttt{Aug}_{\texttt{FT}}$ and $\texttt{Aug}_{\texttt{PT}} \rightarrow \texttt{Aug}_{\texttt{FT}}$ consistently showing the highest robustness to perturbations.

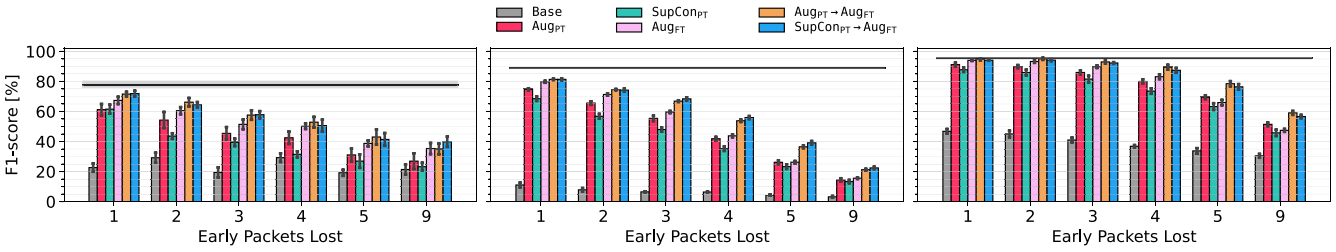
Focusing on PktHid (Fig. 11), we observe that on MIRAGE-19 and VPN-16, Base shows a drastic drop when the first two packets of the biflow are lost (e.g., up to −81% of F1-score on MIRAGE-19). Conversely, both $\texttt{SupCon}_{\texttt{PT}} \rightarrow \texttt{Aug}_{\texttt{FT}}$ and $\texttt{Aug}_{\texttt{PT}} \rightarrow \texttt{Aug}_{\texttt{FT}}$ undergo a much smaller degradation (i.e., <15% compared to the unperturbed case). Although the degradation increases as more initial packets are lost, these models consistently outperform Base under such perturbation. Notably, on MIRAGE-24, the performance drop remains relatively small (i.e., <6%) even when the first four packets are lost.

Examining the DHidings cases (Fig. 12), we observe that the Aug- and SupCon-based approaches exhibit much lower sensitivity to both the observed directional flow (i.e., C→S or C←S) and the packet-loss probability p, compared to Base. For instance, Base experiences a substantial performance drop of up to −84% of F1-score when the VP observed only the $C \leftarrow S$ traffic with $p=30\%$ on the MIRAGE-19. In contrast, with $\texttt{Aug}_{\texttt{PT}} \rightarrow \texttt{Aug}_{\texttt{FT}}$ and $\texttt{SupCon}_{\texttt{PT}} \rightarrow \texttt{Aug}_{\texttt{FT}}$, this drop is limited to at most −20% and is further reduced to −9% on MIRAGE-24.

Finally, looking at RTOs (Fig. 13), we observe that while Base shows distinct behaviors between the RTOaft and RTObef scenarios, this does not hold across the new variants. Specifically, under RTOaft, Base exhibits very limited performance degradation as the retransmission rate r increases (e.g., <1% on both MIRAGE-19 and MIRAGE-24). Conversely, the Aug- and CL-based variants, in particular $\texttt{Aug}_{\texttt{PT}} \rightarrow \texttt{Aug}_{\texttt{FT}}$ and $\texttt{SupCon}_{\texttt{PT}} \rightarrow \texttt{Aug}_{\texttt{FT}}$, not only show negligible degradation but even achieve slight improvements of up to +2% on VPN-16 in some cases. Lastly, in the case of RTObef, Base shows a performance drop of up to −22% on MIRAGE-24 when $r=30\%$. In contrast, $\texttt{Aug}_{\texttt{PT}} \rightarrow \texttt{Aug}_{\texttt{FT}}$ and $\texttt{SupCon}_{\texttt{PT}} \rightarrow \texttt{Aug}_{\texttt{FT}}$ show a significantly lower drop of only −1% on the same dataset.

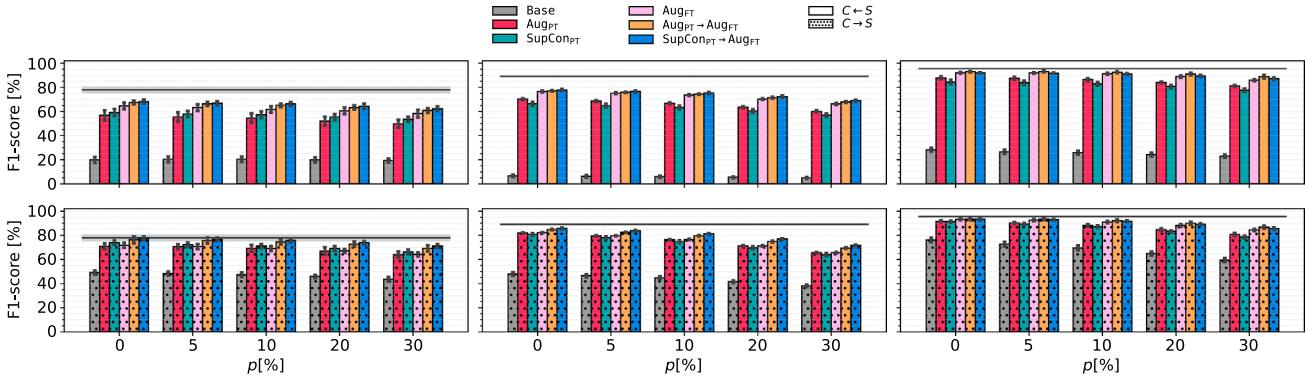
Takeaway:

While Base suffers substantial performance degradation under perturbations and limited visibility, integrating perturbation-based transformations during the training process greatly improves model robustness across all the considered scenarios. Notably, the $\texttt{Aug}_{\texttt{PT}} \rightarrow \texttt{Aug}_{\texttt{FT}}$ *and* $\texttt{SupCon}_{\texttt{PT}} \rightarrow \texttt{Aug}_{\texttt{FT}}$ *variants consistently exhibit the highest robustness across diverse scenarios and datasets.*



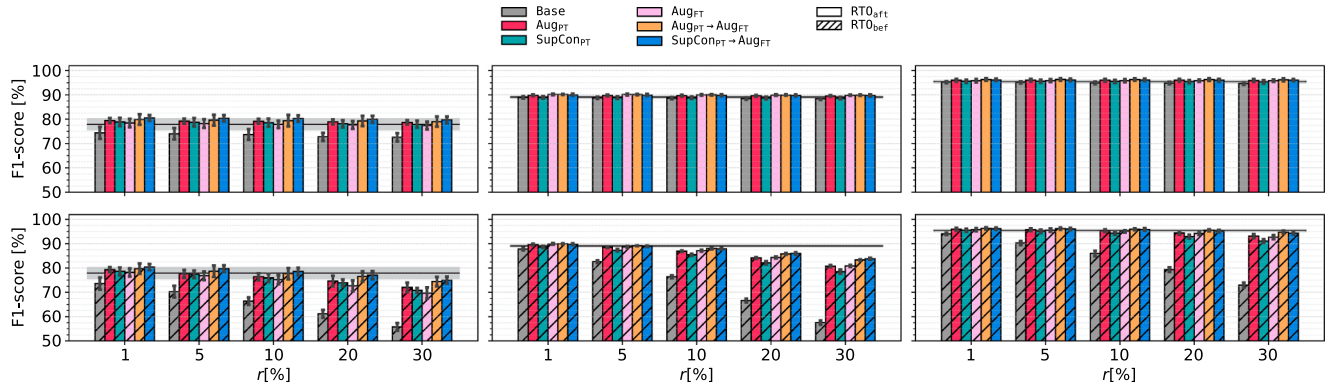
(a) VPN-16 (b) MIRAGE-19 (c) MIRAGE-24

Fig. 11. Comparison of Base, Aug$_{PT}$, SupCon$_{PT}$, Aug$_{FT}$, Aug$_{PT}$ → Aug$_{FT}$, and SupCon$_{PT}$ → Aug$_{FT}$ under PHiding as the number of lost initial packets in the biflow varies in $\{1, 2, 3, 4, 5, 9\}$. The *black line* reports the performance on the unperturbed test samples. Values are reported as $mean \pm std.dev.$ over 10 folds.



(a) VPN-16 (b) MIRAGE-19 (c) MIRAGE-24

Fig. 12. Comparison of Base, Aug$_{PT}$, SupCon$_{PT}$, Aug$_{FT}$, Aug$_{PT}$ → Aug$_{FT}$, and SupCon$_{PT}$ → Aug$_{FT}$ under DHiding (p=0%) and DHiding$_{loss}$ (p>0%) with packet-loss rate $p \in \{0, 5, 10, 20, 30\}$. Results are reported separately for scenarios where only the Client-to-Server ($C \rightarrow S$) or Server-to-Client ($C \leftarrow S$) direction of the biflow is visible at the VP. The *black line* reports the performance on the unperturbed test samples. Values are reported as $mean \pm std.dev.$ over 10 folds.



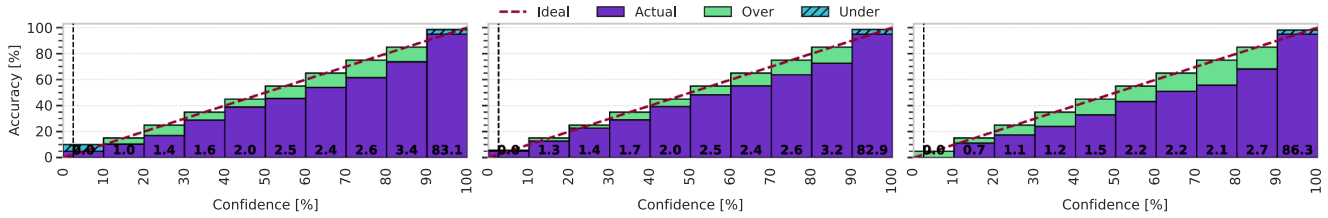
(a) VPN-16 (b) MIRAGE-19 (c) MIRAGE-24

Fig. 13. Comparison of Base, Aug$_{PT}$, SupCon$_{PT}$, Aug$_{FT}$, Aug$_{PT}$ → Aug$_{FT}$, and SupCon$_{PT}$ → Aug$_{FT}$ under RTO scenarios with retransmission rate $r \in \{1, 5, 10, 20, 30\}$ and timeout T_{out} = 1s. The *black line* reports the performance on the unperturbed test samples. Values are reported as $mean \pm std.dev.$ over 10 folds.

5.5. Assessing model reliability

In this section, we analyze the calibration of the models using the ECE metric. To this end, Table 7 reports the results obtained for Base, Aug$_{PT}$, SupCon$_{PT}$, Aug$_{FT}$, Aug$_{PT}$ → Aug$_{FT}$, and SupCon$_{PT}$ → Aug$_{FT}$ across all datasets.

Overall, we note that calibration varies significantly across datasets and models. Specifically, on MIRAGE-19 and MIRAGE-24, all models achieve good calibration, with ECE values typically below 3%. In particular, on MIRAGE-24, the ECE consistently remains below 1%, indicating well-calibrated models in this scenario, including the baseline. Conversely, calibration on VPN-16 is generally worse, with ECE values around 6%–7% for all models. This is consistent with the lower F1-score of Base on VPN-16 compared to the other datasets, even in the clean setting, as previously discussed. Comparing the different strategies, we note that incorporating perturbations solely in PT consistently improves calibration (i.e., Aug$_{PT}$ or SupCon$_{PT}$). For instance, SupCon$_{PT}$ achieves the lowest ECE on MIRAGE-19 and MIRAGE-24, outperforming Base and all other variants. Conversely, using transformations only in the FT phase (i.e., Aug$_{FT}$) consistently worsens calibration across all datasets compared to Base. This is likely due to applying perturbations only during FT, which introduces input noise without significantly altering the feature space, as only the final layers—operating on fixed features from the frozen branch layers—are updated. Interestingly, for SupCon$_{PT}$ → Aug$_{FT}$,



(a) Base (b) $\text{SupCon}_{\text{PT}}$ (c) $\text{SupCon}_{\text{PT}} \rightarrow \text{Aug}_{\text{FT}}$

Fig. 14. Reliability diagrams of Base (a), $\text{SupCon}_{\text{PT}}$ (b), and $\text{SupCon}_{\text{PT}} \rightarrow \text{Aug}_{\text{FT}}$ (c) on MIRAGE-19. Confidence is divided into 10 bins, and it is $\geq \frac{1}{|C|}$ (vertical dashed line), with $|C|$ being the number of classes. Over and under gaps represent an over-confident (optimistic) and under-confident (pessimistic) miscalibration pattern, respectively. The number at the bottom of each bar reports the share of samples within the corresponding bin.

Table 7
Calibration (ECE) of Aug_{PT}, Aug_{PT}, SupCon, $\text{Aug}_{\text{PT}} \rightarrow \text{Aug}_{\text{FT}}$, and $\text{SupCon}_{\text{PT}} \rightarrow \text{Aug}_{\text{FT}}$ compared with Base on VPN-16, MIRAGE-19, and MIRAGE-24. Results are reported as *mean* ± *std.dev.* over 10 folds.ECE is reported as a percentage (lower is better). Baseline (Base) performance is shown in blue. For the new variants, green (resp. red) highlights improvements (resp. degradations) w.r.t. Base, with darker shades indicating larger changes.

	VPN-16	MIRAGE-19	MIRAGE-24
Base	5.75 ± 2.05	2.48 ± 0.34	0.75 ± 0.12
Aug_{PT}	5.78 ± 1.46	2.53 ± 0.38	0.64 ± 0.20
$\text{SupCon}_{\text{PT}}$	5.53 ± 0.87	2.28 ± 0.41	0.63 ± 0.11
Aug_{FT}	7.05 ± 1.09	4.06 ± 0.23	0.93 ± 0.16
$\text{Aug}_{\text{PT}} \rightarrow \text{Aug}_{\text{FT}}$	6.53 ± 1.00	3.29 ± 0.34	0.82 ± 0.17
$\text{SupCon}_{\text{PT}} \rightarrow \text{Aug}_{\text{FT}}$	6.47 ± 1.95	3.20 ± 0.23	0.81 ± 0.15

Table 8
OOD detection in terms of *AUROC (%)*. When training on MIRAGE-19 (resp. MIRAGE-24), test samples from MIRAGE-19 (resp. MIRAGE-24) are considered ID, whereas those from MIRAGE-24 (resp. MIRAGE-19) are treated as OOD. *AUROC* (higher is better) is reported as *mean* ± *std.dev.* over 10 folds. Baseline (Base) performance is shown in blue, while the best-performing approach in each configuration (i.e., column) is highlighted in green.

	MIRAGE-19 (ID)	MIRAGE-24 (ID)
Base	85.33 ± 1.17	91.97 ± 1.31
Aug_{PT}	91.46 ± 1.24	94.40 ± 0.53
$\text{SupCon}_{\text{PT}}$	86.82 ± 1.76	93.46 ± 2.07
Aug_{FT}	90.42 ± 1.32	92.78 ± 0.83
$\text{Aug}_{\text{PT}} \rightarrow \text{Aug}_{\text{FT}}$	91.25 ± 0.78	94.99 ± 0.44
$\text{SupCon}_{\text{PT}} \rightarrow \text{Aug}_{\text{FT}}$	91.03 ± 0.92	95.27 ± 0.61
	MIRAGE-24 (OOD)	MIRAGE-19 (OOD)

the benefits of SupCon in PT are partially offset by the use of augmentation in FT. As a result, $\text{SupCon}_{\text{PT}} \rightarrow \text{Aug}_{\text{FT}}$ shows worse calibration than $\text{SupCon}_{\text{PT}}$ alone, although it still outperforms $\text{Aug}_{\text{PT}} \rightarrow \text{Aug}_{\text{FT}}$.

In Fig. 14, we report the reliability diagrams for Base, $\text{SupCon}_{\text{PT}}$, and $\text{SupCon}_{\text{PT}} \rightarrow \text{Aug}_{\text{FT}}$ on MIRAGE-19. For Base, the sample distribution across bins is highly skewed, with the majority of predictions falling in the last bin (i.e., $\hat{p} > 90\%$). In this range, the model exhibits slight under-confidence. For lower-confidence bins, containing $< 20\%$ of samples, overconfidence increases with $\hat{p}$.

For $\text{SupCon}_{\text{PT}}$, the distribution is nearly identical compared to Base, but over-confidence is reduced, especially in the range $[0, 30]\%$. For $\text{SupCon}_{\text{PT}} \rightarrow \text{Aug}_{\text{FT}}$, predicted confidence increases overall, with samples shifting toward high-confidence bins. Specifically, overconfidence increases in bins with confidence $< 90\%$, but its impact is reduced as more samples shift into the last bin. Since the behavior in this bin remains similar to Base, the share of samples with reliable predictions increases, leading to improved F1-score (see Section 5.3).

Takeaway:

Calibration varies with the training strategy but shows consistent patterns across datasets. PT with SupCon consistently improves model calibration. Conversely, FT with Aug increases overconfidence in low-confidence predictions. For $\text{SupCon}_{\text{PT}} \rightarrow \text{Aug}_{\text{FT}}$, this effect is mitigated by a shift toward higher-confidence predictions, improving both reliability and classification performance.

5.6. Benefits in OOD setups

Herein, we evaluate the ability of models to detect OOD traffic samples—i.e., samples from classes not seen during training.

Accordingly, Table 8 compares the AUROC of Base with the best variants from Section 5.3 on the MIRAGE-19 and MIRAGE-24 datasets. To test generalization to truly unseen classes, we discarded overlapping classes between the training and test sets (i.e., Messenger, Skype, and Telegram). We focused on MIRAGE-19 and MIRAGE-24, which share the same collection setup, ensuring that observed performance differences are due to intrinsic traffic characteristics rather than artifacts from differing acquisition conditions. Specifically, when training on MIRAGE-19 (resp. MIRAGE-24), test samples from MIRAGE-19 (resp. MIRAGE-24) are treated as ID, while those from the other dataset are considered OOD.

The results show that models trained with Aug or CL consistently achieve higher AUROC values compared to the baseline, indicating better OOD detection capabilities.

When MIRAGE-24 is considered as OOD, the greatest improvement (i.e., +6% of AUROC) is obtained by using only augmentation during PT (i.e., Aug_{PT}). In the reverse scenario, using augmentation or SupCon in PT (i.e., Aug_{PT} or $\text{SupCon}_{\text{PT}}$) already provides benefits (e.g., +2.74% with Aug_{PT}), which are further enhanced by incorporating augmentation also in FT (e.g., +3.3% with $\text{SupCon}_{\text{PT}} \rightarrow \text{Aug}_{\text{FT}}$).

Notably, although MIRAGE-24 contains only 20 classes compared to the 40 in MIRAGE-19, the baseline model already exhibits significantly higher OOD detection capabilities when trained on MIRAGE-24. This suggests that OOD detection performance is influenced not only by the number of classes but also by the intrinsic characteristics of the dataset.

Takeaway:

Incorporating perturbation-based transformations into the learning process, especially during both the pre-training and fine-tuning phases, substantially improves also OOD detection capabilities of the models.

5.7. Advantages in data-constrained training

In this section, we analyze the ability of models to generalize when trained with a limited amount of data. This scenario is particularly useful in real-world contexts where data scarcity often occurs due to privacy issues, collection costs, or other limitations. Hence, we systematically

Table 9
Performance of Base, $\text{Aug}_{\text{PT}} \rightarrow \text{Aug}_{\text{FT}}$, and $\text{SupCon}_{\text{PT}} \rightarrow \text{Aug}_{\text{FT}}$ on MIRAGE-19, VPN-16, and MIRAGE-24 when reducing the training set size. δ represents the percentage of the training set used for training. Results are reported as *mean* ± *std.dev.* over 10 folds. F1-score is reported as a percentage. ↓ (resp. ↑): lower (resp. higher) is better.

δ [%]	Config.	VPN-16			MIRAGE-19			MIRAGE-24		
		F1-score [↑]	DBI [↓]	SIL [↑]	F1-score [↑]	DBI [↓]	SIL [↑]	F1-score [↑]	DBI [↓]	SIL [↑]
100	Base	77.91 ±2.25	1.84 ±0.09	0.13 ±0.01	89.06 ±0.44	1.92 ±0.03	0.14 ±0.00	95.42 ±0.48	1.36 ±0.04	0.27 ±0.01
30	Base	69.30 ±2.18	2.31 ±0.12	0.09 ±0.01	82.89 ±0.69	2.29 ±0.03	0.10 ±0.01	91.54 ±1.11	1.53 ±0.06	0.21 ±0.01
	$\text{Aug}_{\text{PT}} \rightarrow \text{Aug}_{\text{FT}}$	70.10 ±2.25	2.14 ±0.11	0.09 ±0.01	84.60 ±0.51	2.04 ±0.03	0.14 ±0.00	93.73 ±0.80	1.34 ±0.06	0.29 ±0.02
	$\text{SupCon}_{\text{PT}} \rightarrow \text{Aug}_{\text{FT}}$	72.34 ±2.17	1.97 ±0.10	0.12 ±0.01	84.03 ±0.51	1.72 ±0.02	0.20 ±0.01	93.28 ±1.12	1.11 ±0.06	0.37 ±0.01
10	Base	57.73 ±3.18	2.94 ±0.13	0.02 ±0.01	73.34 ±0.91	2.69 ±0.05	0.07 ±0.01	83.89 ±1.60	1.77 ±0.06	0.16 ±0.01
	$\text{Aug}_{\text{PT}} \rightarrow \text{Aug}_{\text{FT}}$	57.46 ±2.34	2.80 ±0.10	0.03 ±0.01	76.00 ±0.50	2.47 ±0.04	0.10 ±0.00	88.07 ±1.43	1.58 ±0.05	0.22 ±0.01
	$\text{SupCon}_{\text{PT}} \rightarrow \text{Aug}_{\text{FT}}$	58.73 ±2.90	2.93 ±0.19	0.01 ±0.02	75.84 ±0.40	2.25 ±0.05	0.12 ±0.01	87.66 ±1.80	1.36 ±0.07	0.30 ±0.01
5	Base	50.09 ±1.85	3.52 ±0.34	−0.02 ±0.01	64.87 ±1.84	3.15 ±0.06	0.04 ±0.01	77.97 ±2.29	1.99 ±0.05	0.13 ±0.01
	$\text{Aug}_{\text{PT}} \rightarrow \text{Aug}_{\text{FT}}$	49.26 ±3.01	3.42 ±0.26	−0.02 ±0.01	68.45 ±0.99	2.88 ±0.06	0.07 ±0.01	81.05 ±1.83	1.82 ±0.03	0.17 ±0.02
	$\text{SupCon}_{\text{PT}} \rightarrow \text{Aug}_{\text{FT}}$	45.97 ±6.98	3.82 ±0.47	−0.06 ±0.02	67.96 ±1.99	2.70 ±0.10	0.06 ±0.02	82.72 ±2.06	1.62 ±0.07	0.24 ±0.01

reduce the training set and evaluate the impact on models performance. The primary focus is to assess whether Aug and CL can still yield benefits under these challenging conditions.

To this end, Table 9 reports the performance of Base, $\text{Aug}_{\text{PT}} \rightarrow \text{Aug}_{\text{FT}}$, and $\text{SupCon}_{\text{PT}} \rightarrow \text{Aug}_{\text{FT}}$ in terms of F1-score, DBI, and SIL across all datasets when reducing (viz. downsampling) the original training set to $\delta \in \{5, 10, 30\}\%$. The downsampling was performed in a stratified manner to maintain the relative class proportions within each dataset. For comparison, we also report the performance of Base using the full training set ($\delta = 100\%$).

As expected, reducing the training set leads to a significant performance degradation across all datasets, especially in terms of F1-score (e.g., −24% for Base at $\delta = 5\%$ on MIRAGE-19). Nevertheless, $\text{Aug}_{\text{PT}} \rightarrow \text{Aug}_{\text{FT}}$ and $\text{SupCon}_{\text{PT}} \rightarrow \text{Aug}_{\text{FT}}$ variants consistently outperform Base, especially as δ decreases on MIRAGE-19 and MIRAGE-24. For instance, $\text{SupCon}_{\text{PT}} \rightarrow \text{Aug}_{\text{FT}}$ outperforms Base of up to +4.8% of F1-score when using only 5% of MIRAGE-24, along with lower DBI (i.e., 1.6 vs. 2) and higher SIL (i.e., 0.24 vs. 0.13) values. Notably, as δ increases to 30%, both $\text{Aug}_{\text{PT}} \rightarrow \text{Aug}_{\text{FT}}$ and $\text{SupCon}_{\text{PT}} \rightarrow \text{Aug}_{\text{FT}}$ nearly recover the performance gap with Base with the entire training set (e.g., ≈−2% of F1-score on MIRAGE-24), with $\text{SupCon}_{\text{PT}} \rightarrow \text{Aug}_{\text{FT}}$ constantly achieving lower DBI and higher SIL values.

These results confirm that $\text{Aug}_{\text{PT}} \rightarrow \text{Aug}_{\text{FT}}$ and $\text{SupCon}_{\text{PT}} \rightarrow \text{Aug}_{\text{FT}}$ perform better under limited data conditions, showing improved performance on MIRAGE-19 and MIRAGE-24, along with better generalization despite fewer training samples. Conversely, on VPN-16, these trends are less marked, and in some cases, Base slightly outperforms both approaches in terms of F1-score—e.g., 50% for Base vs. 49% for $\text{Aug}_{\text{PT}} \rightarrow \text{Aug}_{\text{FT}}$ when $\delta = 5\%$. However, $\text{Aug}_{\text{PT}} \rightarrow \text{Aug}_{\text{FT}}$ and $\text{SupCon}_{\text{PT}} \rightarrow \text{Aug}_{\text{FT}}$ still achieve better DBI and SIL values.

Takeaway:

Our findings reveal that while downsampling inevitably reduces performance, particularly in terms of F1-score, the use of Aug and SupCon strategies can partially mitigate these effects on MIRAGE-19 *and* MIRAGE-24. *However, this does not hold for* VPN-16, *where, in some cases, the baseline outperforms the other approaches in F1-score, despite a worse latent space regularization in terms of both DBI and SIL.*

5.8. Assessing framework's generalizability

In this section, we investigate whether the proposed perturbation-aware learning approach can serve as a general, "model-agnostic" mechanism to robustify existing traffic classifiers, regardless of their underlying DL architecture.

Table 10
Peformance (F1-score) of single-modal—1D-CNN (PAY) and HYBRID (PSQ)—and multi-modal—APPNET, MIMETIC, and MIMETIC-ENHANCED (PAY and PSQ)—architectures on MIRAGE-24. The results compare the standard training procedure (Base) with the proposed Aug strategies under both clean traffic and critical perturbation scenarios. The results for MIMETIC and MIMETIC-ENHANCED refer to the $\text{Aug}_{\text{PT}} \rightarrow \text{Aug}_{\text{FT}}$ strategy, while other (single-modal) models to the Aug_{PT} one. Green highlighted the best configuration for each architecture. Results are reported as *mean* ± *std.dev.* over 10 folds.

Arch.	Var.	Clean	Test Perturbation ⊕		
			PHiding ($LP = 5$)	$\text{DHiding}_{\text{loss}}$ ($C \leftarrow S, p = 30\%$)	RTObef ($r = 30\%$)
1D-CNN	Base	85.78 ± 1.10	12.16 ± 1.36	8.39 ± 0.60	59.88 ± 0.89
	Aug_{PT}	87.30 ± 1.29	32.62 ± 1.21	64.39 ± 1.01	80.32 ± 0.77
HYBRID	Base	88.09 ± 0.80	21.32 ± 1.25	8.82 ± 1.11	8.74 ± 0.80
	Aug_{PT}	92.08 ± 0.77	48.86 ± 3.83	56.91 ± 5.58	63.35 ± 4.34
APPNET	Base	92.30 ± 0.69	14.50 ± 1.95	8.16 ± 1.13	66.92 ± 1.02
	Aug_{PT}	92.67 ± 0.76	41.13 ± 3.27	71.90 ± 2.36	87.00 ± 0.86
MIMETIC	Base	93.28 ± 0.38	18.53 ± 1.30	8.86 ± 0.64	57.45 ± 0.80
	$\text{Aug}_{\text{PT}} \rightarrow \text{Aug}_{\text{FT}}$	95.28 ± 0.44	58.47 ± 1.50	81.49 ± 1.10	88.80 ± 0.62
MIMETIC-ENHANCED	Base	95.42 ± 0.48	33.76 ± 1.82	22.99. ± 1.49	72.94 ± 1.04
	$\text{Aug}_{\text{PT}} \rightarrow \text{Aug}_{\text{FT}}$	96.57 ± 0.49	78.49 ± 1.71	88.87 ± 1.42	94.76 ± 0.57

Clean: unperturbed traffic samples. *LP*: number of lost initial packets in the biflow. *p*: packet-loss rate. $C \leftarrow S$: only Server-to-Client direction of the biflow is visible at the VP. *r*: retransmission rate (%) with $T_{out} = 1$ s.

To this end, we extended our analysis to a diverse set of state-of-the-art baselines, applying the proposed training framework to both single-modal and external multimodal architectures. Specifically, we selected 1D-CNN [11] and HYBRID [13] as representative single-modal baselines operating on PAY and PSQ inputs, respectively. To validate the framework's transferability to different multimodal architectures, we also included MIMETIC [15] and APPNET [18], leveraging both traffic inputs. [8]

Table 10 details the comparative results on the MIRAGE-24 dataset. For each model, we report the F1-score obtained without integrating perturbations (viz. Base) against that achieved using the proposed perturbation-aware strategies. Specifically, we applied Aug_{PT} for 1D-CNN, HYBRID, and APPNET, whereas we employed $\text{Aug}_{\text{PT}} \rightarrow \text{Aug}_{\text{FT}}$ for MIMETIC and MIMETIC-ENHANCED. The evaluation considers both clean traffic—monitoring potential degradation on nominal data—and critical perturbation scenarios, corresponding to specific parameter settings that induce critical degradations.

The results highlight that integrating perturbations into the learning process yields consistent and substantial improvements in both scenarios. On the one hand, the proposed approach enhances the models' capability to discern classes in nominal (viz. clean) scenarios, particularly for architectures leveraging only PSQ input (i.e., +4% in F1-score). On the other hand, it improves model robustness when classifying perturbed traffic characterized by packet retransmission (i.e., RTObef) or partial visibility (i.e., PktHid and DirHidLoss). Finally,

[8] APPNET considers only PL as PSQ input.

Table 11
Performance (F1-score) of MIMETIC-ENHANCED ($\text{Aug}_{\text{PT}} \rightarrow \text{Aug}_{\text{FT}}$) trained with individual perturbation types on MIRAGE-24 on both clean traffic and under critical perturbation scenarios. The *Baseline* ($\mathcal{T} = \emptyset$) and the *Complete Model* ($\mathcal{T} = \text{ALL}$) are included as lower and upper bounds, respectively. For each column, the top two performing configurations are highlighted in green, with darker shades indicating higher values. Entries on the main diagonal represent the performance when the model is evaluated against the same perturbation "seen" during training. Results are reported as $mean \pm std.dev.$ over 10 folds.

			Test Perturbation				
	$\mathcal{T}$	Clean	TJitter ($C \leftarrow S_{dl}, \sigma_d = 30\%$)	PHiding ($LP = 5$)	$\text{DHiding}_{\text{loss}}$ ($C \leftarrow S, p = 30\%$)	RTObef ($r = 30\%$)	VPC_{loss} ($\alpha = 0.8, p = 30\%$)
	$\emptyset$	95.42 ±0.48	96.07 ±0.53	33.76 ±1.82	22.99 ±1.49	72.94 ±1.04	71.98 ±0.50
Train Perturb.	TJitter	95.31 ±0.50	96.32 ±0.60	33.30 ±1.46	23.41 ±1.45	73.51 ±0.91	72.30 ±1.56
	PHidings	96.26 ±0.55	96.62 ±0.72	73.96 ±0.82	69.66 ±1.25	88.20 ±1.47	91.02 ±1.08
	DHidings	96.81 ±0.41	97.32 ±0.52	75.14 ±1.10	92.41 ±0.73	92.28 ±0.66	92.97 ±1.05
	RTOs	96.02 ±0.65	96.69 ±0.63	57.59 ±1.79	63.71 ±3.32	94.36 ±0.66	90.35 ±1.55
	VPCs	95.63 ±0.55	96.03 ±0.65	53.91 ±2.11	43.86 ±1.90	80.91 ±0.82	84.87 ±1.68
	ALL	96.57 ±0.49	97.39 ±0.49	78.49 ±1.71	88.87 ±1.42	94.76 ±0.57	94.38 ±0.92

while multi-modal architectures consistently outperform single-modal ones, our perturbation-based strategy significantly bridges this gap on clean traffic. Notably, HYBRID performs almost on par with APPNET (with a difference of $< 0.5\%$ in F1-score), despite relying solely on the PSQ modality. Overall, among the multi-modal architectures considered, MIMETIC-ENHANCED proves to be the best performing model.

Takeaway:

Integrating perturbations into the training loop yields consistent improvements on both clean and perturbed scenarios across diverse architectures without requiring structural modifications. This confirms the model-agnostic nature of our approach. Moreover, while multi-modal architectures consistently outperform single-modal ones, our perturbation-based strategy significantly bridges this gap on clean traffic.

5.9. Assessing the contribution of specific perturbations

In this section, we disentangle the specific contribution of each perturbation type (defined in Section 3.2) to the model's overall performance. In contrast to the comprehensive perturbation-aware strategy—which aggregates all transformations into a unified pipeline—we adopted a targeted ablation approach. Specifically, we trained distinct model variants, each exposed to only one specific class of perturbation during the learning phase (e.g., leveraging only DHiding-based augmentation).

Table 11 reports the results of the $\text{Aug}_{\text{PT}} \rightarrow \text{Aug}_{\text{FT}}$ variant on MIRAGE-24. For each perturbation type considered during training (see column $\mathcal{T}$), we report the F1-score on both clean traffic and under critical perturbation scenarios, corresponding to the parameter settings that induced the maximum degradation. For comparison, we also report the performance of the Base model—trained without perturbations ($\mathcal{T} = \emptyset$)—and the model obtained by considering all perturbations ($\mathcal{T} = \text{ALL}$).

As expected, across all scenarios, integrating a specific perturbation into the training loop confers robustness against that specific perturbation (see main diagonal). Remarkably, none of the perturbations introduced during training compromises the model's ability to classify unperturbed samples. In fact, except for perturbations that mainly alter packet timing (i.e., TJitter and VPC), those modifying the biflow structure yield a slight improvement on unperturbed traffic compared to Base (up to +1.4% F1-score). Notably, DHidings lead to consistent improvements in both nominal and perturbed scenarios. This suggests that integrating these perturbation types enhances robustness against other categories as well, effectively reducing the model's dependence on complete packet sequences.

Consequently, by exposing the network to structural degradations such as DHidings, the model is compelled to reconstruct its classification logic from partial information. This indicates that the learned patterns are not merely a template match of the input stream, but rather represent a deeper abstraction of the communication flow. Lastly, the combined integration of all perturbations (viz. ALL) into the learning process yields the best performance across nearly all scenarios.

It is worth noting that, although this study adopts thresholds commonly used in the literature to limit PAY and PSQ inputs—i.e., $N_p = 10$ packets and $N_b = 576$ bytes, respectively—, such limits may vary in real-world scenarios based on operational needs. A robust classifier should not solely depend on the exact presence of specific initial units. Our evaluation of the PHiding perturbation serves as a proxy for assessing this sensitivity. By hiding the first LP packets, we effectively simulate a scenario where the "optimal" early window is either unavailable or shifted. Our results show that while the Base model experiences a significant drop when the initial handshake is missed (e.g., a drop of up to 81% in F1-score on MIRAGE-19), the proposed perturbation-aware strategies (e.g., $\text{Aug}_{\text{PT}} \rightarrow \text{Aug}_{\text{FT}}$) maintain stability. This confirms that our framework decreases the model's sensitivity to the specific thresholds adopted. Additionally, the multimodal approach ensures that if a stricter byte threshold truncates the PAY modality, the PSQ modality effectively compensates for the missing data.

Takeaway:

Our analysis highlights a hierarchy in the effectiveness of domain-specific augmentations: while timing-based perturbations mainly enhance local robustness, structural changes, such as DHidings, enable the model to learn deeper and more invariant representations of traffic flow. This suggests that encouraging the model to reconstruct missing information is a more powerful learning signal than simply exposing it to variations in packet timing.

5.10. Computational complexity and real-world feasibility

In this section, we evaluate the practical deployability of our framework through a comprehensive analysis of the computational overhead during the learning phase (viz. training) and the computational costs, in

Table 12
Training time [$HH:mm$] of $\texttt{Aug}_{\texttt{PT}}$, $\texttt{SimCon}_{\texttt{PT}}$, $\texttt{SupCon}_{\texttt{PT}}$, $\texttt{Aug}_{\texttt{FT}}$, $\texttt{Aug}_{\texttt{PT}} \rightarrow \texttt{Aug}_{\texttt{FT}}$, $\texttt{SimCon}_{\texttt{PT}} \rightarrow \texttt{Aug}_{\texttt{FT}}$, and $\texttt{SupCon}_{\texttt{PT}} \rightarrow \texttt{Aug}_{\texttt{FT}}$ compared with Base on MIRAGE-19, VPN-16, and MIRAGE-24. Results are reported as *mean* ± *std.dev.* over 10 folds.

	VPN-16	MIRAGE-19	MIRAGE-24
Base	00:03 ±00:00	00:21 ±00:01	00:17 ±00:02
$\texttt{Aug}_{\texttt{PT}}$	00:30 ±00:04	06:42 ±00:53	05:40 ±00:50
$\texttt{SimCon}_{\texttt{PT}}$	02:12 ±00:16	28:08 ±01:51	21:17 ±01:12
$\texttt{SupCon}_{\texttt{PT}}$	01:40 ±00:10	24:18 ±00:44	14:21 ±01:18
$\texttt{Aug}_{\texttt{FT}}$	00:14 ±00:02	05:27 ±00:50	04:30 ±00:59
$\texttt{Aug}_{\texttt{PT}} \rightarrow \texttt{Aug}_{\texttt{FT}}$	00:39 ±00:03	08:23 ±01:11	06:59 ±00:59
$\texttt{SimCon}_{\texttt{PT}} \rightarrow \texttt{Aug}_{\texttt{FT}}$	02:24 ±00:16	34:55 ±02:52	25:11 ±01:37
$\texttt{SupCon}_{\texttt{PT}} \rightarrow \texttt{Aug}_{\texttt{FT}}$	01:51 ±00:10	27:35 ±00:56	16:17 ±01:22

terms of memory and inference time, during the operational phase (viz. inference).

Training Overhead Analysis. We analyze the training overhead introduced by the strategies defined in Section 4.2. While these strategies enable the parallel training of individual branches during PT, we report sequential training times as the worst-case training time—i.e., measured by individually training the branches one after another during PT, followed by the subsequent FT phase.

Table 12 reports the training time of all evaluated methods across the three datasets. As expected, variants incorporating transformations in PT ($\texttt{Aug}_{\texttt{PT}}$, $\texttt{SimCon}_{\texttt{PT}}$, $\texttt{SupCon}_{\texttt{PT}}$), in FT ($\texttt{Aug}_{\texttt{FT}}$), or in both ($\texttt{Aug}_{\texttt{PT}} \rightarrow \texttt{Aug}_{\texttt{FT}}$, $\texttt{SimCon}_{\texttt{PT}} \rightarrow \texttt{Aug}_{\texttt{FT}}$, $\texttt{SupCon}_{\texttt{PT}} \rightarrow \texttt{Aug}_{\texttt{FT}}$) exhibit significantly longer training times than Base, which completes in under 30 minutes overall.

Among variants leveraging transformations solely during PT, $\texttt{SimCon}_{\texttt{PT}}$ shows the highest time (e.g., ≈80× on MIRAGE-19), followed by $\texttt{SupCon}_{\texttt{PT}}$. This overhead is partially due to the contrastive loss computation, which requires evaluating the similarity for all $B \cdot (B-1)$ pairs in each training batch—introducing a quadratic computational cost with respect to batch size B. Notably, despite involving the same number of comparisons, $\texttt{SupCon}_{\texttt{PT}}$ exhibits shorter training times compared to $\texttt{SimCon}_{\texttt{PT}}$, likely because its supervised loss benefits from a more structured optimization objective guided by class labels. Conversely, applying Supervised Data Augmentation (Aug) either during PT ($\texttt{Aug}_{\texttt{PT}}$) or FT ($\texttt{Aug}_{\texttt{FT}}$) remains relatively efficient (e.g., up to 19× on MIRAGE-19), and is consistently faster than the former variants.

Lastly, leveraging transformations in both training phases inevitably leads to a further increase in the total training time. However, the results suggest that starting from richer pre-trained branches—via Aug or SupCon—can facilitate optimization during FT, resulting in only a moderate additional overhead when employing Aug during the subsequent FT (e.g., an additional 10× for $\texttt{SupCon}_{\texttt{PT}} \rightarrow \texttt{Aug}_{\texttt{FT}}$ on MIRAGE-19). This is further supported by the fact that, when starting from pre-trained branches with SimCon, the FT based on Aug requires a significantly higher additional training time (e.g., an additional 19× on MIRAGE-19).

Since FT with Aug is consistent across all variants, this overhead does not stem from the FT setup itself. Instead, as earlier results show, it reflects the lower quality of the representation learned during PT with SimCon—leading to poorer classification, worse calibration, and less efficient FT.

Takeaway:

Training models with transformations adds a substantial time overhead compared to the Base *model. While contrastive loss—especially SimCon—results in longer training times, a simpler Aug strategy offers a more efficient alternative. Higher-quality pre-trained models also improve FT efficiency, whereas weaker ones slow down the optimization process.*

Inference Feasibility and Edge Deployment. Unlike the training phase, the operational phase must satisfy strict latency and memory constraints, typically imposed by resource-limited edge devices. In this regard, it is worth noting that the additional overhead induced by the proposed training strategies occurs only during the "offline" learning phase (one-time cost). Since these strategies act solely on the model weights, leaving the architectural backbone invariant, *the inference complexity remains unchanged compared to that of the* Base *model.*

To validate real-world deployability, Table 13 summarizes the computational performance of MIMETIC-ENHANCED across all datasets. Specifically, we report:

1. the number of trainable parameters (**TP**),
2. the number of FLOPs [9] (**FP**),
3. the memory size on disk (**DS**), and
4. the inference time (**iTime**).

Furthermore, we analyze the impact of post-training quantization on these metrics. We compare the reference full-precision model (FP32) on both CPU and GPU against reduced-precision configurations, namely Half-Precision [10] (FP16) on both architectures and Dynamic Quantization [11] (INT8) on CPU.

Overall, our findings highlight that the full-precision model is computationally lightweight (≈10.8 MFLOPs) and has an inference latency on CPU that is comparable to—and in some cases lower than—that of the GPU (i.e., 1.20 vs. 1.25 ms). This indicates that the model is latency-bound rather than compute-bound, making expensive GPU accelerators unnecessary for single-flow processing. In terms of memory optimization, dynamic quantization (INT8) proves to be a suitable strategy for CPU-based edge devices, reducing the memory footprint by ≈ 3.5× compared to the highest precision model (FP32), albeit with a slight latency overhead. As expected, FP16 emulation on standard CPUs results in a significant latency increase (≈ +5 ms) due to the lack of native hardware support for half-precision arithmetic.

Finally, we assessed whether the efficiency gains provided by quantization come at the expense of classification effectiveness and robustness. Focusing on the MIMETIC-ENHANCED architecture trained with $\texttt{Aug}_{\texttt{PT}} \rightarrow \texttt{Aug}_{\texttt{FT}}$, Table 14 compares the full-precision (FP32) model against its quantized (INT8) counterpart on MIRAGE-24. The evaluation covers both nominal conditions and specific critical perturbation scenarios.

As reported, the performance gap between the two models is negligible across all tested conditions. This confirms that the robust feature representations learned via our perturbation-aware strategy are resilient to aggressive numerical compression, validating the viability of INT8 deployment on edge devices without compromising reliability.

Takeaway:

Mimetic-Enhanced *is computationally lightweight (≈1.20 ms/flow on standard CPUs), rendering dedicated GPU accelerators unnecessary. Furthermore, Dynamic Quantization (INT8) effectively reduces memory usage by ≈70% while preserving classification effectiveness, even under severe network perturbations, confirming suitability for resource-constrained edge devices.*

[9] We used the thop Python library https://pypi.org/project/thop/
[10] We leveraged the native implementation provided by PyTorch.
[11] Leveraging the native PyTorch implementation, we selectively quantized the Dense and BiGRU layers, as they account for the vast majority of the model parameters.

Table 13
Computational performance of MIMETIC-ENHANCED on VPN-16, MIRAGE-19, and MIRAGE-24. The results compare the full-precision model (FP32) against reduced-precision configurations on both CPU and GPU—i.e., Half-Precision (FP16) and Dynamic Quantization (INT8). Inference time (iTime) is reported as $mean \pm std.dev.$ over 1,000 runs.

Dev.	Prec.	VPN-16				MIRAGE-19				MIRAGE-24			
		TP[K]	FP[M]	DS[MB]	iTime[ms]	TP[K]	FP[M]	DS[MB]	iTime[ms]	TP[K]	FP[M]	DS[MB]	iTime[ms]
GPU	FP32	877	10.79	5.90	1.24 ± 0.01	880	10.80	3.52	1.25 ± 0.01	878	10.8	3.47	1.25 ± 0.01
	FP16	877	10.79	2.96	1.27 ± 0.01	880	10.80	1.77	1.27 ± 0.01	878	10.8	1.74	1.27 ± 0.01
CPU	FP32	877	10.79	5.90	1.20 ± 0.03	880	10.80	3.52	1.20 ± 0.02	878	10.8	3.47	1.20 ± 0.03
	FP16	877	10.79	2.96	6.26 ± 0.04	880	10.80	1.77	6.25 ± 0.07	878	10.8	1.74	6.25 ± 0.05
	INT8	16.9	8.51	3.44	1.25 ± 0.03	16.9	8.51	1.01	1.24 ± 0.02	16.9	8.51	0.99	1.27 ± 0.03

TP: Trainable Parameters, **FP**: FLOPs, **DS**: Memory Size (on Disk), **iTime**: inference time.

Table 14
Performance (`F1-score`) of MIMETIC-ENHANCED ($\texttt{Aug}_{\texttt{PT}} \rightarrow \texttt{Aug}_{\texttt{FT}}$) under post-training quantization on MIRAGE-24. The full-precision model (FP32) is compared against the quantized model (INT8) across clean traffic and critical perturbation scenarios. Results are reported as $mean \pm std.dev.$ over 10 folds.

Prec.	Clean	⊕ Test Perturbation ⊕		
		`PHiding` (LP=5)	$\texttt{DHiding}_{\texttt{loss}}$ ($C \leftarrow S, p = 30\%$)	`RTObef` ($r = 30\%$)
FP32	96.57 ± 0.49	78.49 ± 1.71	88.87 ± 1.32	94.76 ± 0.57
INT8	96.58 ± 0.48	78.44 ± 1.72	88.86 ± 1.40	94.77 ± 0.53

Clean: unperturbed traffic samples. *LP*: number of lost initial packets in the biflow. *p*: packet-loss rate (%). *r*: retransmission rate (%) with $T_{out} = 1$ s.
$C \leftarrow S$: only Server-to-Client direction of the biflow is visible at the VP.

6. Conclusion

Ensuring robustness in TC is essential for maintaining accurate and resilient network management in the face of evolving traffic patterns and the inherent uncertainty of real-world deployments.

In this work, we analyzed the impact of temporal and structural perturbations on TC performance. Through an extensive evaluation on three public datasets—i.e., VPN-16, MIRAGE-19, and MIRAGE-24—we showed that MIMETIC-ENHANCED, a state-of-the-art multimodal architecture, offers improved classification performance on both clean (i.e., *effectiveness*) and perturbed (i.e., *robustness*) traffic when compared to single-modal approaches. However, while resilient to time-based perturbations, MIMETIC-ENHANCED *is not robust to structural perturbations* involving packet reordering, delayed monitoring, and asymmetric paths, especially when its view on traffic is limited (e.g., up to −81% `F1-score` when just the first two packets of the biflow are missed on MIRAGE-19). This suggests a *strong dependence on the visibility and completeness of the biflow* observed at the VP.

To mitigate these vulnerabilities, we integrated *perturbation-aware training* into the MIMETIC-ENHANCED pipeline by leveraging Supervised Data Augmentation (`Aug`) and Contrastive Learning (`CL`) (`SimCon` and `SupCon`) during its Pre-Training (PT) and Fine-Tuning (FT) phases. Our analysis revealed that `Aug` and `SupCon` consistently enhance TC performance and robustness when applied across both phases (i.e., $\texttt{Aug}_{\texttt{PT}} \rightarrow \texttt{Aug}_{\texttt{FT}}$ and $\texttt{SupCon}_{\texttt{PT}} \rightarrow \texttt{Aug}_{\texttt{FT}}$), while `SimCon`-based strategies fail to yield similar gains.

Beyond gains in effectiveness and robustness, *our analysis highlights further benefits*. In particular, using `Aug` during PT (or FT) *enhances OOD detection*—e.g., yielding an improvement of up to +6% `AUROC` when training on MIRAGE-19 and testing on MIRAGE-24. Similarly, employing `SupCon` in PT improves model calibration, with reductions of up to −0.2 ECE on MIRAGE-19. However, these benefits come at the cost of a slight degradation in calibration performance when applying `Aug` during FT, as it tends to increase model overconfidence. This effect is well-mitigated by $\texttt{SupCon}_{\texttt{PT}} \rightarrow \texttt{Aug}_{\texttt{FT}}$ and $\texttt{Aug}_{\texttt{PT}} \rightarrow \texttt{Aug}_{\texttt{FT}}$ strategies, which offer the best trade-off between classification performance and reliability.

We further investigated the impact of these strategies under *constrained training data conditions*. Our finding reveals that, while data scarcity inevitably reduces effectiveness, the adoption of `Aug` and `SupCon` can partially mitigate its effects in almost all cases.

Although perturbation-aware strategies significantly increase training time, `Aug`-based methods offer the best trade-off between effectiveness, robustness improvement, and computational overhead. In particular, while `CL`-based approaches (e.g., $\texttt{SupCon}_{\texttt{PT}}$) introduce a training overhead of over 80×, the simpler `Aug` strategies (e.g., $\texttt{Aug}_{\texttt{PT}}$) achieve comparable or even better performance with less than 30× additional training time. Crucially, this computational burden represents a *one-time offline cost*. It does not affect the operational phase—as the model architecture and inference latency remain unchanged—but significantly enhances generalizability and robustness against real-world network fluctuations.

Finally, we demonstrate that the proposed *learning framework* is effectively *model-agnostic*. By explicitly validating its *broad generalizability* on representative classifiers—spanning both single-modal and multimodal paradigms—we prove that it enhances robustness regardless of the specific model design or training scheme.

In summary, our findings support the adoption of *multimodal architectures combined with perturbation-aware training strategies* as a practical, general-purpose solution to build robust, calibrated, and generalizable traffic classifiers, prioritizing deployment efficiency over architectural complexity.

Future directions will account for

1. adopting eXplainable Artificial Intelligence (XAI) for analyzing and improving model robustness;
2. extending robustness analysis to few-shot class-incremental learning scenarios;
3. evaluating model robustness in real-perturbed scenarios; and
4. comparing our traditional perturbation-aware training with Generative AI-based data augmentation

.

Data availability

Data will be made available on request.

CRediT authorship contribution statement

Idio Guarino: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Data curation, Conceptualization; **Giampaolo Bovenzi:** Writing – review & editing, Writing – original draft, Software, Methodology, Data curation, Conceptualization; **Alfredo Nascita:** Writing – review & editing, Writing – original draft, Visualization, Validation, Investigation; **Domenico Ciuonzo:** Writing – review & editing, Writing – original draft, Validation, Supervision, Methodology, Investigation; **Damiano Carra:** Writing – review & editing, Supervision, Resources, Conceptualization; **Antonio Pescapè:** Writing – review & editing.

Algorithm 1 DHiding/DHiding$_{\text{loss}}$.

Input: Biflow BF of N packets; number of output packets N_p; number of output bytes N_b; maximum drop probability $p_{\max}$; magnitude a
Output: PAY: N_b payload bytes; PSQ: headers of up to N_p packets
Note: ExtrPAY and ExtrPSQ return the PAY and PSQ modalities from the selected packets, applying padding if needed;

Identify unique directions D in BF
if $|D| = 1$ **then**
  return (ExtrPAY(BF, N_b), ExtrPSQ(BF, N_p)) ▷ **Skip**: *one-way flow*
end if
Sample direction: $d_{tx} \sim \mathcal{U}\{0, 1\}$ ▷ Direction to transmit
Sample $\hat{p} \sim \mathcal{U}(0, p_{\max})$; $p \leftarrow a \cdot \hat{p}$
$I \leftarrow \{i \in [0, N) \mid \text{DIR}_i = d_{tx}\}$
For each $i \in I$, sample $z_i \sim \mathcal{U}(0, 1)$
$I' \leftarrow \{i \in I \mid z_i \geq p\}$ ▷ Keep only non-dropped packets
if $|I'| = 0$ **then**
  return (ExtrPAY(BF, N_b), ExtrPSQ(BF, N_p)) ▷ **Skip**: *no drop all*
end if
PAY $\leftarrow$ ExtrPAY(BF, N_b, I')
PSQ $\leftarrow$ ExtrPSQ(BF, N_p, I')
return (PAY, PSQ)

Algorithm 2 Packets Hiding (PHiding).

Input: Biflow BF of N packets; number of output packets N_p; number of output bytes N_b; maximum drop ratio $r_{\max}$; magnitude parameter a
Output: PAY: N_b payload bytes; PSQ: headers of up to N_p packets
Note: ExtrPAY and ExtrPSQ return the PAY and PSQ modalities from the selected packets, applying padding if needed; ExtrPSQ also recomputes IATs.

$T \leftarrow \min(N, N_p)$
if $T = 1$ **then** ▷ Single packet
  return (ExtrPAY(BF, N_b), ExtrPSQ(BF, N_p)) ▷ **Skip**: *single packet*
end if
Sample $r \sim \mathcal{U}(0, r_{\max})$; $r \leftarrow a \cdot r$
Sample $f \sim \mathcal{U}\{1, \lfloor r \cdot T \rfloor\}$ ▷ Index of first visible packet
$I \leftarrow \{f, f + 1, \ldots, \min(f + N_p - 1, N - 1)\}$ ▷ Up to N_p packets, within bounds
PAY $\leftarrow$ ExtrPAY(BF, N_b, I)
PSQ $\leftarrow$ ExtrPSQ(BF, N_p, I)
return (PAY, PSQ)

Algorithm 3 RTOaft/ RTObef.

Input: Biflow BF of N packets; number of output packets N_p; number of output bytes N_b; max retransmission probability $\lambda_{\max}$; magnitude parameter a; timeout T_{out}; min and max retransmissions $L_{\min}$, $L_{\max}$; RTOaft flag drop$_{\text{a}}$
Output: PAY: N_b payload bytes; PSQ: headers of up to N_p packets
Note: ExtrPAY and ExtrPSQ return the PAY and PSQ modalities from the selected packets, applying padding if needed; ExtrPSQ also recomputes IATs.

Sample $\lambda \sim a \cdot \mathcal{U}(0, \lambda_{\max})$ ▷ Retransmission rate
$T \leftarrow \min(N, N_p)$ ▷ Maximum packets to consider
Sample $r_i \sim \mathcal{U}(0, 1)$ for each $i \in [0, T - 1]$
$P \leftarrow []$ ▷ List of selected packets and retransmissions
for $i = 0$ to $T - 1$ **do**
  $t_e^i \leftarrow \sum_{k=0}^{i-1} \text{IAT}_k$ ▷ Elapsed time from start
  if $r_i < \lambda$ **then** ▷ Retransmission condition
    if drop$_{\text{a}}$ **then** ▷ Add original packet (RTOaft)
      Add $\langle i, t_e^i \rangle$ to P
    end if
    Sample $L \sim \mathcal{U}(L_{\min}, L_{\max})$ ▷ # retransmissions
    for $r = 1$ to L **do**
      Add $\langle i, t_e^i + T_{out} \cdot r \rangle$ to P
    end for
  else ▷ No retransmission
    Add $\langle i, t_e^i \rangle$ to P
  end if
end for
$P \leftarrow \text{sort}(P)$ ▷ Sort packets by elapsed time
PAY $\leftarrow$ ExtrPAY(BF, N_b, P)
PSQ $\leftarrow$ ExtrPSQ(BF, N_p, P)
return (PAY, PSQ)

Declaration of interests

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work is partially supported by the Italian PNRR MUR "Centro Nazionale HPC, Big Data e Quantum Computing, Spoke9 - Digital Society & Smart Cities"; in part by the "xInternet" Project—funded by MUR—within the PRIN 2022 program (D.D.104-02/02/2022); in part by the European Union (EU) under the Italian National Recovery and Resilience Plan (NRRP) of NextGenerationEU, partnership on "Telecommunications of the Future" (PE00000001 - program "RESTART"); and in part by the EU and MUR under the NRRP through the project "CSI-Future (Joint Communication and Sensing: CSI-Based Sensing for Future Wireless Networks)", PRIN 2022 PNRR (CUP B53D23023920001)." This manuscript reflects only the authors' views and opinions and the Ministry cannot be considered responsible for them.

Appendix A. Pseudo-code

The present Appendix provides the pseudo-code of the transformations described in Section 3.2, including their input parameters, internal logic, and returned modalities.

Direction Hiding with Packet Loss (DHiding$_{\text{loss}}$) (Algorithm 1) simulates asymmetric paths where only one directional flow (i.e., upstream or downstream) crosses the VP, and packets may be dropped with probability $p \sim a \cdot \mathcal{U}(0, p_{\max})$. Setting $p_{\max} = 0$ disables packet dropping and corresponds to the basic **Direction Hiding (DHiding)**.

Packets Hiding (PHiding) (Algorithm 2) simulates the scenario in which the VP starts observing the biflow with delay. The first visible packet is randomly selected from the initial portion of the flow, with the starting index $f \sim \mathcal{U}(1, \lfloor r \cdot T \rfloor)$, where $r \sim a \cdot \mathcal{U}(0, r_{\max})$.

Retransmission Timeout (RTO) (Algorithm 3) simulates packet retransmission within a biflow, where the packets to be retransmitted and the number of retransmissions are chosen randomly. Packets may be retransmitted with probability $\lambda \sim a \cdot \mathcal{U}(0, \lambda_{\max})$. Each selected packet may be retransmitted $L \sim \mathcal{U}(L_{\min}, L_{\max})$ times. Each selected packet is retransmitted after a preset timeout T_{out}. Two scenarios are considered: (*i*) RTOaft, where the original packet and all retransmissions are observed (loss after VP), and (*ii*) RTObef, where only retransmissions are observed (original packet lost before VP).

Vantage Point Centralization (VPC)(Algorithm 4) simulates a scenario where both forward and backward paths cross the VP, but the VP is not

Algorithm 4 Vantage Point Centralization (VPC)/Vantage Point Centralization with Packet Loss ($\texttt{VPC}_{\texttt{loss}}$).

Input: Biflow BF of N packets; number of output packets N_p; number of output bytes N_b; $\alpha_{\max}$; maximum drop probability $p_{\max}$; magnitude parameter a
Output: PAY: N_b payload bytes; PSQ: headers of up to N_p packets
Note: ExtrPAY and ExtrPSQ return the PAY and PSQ modalities from the selected packets, applying padding if needed;

$\hat{RTT} \leftarrow$ EstimateRTT(BF) ▷ RTT estimation
Sample $\hat{p} \sim \mathcal{U}(0, p_{\max})$; $p \leftarrow a \cdot \hat{p}$ ▷ Packet drop rate
Sample $\alpha \sim \mathcal{U}(0, \alpha_{\max})$; ▷ RTT lag
$T \leftarrow \min(N, N_p)$
if $T = 1$ **then**
  return (ExtrPAY(BF, N_b), ExtrPSQ(BF, N_p)) ▷ **Skip**: *single packet*
end if
Sample $p_i \sim \mathcal{U}(0, 1)$ for each $i \in [0, T-1]$
$I \leftarrow \{i \mid i \in [0, T-1] \text{ and } p_i \geq \hat{p}\}$ ▷ Indices of visible packets
if $|I| = 0$ **then**
  return (ExtrPAY(BF, N_b), ExtrPSQ(BF, N_p)) ▷ **Skip**: *no visible packets*
end if
for $i \in I \backslash \{\max(I)\}$ **do** ▷ Iterate across the selected packets of BF
  if $DIR_i \neq DIR_{i+1}$ **then**
    $IAT_i \leftarrow IAT_i + \hat{RTT} \times \alpha \times (1 - 2 \cdot 1(DIR_i = dw))$ ▷ Adjust IAT of ith packet of BF
  end if
end for
PAY $\leftarrow$ ExtrPAY(BF, N_b, I)
PSQ $\leftarrow$ ExtrPSQ(BF, N_p, I)
return (PAY, PSQ)

Table A.1
IATs and jitter contributions observed at the VP.

Interval	Direction(s)	IAT_i (no jitter)	Jitter term
$t_1 - t_0$	C→S → C→S (cv)	$\Delta_1 + \delta_1^{cv} - \delta_0^{cv}$	$\lambda_1^{cv} - \lambda_0^{cv}$
$t_2 - t_1$	C→S → S→C (vs, sv)	$\Delta_2 + \delta_2^{sv} + \delta_2^{vs}$	$\lambda_2^{sv} + \lambda_2^{vs}$
$t_3 - t_2$	S→C → S→C (sv)	$\Delta_3 + \delta_3^{sv} - \delta_2^{sv}$	$\lambda_3^{sv} - \lambda_2^{sv}$
$t_4 - t_3$	S→C → C→S (vc, cv)	$\Delta_4 + \delta_4^{cv} + \delta_4^{vc}$	$\lambda_4^{cv} + \lambda_4^{vc}$

located near the endpoints. Under the assumption that traffic is captured close to the client, the transformation adjusts the IATs of packets sent in opposite directions to emulate the timing distortions observed by a VP placed farther away from the client (viz. *centralized*).

The IAT of each packet is adjusted according to the estimated RTT and a lag factor $\alpha \in [0, 1]$, which is sampled from $\mathcal{U}(0, \alpha_{\max})$.

The RTT is estimated as the time between the SYN and SYN + ACK packets for TCP biflows or the delay between the pair of packets with minimum cumulative size in opposite directions for UDP biflows. The Vantage Point Centralization with Packet Loss ($\texttt{VPC}_{\texttt{loss}}$) variant additionally introduces packet loss with probability $p \sim a \cdot \mathcal{U}(0, p_{\max})$. Setting $p_{\max} = 0$ disables packet dropping and corresponds to the basic VPC.

Directional Time Jittering (TJitter) (Algorithm 5) introduces delay jitter by modifying the IAT of packet data based on traffic direction (i.e., 0 = Client-to-VP and 1 = Server-to-VP). For each specified direction $d \in D$, jitter values are sampled from the positive half of a normal distribution—$\mathcal{N}^+(u, \sigma_d)$, where σ_d is the standard deviation for that direction. For the ith packet of a biflow, jitter is applied according to its direction, DIR_i, and influences its IAT, IAT_i, depending on whether the previous packet had the same direction or a different one.

To obtain this transformation, we provide a *packet timing model at a vantage point*. In particular, we model a sequence of packets exchanged

Algorithm 5 Directional Time Jittering (TJitter).

Input: Biflow BF of N packets; number of output packets N_p; number of output bytes N_b; list of traffic directions to jitter D; mean jitter u; per-direction standard deviation $\Delta = \{\sigma_d \mid d \in D\}$.
Output: PAY: N_b payload bytes; PSQ: headers of up to N_p packets
Output: PAY: N_b payload bytes; PSQ: headers of up to N_p packets
Note: ExtrPAY and ExtrPSQ return the PAY and PSQ modalities from the selected packets, applying padding if needed;

for d in D **do**
  for $i \in [1, N_p - 1]$ **do** ▷ Iterate from the 2nd to the N_p^{th} packet of BF
    if $\text{DIR}_i = d$ **then**
      Sample $\delta_a^i, \delta_b^i \sim \mathcal{N}^+(u, \sigma_d)$ ▷ Per-direction jitter values
      if $\text{DIR}_{i-1} = \text{DIR}_i$ **then** ▷ Adjust IAT of i^{th} packet of BF
        $\text{IAT}_i \leftarrow \min(\text{IAT}_i + |\delta_a^i - \delta_b^i|, \text{IAT}_{\max})$
      else ▷ Adjust IAT of i^{th} packet of BF
        $\text{IAT}_i \leftarrow \min(\text{IAT}_i + |\delta_a^i + \delta_b^i|, \text{IAT}_{\max})$
      end if
    end if
  end for
end for
PAY $\leftarrow$ ExtrPAY(BF, N_b)
PSQ $\leftarrow$ ExtrPSQ(BF, N_p)
return (PAY, PSQ)

between a Client (C) and a Server (S), as observed from a Vantage Point (V) located between them. Each packet is associated with:

- Δ_i: host-side thinking time before sending the ith packet,
- δ_i^{xy}: base propagation delay from x to y, where $xy \in \{cv, vs, sv, vc\}$,
- λ_i^{xy}: network-induced jitter on path xy for the ith packet,
- t_i: arrival time at the VP of the ith packet,
- $\text{IAT}_i = t_i - t_{i-1}$: inter-arrival time between packets $i-1$ and i at VP.

The following equations define the arrival times of five example packets, sent in this order: C→S, C→S, S→C, S→C, and C→S. This sequence encompasses all relevant client/server interleaving, assuming a request/response communication pattern, as appropriate for a model restricted to application-level packets.

Accordingly, the arrival time t_i of each packet is composed as follows.

$$t_0 = \delta_0^{cv} + \lambda_0^{cv}$$
$$t_1 = \Delta_1 + \delta_1^{cv} + \lambda_1^{cv}$$
$$t_2 = t_1 + \delta_2^{vs} + \lambda_2^{vs} + \Delta_2 + \delta_2^{sv} + \lambda_2^{sv}$$
$$t_3 = t_1 + \delta_2^{vs} + \lambda_2^{vs} + \Delta_2 + \Delta_3 + \delta_3^{sv} + \lambda_3^{sv}$$
$$t_4 = t_3 + \delta_4^{vc} + \lambda_4^{vc} + \delta_4^{cv} + \lambda_4^{cv} + \Delta_4$$

Following these equations, the jittered IATs of packets results in Table A.1. Let DIR_i denote the direction of the ith packet in the biflow. The perturbed IAT, $\widehat{\text{IAT}}_i$, is computed as follows:

- **If** the preceding packet had the same direction as the current one ($\text{DIR}_i = \text{DIR}_{i-1}$):
$$\widehat{\text{IAT}}_i = \Delta_i + \delta_i^{\text{DIR}_i} - \delta_{i-1}^{\text{DIR}_i} + \lambda_i^{\text{DIR}_i} - \lambda_{i-1}^{\text{DIR}_i} = \text{IAT}_i + \lambda_i^{\text{DIR}_i} - \lambda_{i-1}^{\text{DIR}_i}$$
- **If** the preceding packet had the opposite direction as the current one ($\text{DIR}_i \neq \text{DIR}_{i-1}$, direction switch):
$$\widehat{\text{IAT}}_i = \Delta_i + \delta_i^{\text{DIR}_i} + \delta_{i-1}^{\text{DIR}_{i-1}} + \lambda_i^{\text{DIR}_i} + \lambda_{i-1}^{\text{DIR}_{i-1}} = \text{IAT}_i + \lambda_i^{\text{DIR}_i} + \lambda_{i-1}^{\text{DIR}_{i-1}}$$

References

[1] T.T.T. Nguyen, G.J. Armitage, A survey of techniques for internet traffic classification using machine learning, IEEE Commun. Surv. Tutorials 10 (1–4) (2008) 56–76.

[2] F. Pacheco, E. Exposito, M. Gineste, C. Baudoin, J. Aguilar, Towards the deployment of machine learning solutions in network traffic classification: a systematic survey, IEEE Commun. Surv. Tutorials (2018) 1–1.

[3] G. Aceto, D. Ciuonzo, A. Montieri, V. Persico, A. Pescapé, AI-powered internet traffic classification: past, present, and future, IEEE Commun. Mag. 62 (9) (2024) 168–175.

[4] L. Bernaille, R. Teixeira, I. Akodkenou, A. Soule, K. Salamatian, Traffic classification on the fly, ACM SIGCOMM Comput. Commun. Rev. 36 (2) (2006) 23–26.

[5] M. Crotti, M. Dusi, F. Gringoli, L. Salgarelli, Traffic classification through simple statistical fingerprinting, ACM SIGCOMM Comput. Commun. Rev. 37 (1) (2007) 5–16.

[6] A. Dainotti, F. Gargiulo, L.I. Kuncheva, A. Pescapè, C. Sansone, Identification of traffic flows hiding behind TCP port 80, in: IEEE Int. Conf. Commun. (ICC), 2010, pp. 1–6.

[7] P. M. Santiago del Rio, D. Rossi, F. Gringoli, L. Nava, L. Salgarelli, J. Aracil, Wire-Speed statistical classification of network traffic on commodity hardware, in: Proceedings of the ACM Internet Measurement Conference (IMC), 2012, pp. 65–72.

[8] G. Aceto, D. Ciuonzo, A. Montieri, V. Persico, A. Pescapé, MIRAGE: mobile-app traffic capture and ground-truth creation, in: IEEE 4th International Conference on Computing, Communications and Security (ICCCS), 2019, pp. 1–8.

[9] I. Guarino, G. Aceto, D. Ciuonzo, A. Montieri, V. Persico, A. Pescapè, Contextual counters and multimodal deep learning for activity-level traffic classification of mobile communication apps during COVID-19 pandemic, Elsevier Comput. Netw. 219 (2022) 109452.

[10] Z. Wang, The applications of deep learning on traffic identification, BlackHat USA (2015).

[11] W. Wang, M. Zhu, J. Wang, X. Zeng, Z. Yang, End-to-end encrypted traffic classification with one-dimensional convolution neural networks, in: IEEE International Conference on Intelligence and Security Informatics (ISI), 2017, pp. 43–48.

[12] G. Aceto, D. Ciuonzo, A. Montieri, A. Pescapé, Mobile encrypted traffic classification using deep learning: experimental evaluation, lessons learned, and challenges, IEEE Trans. Netw. Serv. Manage. 16 (2) (2019) 445–458.

[13] M. Lopez-Martin, B. Carro, A. Sanchez-Esguevillas, J. Lloret, Network traffic classifier with convolutional and recurrent neural networks for internet of things, IEEE Access 5 (2017) 18042–18050.

[14] M. Lotfollahi, M.J. Siavoshani, R.S.H. Zade, M. Saberian, Deep packet: a novel approach for encrypted traffic classification using deep learning, Springer Soft Comput. 24 (3) (2020).

[15] G. Aceto, D. Ciuonzo, A. Montieri, A. Pescapè, MIMETIC: Mobile encrypted traffic classification using multimodal deep learning, Elsevier Comput. Netw. 165 (2019) 106944.

[16] A. Nascita, A. Montieri, G. Aceto, D. Ciuonzo, V. Persico, A. Pescapé, XAI meets mobile traffic classification: understanding and improving multimodal deep learning architectures, IEEE Trans. Netw. Serv. Manage. 18 (4) (2021) 4225–4246.

[17] Q. Yuan, G. Gou, Y. Zhu, Y. Wang, MMCo: Using multimodal deep learning to detect malicious traffic with noisy labels, Front. Comput. Sci. 18 (1) (2023) 181809.

[18] X. Wang, S. Chen, J. Su, Automatic mobile app identification from encrypted traffic with hybrid neural networks, IEEE Access 8 (2020) 182065–182077.

[19] G. Draper-Gil, A.H. Lashkari, M.S.I. Mamun, A.A. Ghorbani, Characterization of encrypted and VPN traffic using time-related features, in: Proceedings of the Springer International Conference on Information Systems Security and Privacy (ICISSP), 2016, pp. 407–414.

[20] I. Guarino, D. Ciuonzo, A. Montieri, A. Pescapè, Mirage-app×act-2024: a novel dataset for mobile app and activity traffic analysis, in: IEEE 20th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), 2024, pp. 663–666.

[21] Z. Liu, R. Wang, N. Japkowicz, Y. Cai, D. Tang, X. Cai, Mobile app traffic flow feature extraction and selection for improving classification robustness, Elsevier J. Netw. Comput. Appl. 125 (2019) 190–208.

[22] A.M. Sadeghzadeh, S. Shiravi, R. Jalili, Adversarial network traffic: towards evaluating the robustness of deep-learning-based network traffic classification, IEEE Trans. Netw. Serv. Manage. 18 (2) (2021) 1962–1976.

[23] Y. Sharon, D. Berend, Y. Liu, A. Shabtai, Y. Elovici, TANTRA: timing-based adversarial network traffic reshaping attack, IEEE Trans. Inf. Forensics Secur. 17 (2022) 3225–3237.

[24] C. Zhang, X. Costa-Pérez, P. Patras, Adversarial attacks against deep learning-based network intrusion detection systems and defense mechanisms, IEEE/ACM Trans. Networking 30 (3) (2022) 1294–1311.

[25] I. Guarino, C. Wang, A. Finamore, A. Pescapè, D. Rossi, Many or few samples?: comparing transfer, contrastive and meta-learning in encrypted traffic classification, in: IEEE 7th Network Traffic Measurement and Analysis Conference (TMA), 2023, pp. 1–10.

[26] R. Xie, J. Cao, E. Dong, M. Xu, K. Sun, Q. Li, L. Shen, M. Zhang, Rosetta: enabling robust TLS encrypted traffic classification in diverse network environments with TCP-aware traffic augmentation, in: Proceedings of the 32nd USENIX Security Symposium, 2023, pp. 625–642.

[27] W. Li, X.-Y. Zhang, H. Bao, B. Yang, Z. Li, H. Shi, Q. Wang, Prism: real-time privacy protection against temporal network traffic analyzers, IEEE Trans. Inf. Forensics Secur. 18 (2023) 2524–2537.

[28] R. Xie, J. Cao, Y. Zhu, Y. Zhang, Y. He, H. Peng, Y. Wang, M. Xu, K. Sun, E. Dong, Q. Li, M. Zhang, J. Li, Cactus: obfuscating bidirectional encrypted TCP traffic at client side, IEEE Trans. Inf. Forensics Secur. 19 (2024) 7659–7673.

[29] X. Deng, Y. Wang, Z. Xue, AN-Net: an anti-noise network for anonymous traffic classification, in: Proceedings of the ACM Web Conference (WWW), 2024, p. 4417-4428.

[30] C. Wang, A. Finamore, P. Michiardi, M. Gallo, D. Rossi, Data augmentation for traffic classification, in: Springer Nature Passive and Active Measurement (PAM), 2024, pp. 159–186.

[31] R. Ding, L. Sun, W. Zang, L. Dai, Z. Ding, B. Xu, Towards universal and transferable adversarial attacks against network traffic classification, Elsevier Comput. Netw. (2024) 110790.

[32] X. Yang, S. Ruan, J.L. Yinliang, Y. Bo Sun, TrafCL: robust encrypted malicious traffic detection via contrastive learning, in: Proceedings of the 33rd ACM International Conference on Information & Knowledge Management (CIKM), 2024, pp. 2910–2919.

[33] C. Hajaj, P. Aharon, R. Dubin, A. Dvir, The art of time-bending: data augmentation and early prediction for efficient traffic classification, Elsevier Expert Syst. Appl. 252 (2024) 124166.

[34] Y. Jiang, X. Wang, Y. Lai, Y. Wang, A packet sequence permutation-aware approach to robust network traffic classification, IEEE Netw.Lett. 6 (3) (2024) 203–207.

[35] E. Horowicz, T. Shapira, Y. Shavitt, Self-supervised traffic classification: flow embedding and few-shot solutions, IEEE Trans. Netw. Serv. Manage. 21 (3) (2024) 3054–3067.

[36] M. Jin, M. Apostolaki, Robustifying ML-powered network classifiers with PANTS, in: Proceedings of the 34th USENIX Security Symposium, 2025, pp. 625–642.

[37] H. Lu, J. Liu, J. Peng, J. Lu, Adversarial attacks based on time-series features for traffic detection, Elsevier Compu. Secur. 148 (2025) 104175.

[38] X. Wang, Q. Yuan, W. Yu, Q. Meng, S. Lu, W. He, C. Gu, Y. Wang, Entropy-regulated cross-modal generative fusion for multimodal network intrusion detection, Elsevier Inf. Fusion 126 (2026) 103581.

[39] X. Wang, J. Li, X. Kuang, Y.-a. Tan, J. Li, The security of machine learning in an adversarial setting: a survey, J. Parallel Distrib. Comput. 130 (2019) 12–23.

[40] A. Dainotti, A. Pescapè, K.C. Claffy, Issues and future directions in traffic classification, IEEE Netw. 26 (1) (2012) 35–40.

[41] S. Huang, F. Cuadrado, S. Uhlig, Middleboxes in the internet: a HTTP perspective, in: IEEE Network Traffic Measurement and Analysis Conference (TMA), 2017, pp. 1–9.

[42] G. Wan, S. Liu, F. Bronzino, N. Feamster, Z. Durumeric, CATO: end-to-end optimization of ML-based traffic analysis pipelines, in: Proceedings of the 22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25), 2025, pp. 1523–1540.

[43] A. D'Alconzo, I. Drago, A. Morichetta, M. Mellia, P. Casas, A survey on big data for network traffic monitoring and analysis, IEEE Trans. Netw. Serv. Manage. 16 (3) (2019) 800–813.

[44] F. Cerasuolo, A. Nascita, G. Bovenzi, G. Aceto, D. Ciuonzo, A. Pescapè, D. Rossi, MEMENTO: a novel approach for class incremental learning of encrypted traffic, Elsevier Comput. Netw. 245 (2024) 110374.

[45] F. Cerasuolo, I. Guarino, V. Spadari, G. Aceto, A. Pescapé, XAI For interpretable multimodal architectures with contextual input in mobile network traffic classification, in: IEEE/IFIP Networking Conference (IFIP Networking), 2024, pp. 757–762.

[46] X. Chen, H. Kim, J.M. Aman, W. Chang, M. Lee, J. Rexford, Measuring TCP round-trip time in the data plane, in: Proceedings of the ACM Workshop on Secure Programmable Network Infrastructure (SPIN '20), Association for Computing Machinery, 2020, pp. 35–41.

[47] C. Shorten, T.M. Khoshgoftaar, A survey on image data augmentation for deep learning, Springer J. Big Data 6 (1) (2019) 1–48.

[48] T. Chen, S. Kornblith, M. Norouzi, G. Hinton, A simple framework for contrastive learning of visual representations, in: Proceedings of the 37th ACM International Conference on Machine Learning (CML'20), 2020, pp. 1597–1607.

[49] X. Liu, F. Zhang, Z. Hou, L. Mian, Z. Wang, J. Zhang, J. Tang, Self-supervised learning: generative or contrastive, IEEE Trans. Knowl. Data Eng. 35 (1) (2023) 857–876.

[50] P. Khosla, P. Teterwak, C. Wang, A. Sarna, Y. Tian, P. Isola, A. Maschinot, C. Liu, D. Krishnan, Supervised contrastive learning, in: Proceedings of the 34th International Conference on Neural Information Processing Systems (NIPS '20), 33, Curran Associates Inc., 2020, pp. 18661–18673.

[51] D.L. Davies, D.W. Bouldin, A cluster separation measure, IEEE Trans. Pattern Anal. Mach. Intell. PAMI-1 (2) (1979) 224–227.

[52] P.J. Rousseeuw, Silhouettes: a graphical aid to the interpretation and validation of cluster analysis, Elsevier J. Comput. Appl. Math. 20 (1987) 53–65.

[53] M. Kull, M. Perello-Nieto, M. Kängsepp, T.S. Filho, H. Song, P. Flach, Beyond temperature scaling: obtaining well-calibrated multiclass probabilities with dirichlet calibration, in: Proc. of the 33rd International Conference on Neural Information Processing Systems (NeurIPS), 2019, pp. 12316–12326.

[54] M.P. Naeini, G.F. Cooper, M. Hauskrecht, Obtaining well calibrated probabilities using bayesian binning, in: Proceedings of the ACM 29th AAAI Conference on Artificial Intelligence (AAAI'15), 2015, p. 2901-2907.

[55] L. Yang, A. Finamore, F. Jun, D. Rossi, Deep learning and zero-day traffic classification: lessons learned from a commercial-grade dataset, IEEE Trans. Netw. Serv. Manage. 18 (4) (2021) 4103–4118.

[56] H. Wu, Y. Liu, S. Ni, G. Cheng, X. Hu, Lossdetection: real-time packet loss monitoring system for sampled traffic data, IEEE Trans. Netw. Serv. Manage. 20 (1) (2023) 30–45.

[57] M.A. Ahmed, B. Bensaou, Enhancing TCP via hysteresis switching: theoretical analysis and empirical evaluation, IEEE/ACM Trans. Networking 31 (6) (2023) 2614–2623.