The Case for Native Multi-Node In-Network Machine Learning

Lorenzo Bracciale
lorenzo.bracciale@uniroma2.it
Univ. of Rome Tor Vergata / CNIT
Rome, Italy

Tushar Swamy
tswamy@stanford.edu
Stanford University
USA

Muhammad Shahbaz
mshahbaz@purdue.edu
Purdue University
USA

Pierpaolo Loreti
pierpaolo.loreti@uniroma2.it
Univ. of Rome Tor Vergata / CNIT
Italy

Stefano Salsano
stefano.salsano@uniroma2.it
Univ. of Rome Tor Vergata / CNIT
Italy

Hesham Elbakoury
helbakoury@gmail.com
Consultant
USA

ABSTRACT

It is now possible to run per-packet Machine Learning (ML) inference tasks in the data plane at line-rate with dedicated hardware in programmable network switches. We refer to this approach as *per-packet* ML. Existing work in this area focuses on a *single node* setup, where the incoming packets are processed by the switch pipeline to extract features at different levels of granularity: packet-level, flow-level, cross-flow level, while also considering device-level features. The extracted features are then processed by an ML inference fabric inside the same switch.

In this position paper, we propose to extend and enhance this model from a single node to a collection of nodes (including switches and servers). In fact, there are several scenarios where it is impossible for a single node to perform both feature processing (e.g., due to lack of or limited access to data) and the ML inference operations. In a multi-node setup, a node can extract ML features and encode them in packets as metadata, which are then processed by another node (e.g., switch) to execute native inference tasks. We make a case for a standard model of extracting, encoding, and forwarding features between nodes to carryout distributed, native ML inference inside networks; discuss the applicability and versatility of the proposed model; and illustrate the various open research issues and design implications.

1 INTRODUCTION

Recent efforts (e.g., Taurus [17], IIsy [20], and pForest [4]) show that, with programmable hardware blocks (like MapReduce and Match-Action Tables, MATs) inside network switches, it is now possible to run per-packet Machine Learning Inference (MLI) tasks inside the data plane, i.e. to run ML algorithms on each packet individually and at line rate. We refer to this approach as *per-packet* ML. For example, a Taurus-enabled switch (Figure 1) extends the Protocol Independent Switch Architecture (PISA) [14] by adding an ML inference engine (based on MapReduce). The new block allows network switches to operate ML inference tasks on a per-packet basis at line rate.

However, despite the new inference block, the accuracy of a given ML task heavily depends on the availability of and access to input features for inference. When operating inside a switch, these features can either arrive as metadata on packets or via flow tables. For example, Taurus employs the packet parser to extract features

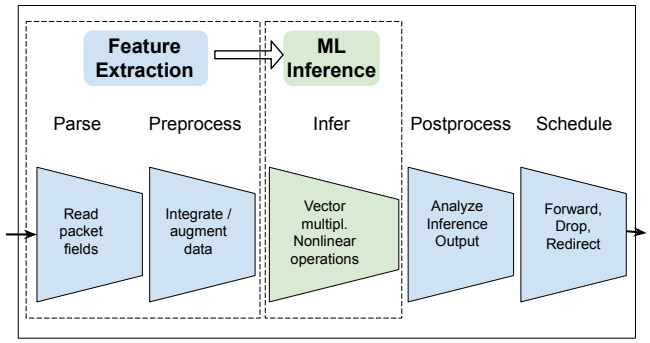

Figure 1: Taurus-enabled switch: a single-node model.

from packets and a sequence of flow tables to preprocess these features before passing them along to the inference block (Figure 1: Parse/Preprocess stages). More generally, the switch retrieves the information inside a packet (header and payload) and combines it with the existing stateful information (e.g., counters)—a process we call *Feature Extraction* (FE). A switch can, for example, evaluate the accumulated number of bytes and packets received for a particular flow.[1] It can also augment it with other (cross-flow) information, which is not direly associated with the flow; for example, the number of total packets received by the switch or the total active flows on a given TCP port. As FE and MLI blocks are operating in the same node in Figure 1, we refer to it as a *single node model*.

Often times a single node (e.g., core or Top-of-the-Rack switch) may not have access to certain features or it might just be infeasible or inefficient to run FE and MLI on the same node. As an example of the first issue (accessibility), the switch does not have access to potentially important features related to the Virtual Machine that originated a packet, such as its current CPU utilization or the number of failed login attempts [6]. Considering the other issue (feasibility and efficiency), the evaluation of flow based features in a switch could be challenging due to the potentially very large number of flows that needs to be monitored. Therefore, we posit that it is necessary to split and distribute the FE and MLI tasks on different nodes [17], and have them share features using a standard interface. In this position paper, we make the case of designing a native interface for FE and MLI inter-communication

[1] Different granularities of a flow can be considered by the switch (e.g., 5-tuple of protocol type, source/destination IP address, source/destination transport layer address or the source/destination MAC address).

when operating in distributed multi-node configuration (such as, in a datacenter network). We discuss the applicability and advantages of such a distributed FE/MLI model and the need for a native inter-communication interface. We also highlight the accompanying open-research challenges and their implications.

2 NATIVE DISTRIBUTED IN-NETWORK MACHINE LEARNING

We posit that there is a need to extract features located elsewhere in a network (e.g., servers and switches) and transmit them to node performing per-packet ML. This can be done in different ways; for example by sending out of band control plane messages from the FE node to the MLI node. On the other hand, to mitigate delays and network overheads, we follow a similar in-band approach as Taurus and propose to forward messages by encoding features as headers on existing traffic flowing in the network. Doing so, therefore, requires extending the data plane (at the network layer) to support the proposed distributed FE/MLI architecture.

We extend the conceptual single-node model (Figure 1) to a more general multi-node model (Figure 3). The nodes that include MLI engines also have accompanying FE components to process features extracted by the previous nodes and, in turn, transmit them to the subsequent nodes. Next, we discuss the scenarios under which the proposed multi-node model yields benefits (§2.1) and how and when to transmit features in a such a model (§2.2).

2.1 When to Use Distributed FE/MLI?

The proposed distributed FE/MLI architecture allows transmitting extracted features between a collection of nodes in a network. Doing so, in turn, would reduce the available capacity on the links and consume compute resources at both the transmitting and receiving nodes. Therefore, it is worth discussing under what scenarios is it feasible to use the proposed approach and in which contexts. Our stand is that the distributed FE/MLI enables scenarios that are not possible using a single-node model, and therefore can yield greater benefits despite the additional overheads.

Benefit #1: Scaling memory and compute. One issue in the single-node model stems from the limited available memory in a node to store state information for the feature-extraction process. For example, the amount of per-flow information that needs to be maintained in the node grows linearly with the number of flows that need to be analyzed. The number of flows in a switch/router that should run the feature extraction in the single-node model could exceed the node capability and capacity. The distributed FE/MLI approach distributes the state information across a number of upstream FE nodes, improving the *scalability* of the system. Each FE node processes a subset of all flows that are crossing a switch, sharing not only state information but also the processing burden.

Benefit #2: Access to remote data. The FE nodes in distributed FE/MLI have *visibility* to information that is not directly available on a switch, operating in a single-node model. Examples of such information include system-level monitoring of data at the end-hosts and virtual machines (e.g., CPU utilization and number of login attempts).

Figure 2: Distributed approach for Feature Extraction and ML Inference: Distributed FE/MLI.

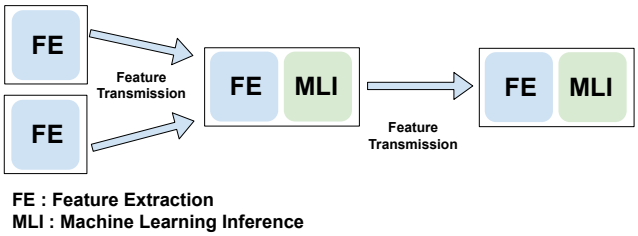

Figure 3: General scenario for Distributed FE/MLI.

Benefit:#3: Complex FE processing at the end-hosts. A distributed FE component running at the end-hosts can exercise the full *flexibility* and capabilities of general-purpose software (as opposed to switch-level hardware) to execute more complex- and custom-processing algorithms for feature extraction. For instance, in a multi-tenant data center (Figure 4, we can have FE modules that execute inside VMs (using eBPF [19]), and a hardware-based ML module working at the aggregation switch level. The software FE modules scale with the number of VMs and have access to all the VM- and application-level information (e.g., CPU utilization and failed login attempts); whereas, the hardware ML modules work at line-rate and can make packet-level decisions.

2.2 How & When to Transmit Features?

There can be various mechanisms to transmit features, which are extracted by an FE node, towards the nodes that run the MLI engine, based on their location (e.g., management, control, or data plane). These different mechanisms operate at varying time scales. The management-plane solutions are the slowest., as they require the extracted features to be collected and distributed via a centralized management system. The control-plane approaches involves control elements running inside the FE and MLI nodes, exchanging control messages that carry the extracted features. These approaches operate on a similar time scale as the management plane and, therefore, introduce latency overheads that can hinder the execution of the desired per-packet ML tasks (as discussed in Taurus [17]). To operate at per-packet time scales, the solution therefore is to extend the network data plane (both physical and virtual) to support the proposed distributed FE/MLI architecture—transmitting features in-band along with the packets.

We will discuss in §4 how to extend the data-plane packets (in the context of the IPv6 network layer) to carry and transmit the extracted features between FE and MLI nodes. Supporting faster reaction time would require adding features to every packet; however, this will induce considerable overhead (1) in terms of the amount of information (features) to encode in the packet as well as the processing time to extract and write these features into the packets on each FE node; and (2) in terms of the processing load to read and process them on the MLI nodes. Depending on the particular MLI

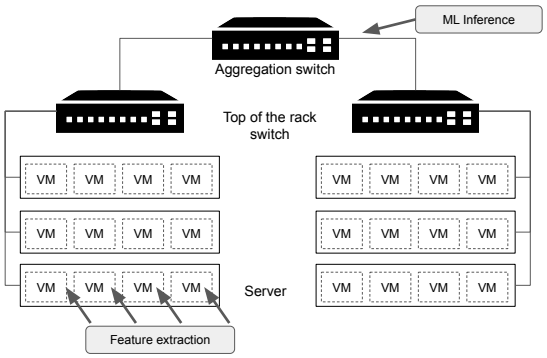

Figure 4: Distributed FE/MLI model applied to a multi-tenant data center.

use case, the extraction, encoding, and transmission of FE features on every packet might not be necessary. To save the bandwidth and processing overhead, we can therefore use a sampling approach. On average, the features are transmitted every N packets for each flow or based on a timer with a rate limit (e.g., at most on every millisecond). There is a tradeoff in implementing such an approach to optimize for detection/reaction time of an MLI task versus network and processing overheads. As such, careful consideration much be taken when employing this approach. We identify this tradeoff as a research challenge that deserves future attention and leave it as future work here.

3 FEATURE DEFINITION & EXTRACTION

Feature engineering is a complex process that can be hard to automate. It is deeply correlated with the specific problem that machine learning has to address, making it difficult to specifically identify the feature set.

3.1 In-band Transmission of Features

In the networking context, we can make a broad classification of features as *per-packet features*, *per-flow features*, and *cross-flow features*.

Examples of per-packet features include those extracted by tools. such as nPrint [11], which provides a convenient representation of packets in a ML friendly format, e.g., associating more than 480 features to IPv4 headers.

However, for the scenario we are discussing in this paper, the above approach is not feasible:

- **Per-packet features lead to redundancy when transmitted in-band.** Carrying per-packet features essentially duplicates the data, which is already present in the packet.
- **Per-packet features bring limited information**: The network state can be better characterized using aggregated information at the flow (or flowlet) level, as evidenced through several work, e.g., [3, 8]. Also, in nPrint [11], the Snowflake classification uses 43 packets (DTLS handshake): single packet information is actionable primarily when correlated with other information of the same flow.

For these reasons, in what follows we will focus on features related to a specific flow (flow-level features) or to a bunch of flows (cross-flow features). It is important to note that we are not considering the final statistic of a flow at the end of the flow, but we are considering features that dynamically change on a packet-by-packet basis during the lifetime of the flow. Examples are the accumulated number of bytes and packets transmitted in a flow. We also refer to these statistics as *window-based* meaning that they are referred to a temporal window of observation and not to the overall lifetime of the flow.

3.2 Attempts at Defining Per-flow Features

A conventional definition of a flow is given by NetFlow, a solution that was invented by Cisco to measure and export flow-level data in routers [7]. Following such definition, a flow is a sequence of packets with the same ingress interface, source and destination IP addresses, IP protocol, source and destination ports, and IP Type of Service.

Then IETF standardized IPFIX [10], a push protocol to monitor and export flow-related information, which has been derived from NetFlow v9 (RFC 3954). The protocol is specified in a series of RFCs (e.g., RFC 3917, 7011 and 6313) which define the architecture, requirements and protocol aspects. For what concerns the feature definition, the IPFIX Information Model, defined in RFC 7012, defines the Information Elements with their Data Types (e.g., `unsigned16`, `ipv4Address`, `dateTimeNanoseconds`) and their Semantics (e.g., `totalCounter`, `deltaCounter`). The complete and updated list of Information Elements is made available by the IANA "IPFIX Information Elements" registry [12] where , currently, 491 different Information Elements are reported.

However, if we look at the feature-extraction software, such as Cisco's Joy[2] used in [2], we can see many deviations from the standards. For instance it adds much more features regarding TLS fingerprints, to enhance encrypted traffic analytics. This is an example of how, in practice, there does not exist a reference set of universally adopted flow-based features, since it varies with the specific needs of the applications.

3.3 Going Beyond Flow-based Features

A flow can be characterized by a number of features, such as `bandwidth`, `packet count` or `bytes exchanged`. It can also have more complex indicators (such as histogram of tcp flag counting) or distribution-based features (such as number of packets with header lengths between 28 and 40).

Most of these features have been standardized by IPFIX, however, these may not be enough for some machine-learning application. For instance, in all the three dataset provided by NetML [3], a proprietary flow feature-extraction library has been used to extract several TLS, DNS, HTTP, and other metadata flow features from raw traffic. Moreover, additional features can represent *cross-flow* information, such as how many flows start from the same node and expanding the set from a single flow to a graph-of-flows information. Example of cross-flow features in ML are provided in [8], which uses the `srv_count` feature defined as the number of connections to the same service as the current connection in the past two seconds. Such features provide a concise information about the network graph to the ML module.

[2] https://github.com/cisco/joy

It is worth mentioning that we can also have device- or software-related features mixed inside network-related ones. For instance, in the popular NSL-KDD dataset [6], we have security-related features (such as the `num_failed_logins` or `su_attempted` that are outside the networking domain, but which can provide a valid knowledge to characterixe the intent of a network flow.

3.4 Challenges with Extracting Features

The problem of feature extraction and transmission is mainly related to the size of the tables needed to contain the information for the different flows. In order to reduce the memory occupancy and, thus, the subsequent amount of information that must be transferred to the ML algorithms, it is better to act by aggregating the measurements of the different flows or by applying techniques to evaluate the statistical characteristics of the packets.

For instance Flowlens [5] learns at line rate the statistical distribution of the packets and propose a compact way to represents the distributions called flow-markers. If the choice of quantization coefficients and truncation level of the distribution are made appropriately, it is possible to control the performance degradation of ML algorithms caused by compression.

Thus, using techniques such as flow-markers or similar, it is possible to (i) extract flow-level features at line rate, even in a distributed manner; (ii) transport these features in packets in a compressed manner, thanks to compression formats that are designed to preserve the significant characteristics of the distributions characterizing the flows; and (iii) use this information to provide the switches with a view of the global network state so that the ML engine is not limited to inference using local state only.

4 ENCODING FEATURES IN DATA PLANE

In our proposed distributed FE/MLI architecture, the features are extracted in FE node and transmitted to MLI nodes (in the data plane). To do so, the extracted information needs to be carried inside packets and processed by the receiving MLI node at the data-plane level—to perform per-packet ML with minimal delay in the ML analysis (and reaction). As packets sizes (and count) lead to network traffic overhead, transmitted feature in-band in the network requires efficient encoding schemes to reduce the transmission overhead (in terms of additional bytes needed) and the processing overhead associated with the writing and reading of these features and the FE and MLI nodes.

Existing Tag-Length-Value (TLV) solutions (as used for example in the IPFIX encoding) will not suffice in this context. Traditional TLV encoding has the advantage of being self descriptive, e.g., by looping up the Tags in the IPFIX specification one can decode the IPFIX message. However, this self-descriptive property results in additional overhead (in terms of bytes per packet). Therefore, to transfer features, its is better to carry information in binary format as a byte array, with no explicit tags. We call this byte array an *Encoded Features Representation* or an *EFR record*. In the distributed FE/MLI architecture, the FE node extracts features, create an EFR record, and transmit it in-band using the data plane. The MLI node (or switch) on receiving the EFR record, parses the EFR header and rung the MLI tasks directly on the incoming bytes (provided as input the MLI engine).

ID	Structure of EFR
x	num of packets (2 bytes), num of bytes (4 bytes), duration in ms (4 bytes)
y	duration in ns (8 bytes), num of packets (2 bytes)

Table 1: An example definition of EFR records.

4.1 The EFR Record

The definition of an EFR record corresponds to the definition of a fixed-sized *struct* in the C programming language: an EFR is composed of a sequence of features, and each feature is encoded in binary format with a specific number of bytes. Different ML applications require different sets of features; hence, different EFR record structures will need to be defined. The sender and receiver of a given EFR record should both be aware of its content: the exact sequence of features it carries, and the size of each feature in bytes. The idea is to associate an identifier (agreed upon by the transmitter and receiver) to describe the EFR record structure, i.e., the sequence of features (and their format) that are carried in the EFR record. In this way, each packet can carry a different type of EFR record identified by a different ID. As an example, in Table 1, two EFR records are defined, a record of 12 bytes that carries three features, has a fixed length of 12 bytes, and is associated to the ID x as well as a record of 10 bytes that carries two features, has a length of 10 bytes and the ID y. (In comparison, the array of bytes containing the features used by the Inference processor in Taurus is called Packet Header Vector or PHV.)

4.2 Standardization Aspects and EIP

We believe that there are different aspects that are relevant for standardization:

(1) How to carry the encoded features (e.g., the proposed EFR record) in a given network protocol (in particular, we focus on the IPv6 data plane)?
(2) Which features are used and how they are encoded (i.e., the format and the content of the EFR records)?
(3) What set of potential features to consider?

We advocate a "lightweight" standardization approach, only focusing on the first aspect, i.e., how to carry the EFR record in IPv6. We believe that the format and content of the EFR record is dependent on the specific use case and any standardization effort should enable support for defining these individual formats in arbitrary ways per application.

Using EIP to carry the EFR Record. Extensible In-band Processing (EIP) [15, 16] is a framework to add information in IPv6 packet headers to support different use cases. The EIP framework can be extended to support new use cases by adding specific Information Elements. An open source prototype of the framework is available at [1].

The EIP framework can be easily extended to support the distributed FE/MLI architecture. A new Information Element in the EIP header, called the EFR (Encoded Features Representation) is needed. The EFR information element contains: (i) the identifier

that is used by the receiver to understand the content and (ii) the EFR record, i.e., the array of bytes with the encoded features.

5 DATASETS AND THEIR LIMITATIONS

Well-defined features are essential to the performance of ML models. Applications like anomaly detection can be assessed on a packet-by-packet basis by interpolating packet-level data from flow-level events to ensure fine-grain security. Systems, like Taurus [17], have had to breakdown existing flow-level data such as the NSL-KDD dataset [6] to reproduce packet-level events. Features that appear in the dataset, such as the duration of the connection or the number of urgent flags, can change on a packet-by-packet basis and need to be monitored on that same granularity. This can also be seen with the use of partial packet length and inter-arrival time histograms seen in applications like Botnet Chatter detection [5]. Structures that aggregate information can be useful for training in their finished forms, but in order to detect events at the appropriate scale, they must be used for inference in their incomplete, packet-level forms.

5.1 Challenges with Existing Datasets

Although the performance of ML applications (and network-ML applications by extension) are largely dependent on their datasets, modern datasets in networking fall short in a number of ways. Datasets tend to be assembled in a haphazard way and cater only towards a singular application.

Limitation #1: Packet-level features vs flow-level features. Most current networking datasets contain flow-level features, primarily to reduce the size of the dataset. ML training relies on drawing correlations between high-dimensional inputs. To do this, large quantities of data are necessary as well as many faceted inputs to draw correlations between. In addition, many network events are not confined to the flow-level. Events like security breaches, congestion window modifications, queue management configuration changes, etc., can all occur at sub-flow, flow-let, or packet levels. Removing access to timescales like this from datasets prevents models from properly picking up on these events, inhibiting their effectiveness. Condensing traffic information to the flow-level trades off data size for the accuracy of ML models and is antithetical to the use of ML in general.

Limitation #2: Need for real-time statistics in datasets. In addition to the added fidelity of having packet-level features, flow-level features can also benefit from being broken down into sub-flow data samples, we call them *window-based* features. In this way, instead of providing just a single value for a given feature of a flow which is computed when the flow ends, we can have a temporal series of values. As shown in the Homunculus compiler [18], working with aggregated statistics made from flows (or even multiple flows) can still yield good results when operating on partially constructed statistics (i.e., sub-flow level data) by training with completed statistics and running predictions on incomplete statistics. Because they use the same datasets, the effect of working with partial histograms can be applied to other security applications like Website Fingerprinting and Covert Channel Detection [5]. The predictions will not signal detection of an event until the partially constructed statistic reaches the fidelity of the completed one. In these cases, the detection is immediate rather than alternative methodologies where flows are logged and the aggregate statistics are calculated offline at regular intervals.

Limitation #3: Consensus on reference network ML datasets. With ML-networking research still in its infancy, datasets are sparse and vary wildly from task to task. The lack of a common format or methodology in constructing these datasets has a deleterious effect on both research in this area as well as industry adoption of ML techniques. In addition to the excess work required to adapt these datasets to existing ML infrastructure, the inconsistency in formatting makes it difficult to extend these datasets as well. Harmonization of header fields for ML means that models need to be trained on datasets with these same harmonized fields as data samples.

6 RELATED WORK

In-Band Processing and Telemetry. In addition to passing information within network elements for consumption, the use of in-band telemetry to collect data-plane information can also be beneficial to the control plane. Geng et al. showed that their system SIMON [9] can send query packets into the data plane to collect information about devices in the network and reconstruct queue states and predict throughout of the network. The SIMON system also uses machine-learning inference to reconstruct the global state. In this scenario, the feature extraction elements are still located in the data plane, but the machine-learning inference is moved to the control plane.

The problem of feature transmission in in-band telemetry systems has been addressed extensively in the literature. In order to reduce the data flow, it is possible to sample traffic (as in [13]) or use probabilistic data structures to obtain limited error representations (as in [21]). Hybrid solutions, such as Flowlens [5], discussed in §3.1, seem to provide a good compromise between implementation complexity and representation efficiency.

Feature Encoding and Extraction. nPrint [11] is proposed by Holland et al. to extract features from packets in a way that is convenient for machine learning. This approach is suited for local data preprocessing but not for multi-node in-network ML, as discussed in §3.1. IPFIX [10] is a standard protocol for sending IP flow data from network devices to collector systems for network monitoring and analysis. It is a management plane protocol and is also not suitable for sending information within the data plane.

7 CONCLUSION

To conclude, let us review our position and recall the open research challenges. Considering the application of machine learning to networking, we identified two emerging needs:

(1) The need for more investigation on **distributed systems** performing machine learning at the networking level. The spatial distribution of the elements improves the *scalability* of the system, and provide hybrid systems the advantages of both *software and hardware* elements.

(2) The need for further investigation on **data-plane solutions** that enable fast communication and timely reaction in ML subsystems. Here, one of the key enablers is the advent of hardware-based ML networking devices to run inference at line rate [17]. In turn, this calls for a continuous in-band flow of information to adequately feed such ML inference engines.

Several specific issues and challenges arise from these two main needs. Among them, we highlight:

- The definition of new mechanisms to allow entities to communicate and exchange ML data in the data plane (e.g., to exchange features or weights). In particular, we have proposed to use the EIP framework.
- The production of new network datasets and the convergence of the community towards "reference" challenges and performance metrics to measure progress, as is happening in other research communities (e.g., an equivalent of the MNIST dataset for handwritten digit classification).
- The improvement of window-based feature extraction (flow and cross-flow), to trade off resource usage and latency.

REFERENCES

[1] Anon. 2022. Extensible In-band Processing (EIP) Home Page. https://eip-home.github.io/eip/. Accessed: 2022-09-29.

[2] Onur Barut, Matthew Grohotolski, Connor DiLeo, Yan Luo, Peilong Li, and Tong Zhang. 2020. Machine Learning Based Malware Detection on Encrypted Traffic: A Comprehensive Performance Study. In *7th International Conference on Networking, Systems and Security*. 45–55.

[3] Onur Barut, Yan Luo, Tong Zhang, Weigang Li, and Peilong Li. 2020. NetML: A Challenge for Network Traffic Analytics. *CoRR* (2020). arXiv:2004.13006 https://arxiv.org/abs/2004.13006

[4] Coralie Busse-Grawitz, Roland Meier, Alexander Dietmüller, Tobias Bühler, and Laurent Vanbever. 2019. pforest: In-network inference with random forests. *arXiv preprint arXiv:1909.05680* (2019).

[5] D. Barradas et al. 2021. FlowLens: Enabling Efficient Flow Classification for ML-based Network Security Applications. In *Network and Distributed System Security (NDSS) Symposium*.

[6] L Dhanabal and SP Shantharajah. 2015. A study on NSL-KDD dataset for intrusion detection system based on classification algorithms. *International journal of advanced research in computer and communication engineering* 4, 6 (2015), 446–452.

[7] Cristian Estan, Ken Keys, David Moore, and George Varghese. 2004. Building a better NetFlow. *ACM SIGCOMM Computer Communication Review* 34, 4 (2004), 245–256.

[8] Tang Tuan A et al. 2016. Deep learning approach for network intrusion detection in software defined networking. In *2016 international conference on wireless networks and mobile communications*. IEEE.

[9] Yilong Geng, Shiyu Liu, Zi Yin, Ashish Naik, Balaji Prabhakar, Mendel Rosenblum, and Amin Vahdat. 2019. SIMON: A Simple and Scalable Method for Sensing, Inference and Measurement in Data Center Networks. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. 549–564.

[10] Rick Hofstede, Pavel Čeleda, Brian Trammell, Idilio Drago, Ramin Sadre, Anna Sperotto, and Aiko Pras. 2014. Flow monitoring explained: From packet capture to data analysis with netflow and ipfix. *IEEE Communications Surveys & Tutorials* 16, 4 (2014), 2037–2064.

[11] Jordan Holland, Paul Schmitt, Nick Feamster, and Prateek Mittal. 2021. New Directions in Automated Traffic Analysis. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security* (Virtual Event, Republic of Korea) *(CCS '21)*. Association for Computing Machinery, New York, NY, USA, 3366–3383. https://doi.org/10.1145/3460120.3484758

[12] IANA. [n.d.]. IP Flow Information Export (IPFIX) Entities. https://www.iana.org/assignments/ipfix/ipfix.xhtml

[13] Yuliang Li, Rui Miao, Changhoon Kim, and Minlan Yu. 2016. {FlowRadar}: A Better {NetFlow} for Data Centers. In *13th USENIX symposium on networked systems design and implementation (NSDI 16)*.

[14] N McKeown. 2015. PISA: Protocol Independent Switch Architecture. In *P4 Workshop*.

[15] S. Salsano et al. 2022. Extensible In-band Processing (EIP) Architecture and Framework. draft-eip-arch. https://datatracker.ietf.org/doc/draft-eip-arch Work in Progress.

[16] S. Salsano et al. 2022. Supporting Future Internet Services with Extensible In-Band Processing (EIP). In *Proceedings of the ACM SIGCOMM Workshop on Future of Internet Routing & Addressing* (Amsterdam, Netherlands) *(FIRA '22)*. Association for Computing Machinery, New York, NY, USA, 92–98. https://doi.org/10.1145/3527974.3545727

[17] Tushar Swamy, Alexander Rucker, Muhammad Shahbaz, Ishan Gaur, and Kunle Olukotun. 2022. Taurus: a data plane architecture for per-packet ML. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 1099–1114.

[18] Tushar Swamy, Annus Zulfiqar, Luigi Nardi, Muhammad Shahbaz, and Kunle Olukotun. 2022. Homunculus: Auto-Generating Efficient Data-Plane ML Pipelines for Datacenter Networks. *arXiv preprint arXiv:2206.05592* (2022).

[19] Marcos AM Vieira, Matheus S Castanho, Racyus DG Pacífico, Elerson RS Santos, Eduardo PM Câmara Júnior, and Luiz FM Vieira. 2020. Fast packet processing with ebpf and xdp: Concepts, code, challenges, and applications. *ACM Computing Surveys (CSUR)* 53, 1 (2020), 1–36.

[20] Zhaoqi Xiong and Noa Zilberman. 2019. Do Switches Dream of Machine Learning? Toward In-Network Classification. In *Proceedings of the 18th ACM Workshop on Hot Topics in Networks* (Princeton, NJ, USA) *(HotNets '19)*. Association for Computing Machinery, New York, NY, USA, 25–33. https://doi.org/10.1145/3365609.3365864

[21] Xiwen Yu, Hongli Xu, Da Yao, Haibo Wang, and Liusheng Huang. 2018. CountMax: A lightweight and cooperative sketch measurement for software-defined networks. *IEEE/ACM Transactions on Networking* 26, 6 (2018), 2774–2786.