University of California
Santa Barbara

PERRY: Flexible and scalable data preprocessing system for "ML for Networks" pipelines

A Thesis submitted in partial satisfaction
of the requirements for the degree

Master of Science
in
Computer Science

by

Navya Battula

September 2023

The Dissertation of Navya Battula is approved.

Professor Elizabeth Belding

Professor Amr El Abadi

Professor Arpit Gupta, Committee Chair

August 2023

PERRY: Flexible and scalable data preprocessing system for “ML for Networks” pipelines

Dedicated to Daddy, mom and Deepak who stood as my pillars of support during the toughest times in my Master's journey.

Acknowledgements

My journey through my Master's degree was a challenging one, filled with constant ups and downs. The unwavering support of some remarkable individuals was the beacon of hope that sustained me throughout this journey. Expressing my gratitude towards them in these acknowledgments is the least I can do.

First and foremost, I want to extend my deepest gratitude and love to my parents and my brother. They have been the source of my happiness since the beginning of my Master's journey and the reason I persevered through the toughest times. When things became exceptionally challenging towards the end of my Master's degree, I faced moments of despair and self-doubt. The persistent feeling that had haunted me since the beginning of grad school — "Maybe I wasn't good enough?" — had grown due to circumstances and the people around me. At my lowest point, feeling desperate, unemployed, and deeply depressed, I contemplated giving up. It was during these trying times that my family supported me like no one else could. My Dad would call me every morning just to talk, allowing me to share my feelings and providing a distraction from my hardships. His unwavering moral support during my darkest moments is something I can never repay. My mom's care for my well-being, ensuring I ate properly during those challenging times, was a constant source of comfort. Talking to my brother was therapeutic; he made me feel like the proud big sister and is the best brother anyone could wish for. Thank you, Dad, Mom, and Chintu, for being the best family I could ever ask for. I am forever indebted to you for your support during this difficult period.

Secondly, to my best friend Srilu, I express my heartfelt gratitude for being my ultimate source of support and guidance throughout my Master's journey. Thank you for always being there, offering valuable advice, and helping me process my grief by sharing it with me. You are a friend that not everyone deserves, but I am fortunate to have you.

Thirdly, I want to extend my sincere thanks to Rajesh Varma Uncle and Gowthami Aunty for welcoming me during my challenging times. I am deeply grateful for your care, treating me like your own daughter, and I am forever indebted to you. I also want to mention my adorable nephew Bhavyansh, who brought me solace during this period.

I want to thank my professional colleagues for their mentorship and support during my Master's journey. Special thanks go to Ph.D. student and fellow lab mate Roman Beltiukov for guiding me in my work, your patience, and unwavering support. I also extend my gratitude to my coworker and fellow lab mate Satyandra Guthula for your support and patience in our project. Your wisdom and mentorship contributed significantly to the development of PERRY. I also want to thank all my lab mates for their guidance and support during my Master's journey.

I am grateful to Professor Elizabeth Belding, Professor Amr El Abadi, and Professor Arpit Gupta for being on my committee.

Describing my Master's degree as just a milestone would be an understatement. I thank everyone who has been part of this incredible journey. To anyone reading this, I want to share a message based on my experience as a grad student at UCSB. I worked hard to secure a place at a prestigious institution like UCSB, coming from a less prestigious undergraduate school. It was natural for me to experience imposter syndrome for a while. But when you realize that imposter syndrome is taking over, seek help from mental health services, change your work environment, and surround yourself with positive influences before it's too late. I hope this message resonates with you.

Love,

Navya

Abstract

PERRY: Flexible and scalable data preprocessing system for "ML for Networks" pipelines

by

Navya Battula

The integration of machine learning techniques into networking research has catalyzed significant advancements in areas such as traffic classification, intrusion detection, and quality of experience (QoE) estimation. This progress has been fueled by remarkable developments in deep learning, leading to state-of-the-art models in various domains, leveraging powerful neural networks, encoders, transformers, and language model architectures.

Developing these complex ML-based models relies heavily on the data pre-processing module to extract features from the raw network data (e.g., packet traces) and add labels to different data points. Different model specifications require extracting disparate sets of features. Currently, there is a tight coupling between the data pre-processing and model training modules in the ML pipelines used for developing ML artifacts for networking. Specifically, the pre-processing modules are only suited to extract a limited set of features (e.g., extract time series features) that are suitable for specific downstream model specifications (e.g., LSTM). Consequently, researchers exploring new learning models for different networking problems end up spending a significant amount of their time developing custom data pre-processing modules, impeding the pace of innovation.

This thesis focuses on decoupling data pre-processing from model training in ML pipelines for networking. Specifically, we present the design and implementation of PERRY, a flexible data pre-processing module for networking that can extract a wide

range of (high-quality) features at scale that can be consumed by disparate model specifications for model training. PERRY offers an intuitive user interface that allows developers to express their data pre-processing intents. More concretely, PERRY supports three distinct classes of features: packet content, time series, and aggregate statistics. For each class, it lets the user specify different parameters. For instance, the user can express which set of fields (e.g., timestamp, number of bytes, etc.) to use for time series features and at what granularity (e.g., per packet, burst, or flow). Similarly, it lets the user select which set of aggregate features to extract and at what granularity.

To scale the pre-processing tasks, PERRY leverages state-of-the-art data analytics and storage tools—making the best use of limited computing and storage resources. Specifically, it decomposes the pre-processing task at flow-level granularity. Such decomposition offers horizontal scalability offered by existing tools without compromising the semantic integrity of the extracted features. Further, to minimize wasteful data processing, it offers a hybrid schema that aims to strike a balance between expressiveness and scale. Specifically, this schema only exposes a subset of popular features to the user, offering pointers to raw data. Such an approach ensures that only a subset of features is extracted for network traffic, and more complex features are dynamically extracted from a subset of network traffic on demand. By decoupling data pre-processing and model training in ML pipelines for networking, PERRY lowers the threshold for developing new ML models in networking. PERRY represents a step forward in simplifying and enhancing data processing in networking research and opens new possibilities for future innovations in the field.

Contents

Chapter 1

Introduction

Constructing networking systems necessitates a comprehensive understanding of the network's overall state, enabling developers to align the system's design with specific requirements. However, this endeavor encounters a fundamental obstacle: the restricted visibility into the network's state. Achieving a holistic perspective is imperative for the successful development of any robust networking system. Additionally, it is important to note that networks are characterized by intricate closed-loop interactions occurring across various spatial and temporal scales. These interactions encompass diverse patterns that can be harnessed to glean insights into the network's condition. Consequently, the incorporation of learning mechanisms becomes imperative, enabling the utilization of these patterns to address the challenge of limited visibility into the network's state.

Learning can be effectively applied to networking solutions through two distinct approaches. The first approach entails the utilization of uncomplicated rule-based heuristics, which operate by leveraging explicitly defined rules and algorithms. To illustrate, consider the adaptive bitrate algorithm used in video streaming applications. This algorithm forecasts the forthcoming channel bandwidth and optimally adjusts the bit rate to align with the prevailing streaming requirements, ensuring an enhanced streaming experience devoid of pixelation and buffering interruptions. Another instance of rule-based heuristics includes congestion control algorithms, as well as selected traffic classification algorithms employing port numbers to categorize the protocol within ongoing traffic. While simple heuristics are notably efficient in numerous scenarios, they do exhibit limitations when confronted with dynamic network conditions.

To address the challenges posed by the limitations of simple heuristics under dynamic network conditions, we turn to the efficacy of Machine Learning techniques, which have demonstrated notable efficiency when compared to their predecessors. Both conventional machine learning algorithms and modern neural network architectures have exhibited effectiveness in discerning inherent patterns and adeptly adapting to the intricacies of dynamic network scenarios. Moreover, these Machine Learning techniques are adept at handling various data types, extracting diverse patterns, thereby enhancing their capacity for more comprehensive learning.

Early approaches to Machine Learning in Networking primarily relied on straightforward traditional ML algorithms such as Random Forest and Naive Bayes, featuring simpler workflows. Data were typically gathered in the form of packet traces, and processing involved deploying pre-existing tools tailored to extract a narrow range of specific features. The feature space was often rudimentary and fixed, and model tuning lacked sophistication.

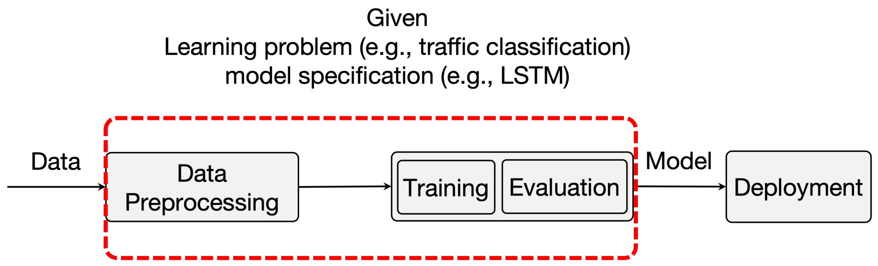
However, the landscape shifted dramatically with the emergence of well-defined neural networks. These networks brought the capability to process diverse data types, encompassing text, images, and sequence patterns. This transformative shift revolutionized the application of machine learning in developing cutting-edge learning solutions, a shift that was equally apparent in networking contexts.

This transition empowered researchers to delve into neural network architectures, processing data in distinct formats such as raw byte-based image representation or extracting time series features through network packet attributes. This concept spurred significant research, culminating in machine learning becoming a preferred approach for cultivating enhanced learning and constructing advanced systems within the realm of networking.

1.1 Existing challenge

Illustrated in Figure 1.1, a conventional Machine Learning (ML) workflow for networks follows a structure encompassing both data processing and model training modules. When addressing a specific problem, the workflow should ideally align with its specifications. For instance, when tackling a traffic classification issue, the process would involve transforming packet traces into time series data using a suitably tailored data processing module, optimally designed for the intended LSTM model.

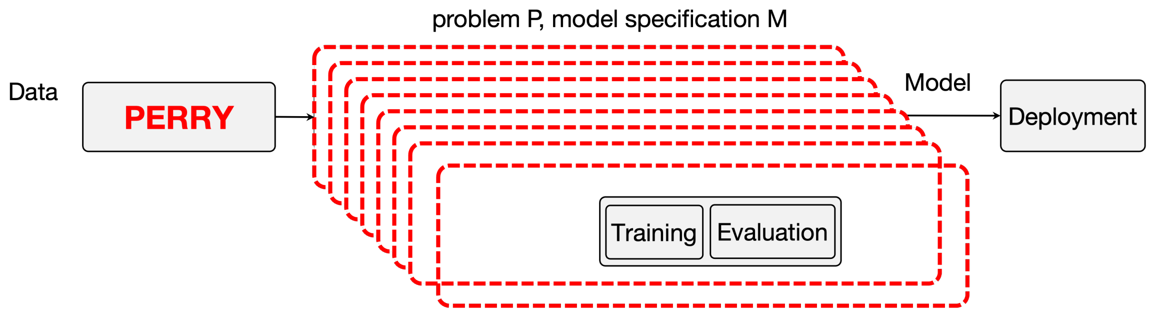
Figure 1.1: A typical machine learning workflow.


Given
Learning problem (e.g., traffic classification)
model specification (e.g., LSTM)
Data
Data Preprocessing
Training
Evaluation
Model
Deployment

However, challenges arise when even slight modifications to the problem specification prompt consequential ripples throughout the pipeline. This predicament stems from the inherent tight coupling that exists between the data processing and model training modules within the ML workflow. Such rigid coupling confines data processing to the specific model at hand, offering minimal adaptability when confronted with evolving problem specifications.

Consequently, as the system becomes governed by two interdependent components, complexity intensifies, making the incorporation of adjustments a convoluted endeavor. Each minor alteration inadvertently introduces redundant supplementary processing steps, thereby needlessly compounding the complexity. As we can observe in Figure 1.2, numerous iterations create multiple copies of the current pipeline, which are often redundant and unnecessary.

Figure 1.2: The iterative modifications result in a lagged progress and hamper productivity



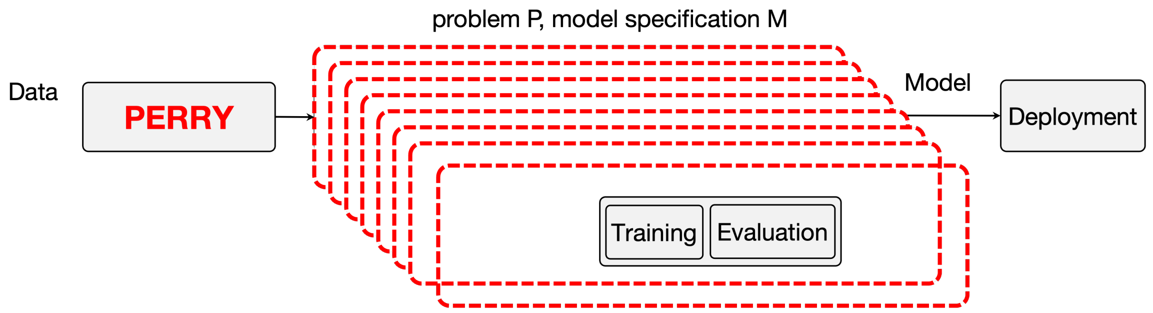
The iterative refinement of the data processing module consumes valuable time and hampers productivity that could otherwise be channeled into devising efficient models. Repeatedly reconfiguring and reconstructing pipelines to suit specific demands does not yield innovation or improvement; it merely alters the approach. Consequently, the pursuit of a generic data processing system emerges as a far more compelling endeavor, poised to alleviate this burden on developers.

The true challenge lies in skillfully crafting a framework that comprehensively addresses all data processing requirements, enabling the effective decoupling of data processing from model training. Herein enters our solution, **PERRY: A Flexible and Scalable Data Preprocessing System for "ML for Networks" Pipelines**.

1.2 Our solution

As previously indicated, our primary emphasis has been on formulating a network data processing framework that effectively tackles the conundrum of the closely interlinked data processing and model design components. In this context, we have meticulously crafted our framework, PERRY, with a distinct objective: to extricate dependencies on models and endow users with a seamless avenue for data processing through an intuitive user interface. By doing so, it streamlines the data processing phase, negating the necessity for supplementary development efforts. PERRY stands as a comprehensive solution capable of processing network data across diverse types and granularity. Refer to Figure 1.3, which illustrates the significant transformations introduced by PERRY in the system. PERRY simplifies the system by consolidating the number of moving parts into just one. As a result, researchers can dedicate more time to developing improved models and making well-informed decisions.

Figure 1.3: Our proposed workflow with PERRY. Notice how PERRY creates a separate data processing platform making it independent of other components in the pipeline.



The subsequent sections of this thesis are structured as follows: In Chapter 2, we delve into the realm of related works, painting a comprehensive picture of current network data processing solutions. Through this exploration, we aim to discern how our present model has been shaped, taking inspiration from the challenges addressed in these existing works. Chapter 3 provides a holistic overview of network data processing tasks, encompassing feature sets and the design objectives we embraced in the development of our framework. The subsequent Chapter 4 delves into the architecture we have devised for our framework, meticulously dissecting each constituent element in detail. Turning to Chapter 5, we elucidate the performance evaluation results of our framework, closely observing the impact of the optimization techniques we implemented on the run time of the pipeline framework. Chapter 6 offers the culmination, presenting the concluding remarks and insights into potential future directions. In the Appendix section, we furnish an exhaustive account of the diverse network feature spaces employed, organized in tabular formats for ease of reference.

Chapter 2

Related Work

2.1 Literature Review

The challenge of processing networking data has persisted over time. Initial works heavily relied on tools like NetMate [1] to process data, generate flows, and compute feature values for datasets. The data processing requirements then were minimal owing to the lower complexity of the models and the problem specifications.

Works such as [2], [3], and [4] utilized traditional machine learning models for traffic classification, with [2] and [3] focusing on flow-based statistics for protocol identification (DNS, SMTP, HTTP) and encrypted traffic classification, respectively, utilizing tools like NetMate for data processing. NetMate was the data processing standard heavily employed in the early prominent works in the field. However, the landscape of learning in network changed soon, leading to dramatic changes in data processing techniques.

Authors in [5] employed 248 flow features for supervised Naive Bayes classification, encompassing packet-wise time series features like inter-arrival time and packet size. This kind of work marked the beginning of exploration into a diverse feature space in network data.

Later approaches, exemplified by [6], [7], [8], and [9], heavily relied on flow statistics using flowmeters like ISCX Flowmeter (CICFlowmeter) [10] and YAF [11]. Drastic changes were observed with the rise of deep neural networks [12], [13], [14], [15], [16], triggering a paradigm shift in networking machine learning models and the representation of network data.

Approaches that incorporate spatio-temporal attributes [17] and methods leveraging raw byte formats [18], time series features, and various derived attributes have become widespread due to the capabilities of neural networks in processing them. Consequently, the data processing step has become more and more specific, offering a tightly coupled architecture incorporated with the model training part that necessitated the installation of numerous flowmeters and constructing specific pipelines for data extraction in specific formats. Despite advancements in model architectures and networking solutions, the data processing step remains rudimentary with minimum to no attention paid to solving this.

The solution to this challenge is surprisingly simple when we analyze network data formats through the lens of three fundamental categories: Packet Content Features, Time Series Features, and Flow-wise Statistical Features. Our exploration of various works in the field that employ network datasets for training models revealed that the majority of these formats, as outlined in prevalent literature, can be categorized within these three foundational data formats. In this section, we compile references from the literature to substantiate and elucidate our selection of these three formats. We also delineate the distinctions of PERRY from other approaches and highlight the challenges that other methods present to networking researchers. Subsequently, we delve into renowned contributions within the community and emphasize challenges that have arisen from their designs. These challenges serve as a source of inspiration for the development of our framework, which aims to effectively address and mitigate them.

The first part of our literature review is structured around these three primary network data formats. We provide succinct insights into the data formats employed in these works, elucidate their methodologies, and detail how our approach, PERRY, efficiently incorporates them.

1. **Flow aggregate statistics:** In the networking community, flowmeters are widely embraced tools for extracting and assessing various aspects of flows, encompassing flow-based attributes like statistics and byte counts. Previous methodologies heavily emphasized flow statistical features due to their inherent simplicity and capacity to encapsulate diverse flow information. Within the landscape of flow data, three predominant categories emerge based on established data export standards: NetFlow (Network Flow), IPFIX (Internet Protocol Flow Information Export), and S-Flow. While these standards exhibit similarities, they are differentiated by their data export conventions. IPFIX, serving as an open standard, has been adopted by flowmeters like Argus [19] and YAF [11], due to its expansive support for various record formats and adaptable design. However, YAF's intricate configuration and composite 38-feature set necessitate further preprocessing to segregate distinct data formats. Conversely, flowmeters such as nProbe [20], centered around the NetFlow format, have been pivotal in generating flow data and inspiring the development of YAF.

   Despite the multitude of flowmeters and data export standards, a comprehensive representation capable of concisely encapsulating aggregate flow statistics remains elusive. Flowmeters such as YAF and Argus adhere to specific export formats and have been employed in certain studies for feature extraction purposes. However, their feature sets comprise a mixture of aggregate statistics, specific packet content features, and traces of other data features. This combination of features can pose challenges when trying to use them as generic flowmeters, as this set of features might not be optimally suited for all types of problem statements.

In our exploration, we acknowledge CICFlowmeter's [10] efficacy in capturing crucial flow aggregate statistical features separately without any mix-up, as evidenced by its widespread adoption in significant research endeavors [6], [7], [8], [9] for constructing feature sets and training data for models. When juxtaposed against alternative flowmeters, CICFlowmeter's performance shines through due to its proficiency in data representation and seamless integration with tools. Its user-friendly interface and versatility in harmonizing with various components further validate its pertinence as a valuable addition to our framework. We provide a brief account of the flow aggregate statistical features in the next chapter.

2. **Time Series Sequences and Data:** The advent of sequential models in deep learning has markedly facilitated the handling of time series data. Architectures such as Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM), and Gated Recurrent Unit (GRU) have wrought transformations in time series forecasting, prediction, and classification, sparking a parallel exploration of their application in network time series data (refer to Appendix A for a comprehensive overview of Packet-wise Time Series data). Interestingly, time series data has historically been included in network feature sets, intermingled with other attributes in prior works [5], but it has surged into the limelight as an individualized data format following the proliferation of sequential models. The realm of time series feature space rapidly gained traction, displaying continuous evolution that consistently yields enhanced outcomes. Owing to its inherently representative character, time series features are harnessed to tackle a spectrum of problems, emerging as a distinctive format adept at capturing multifaceted patterns across varying levels of

granularity.

Numerous studies have thrived by harnessing the time series format across various categories and applications, establishing it as a predominant format within the research community [21], [22], [23], [24]. For instance, in [25], authors extracted time series features in conjunction with IP and TCP headers, employing LSTM for encrypted traffic classification. Similarly, in [26], time series features like packet size and timestamp were utilized, accompanied by statistical, distribution-based, and frequency-based features, temporally taken for the analysis of single-flow time series. Moreover, in [27], flow-based time series features and packet content features were extracted and used for training CNN models, followed by LSTM models. Another approach, outlined in [28], devised an ensemble model that integrates three data formats—aggregate statistics, packet content features, and time series features—each fed individually to distinct networks (MLP, CNN, and LSTM, respectively). The weights of these networks are then connected to a single MLP. This method employs time series features in the form of 1024x3 sequences, comprising packet size, timestamp, and direction. They demonstrate higher F1 scores of 95% using their approach in encrypted traffic classification. Furthermore, in [29], burst-wise time series features were leveraged to model traffic from platforms like Netflix and YouTube, utilizing the Dynamic Adaptive Streaming over HTTP (DASH) protocol. These features were employed as a time series of down/up/all bytes-per-second and down/up/all packets-per-second. By analyzing the "burstiness" of traffic at different time points, they achieved over 98% accuracy in identifying the streamed video titles, even under encryption. Our framework adopts the format presented in [28] for our packet-wise time series representation and the format from [29] for our burst-wise representation, owing to their efficiency in capturing underlying patterns succinctly.

Acquiring time series features entails a consistent workflow across various scenarios. Irrespective of the granularity, the fundamental attributes of timestamps and packet sizes are pivotal. These raw attributes undergo processing and transformation to generate features at diverse levels of granularity, achieved through straightforward grouping and aggregation operations. Within our framework, we have adeptly integrated these functions, empowering users to select the desired granularity and set of time series features for processing. Further elaboration on the time series data format is provided in the subsequent chapter.

3. **Packet content features:** Raw bytes and packet header bytes constitute the most fundamental network data format. Raw packet bytes have maintained prominence over time and have featured in numerous works within the field [27], [30]. The packet content features encompass critical information extracted directly from various headers, rendering them invaluable components of any feature set. These features have been strategically incorporated in tandem with other formats, implicitly acknowledged for their basic nature and direct extraction.

   Additionally, the rise of convolutional neural networks has sparked researchers' interest in exploring packet content features for image representation. For instance, in [31], authors utilize TCP and IP packet headers in conjunction with Bayesian Networks for traffic classification. Similarly, in [32], a selective inclusion of specific fields from various headers forms the basis of packet content feature representation. This feature vector is trained using CNN and SAE for packet-level fingerprinting. Payload bytes are also harnessed for classification, as demonstrated by [33], who employ Support Vector Machine (SVM) for accurate traffic classification. The approach in [28] capitalizes on TLS handshake packets as raw bytes in its ensemble model. Payload bytes, along with flow statistical features, find application in traffic classification through Bayesian networks, as depicted in [34].

Among the explored data representations, we found that, apart from [32], no other approach or representation focused comprehensively on all packet content features. Comparing this approach with [35], we observed that the former exhibited specific feature selection implementation, whereas the latter presented a more generalizable approach. Notably, the nPrint [35] representation stands as a unique standardized packet content representation, featuring a fixed-field, fixed-size structure comprising 1080 fields from IP, TCP, UDP, and ICMP headers, complete with padding for absent fields to maintain consistency. This representation also offers the choice of payload size. Considering these factors, the nPrint representation became our natural choice for integration into our framework.

In the previous section, we established the various data categories and outlined our chosen tools for their processing. In this second part, we delve into the overarching issue we highlighted earlier: the problem of coupling. The heavy interdependence between the data processing module and the model training module serves as a prime example of suboptimal design, introducing numerous complications. This coupling severely restricts flexibility, necessitates redundant redesign efforts, and overall falls short of an ideal system architecture.

For instance, in [36], the authors employ SVM specifically to model TCP traffic, a strategy that may not be suitable for other problem domains. Similarly, [37] introduces a novel approach of converting packet-wise time series features into image representations for CNN-based training. However, this method is closely tied to a specific model specification, making it cumbersome to adapt to different problem specifications. In [38], a unique neural network architecture is proposed for processing datagram-level data, but its data processing approach of segmenting datagrams into byte segments is tailored to its Byte Segment Neural Network architecture. Meanwhile, in [39], genetic algorithms are harnessed for traffic classification, ensuring the selection of crucial features to avoid erroneous predictions. However, their exploration is confined to time series and packet-wise features.

In [28], the authors leverage all three data categories in their respective suitable formats. Nevertheless, a significant challenge lies in the intricate data processing system, intricately designed for their ensemble model. Across these works, the presence of a tight coupling between model training and data processing is evident. In each case, even a slight shift in the problem specification necessitates a comprehensive reimagining of the entire system design, resulting in extensive redevelopment efforts.

As a primary step in our design process, we have resolved to decouple the data processing and model training phases. By implementing this separation, we aim to achieve greater freedom in shaping our data processing framework and, consequently, in designing our model specifications. In the upcoming chapter, we delve deeper into these aspects within the context of our design goals.

Chapter 3

Data and Design

Before delving into the framework's structural intricacies, it is imperative to comprehensively detail data processing tasks, network feature sets, and our design objectives. This entails providing a concise overview of typical network data processing tasks, the formats integrated into our framework, and the foundational principles underpinning our framework's design.

3.1 Network Data Processing Tasks

Network data processing tasks generally encompass two fundamental aspects: feature extraction and labeling. The initial raw packet traces garnered from the network serve as the foundation for extracting diverse feature sets spanning various types (e.g., timeseries) and granularities (e.g., flow level). Feature extraction entails the selective extraction of specific fields from network packets or flows, followed by transformative processing that generates refined datasets. In parallel, the labeling task involves the extraction of hostnames from packets, furnishing insights into their associated services or applications.

Let us try to understand each task in a more detailed way:

1. Feature extraction: Central to the feature extraction process are the considerations of feature category, granularity, and processing approach. Users are empowered to make judicious choices regarding the specific feature category and granularity tailored to their data processing needs. Leveraging data analytics tools, raw packet traces undergo processing to extract targeted features, either from individual packets or higher-level granularities. The complexity of processing varies; while some cases involve straightforward scripting for direct feature extraction in a single step, others demand more intricate procedures, often spanning multiple steps and transformations to attain the desired feature spaces. A comprehensive exploration of feature spaces is expounded upon in the subsequent section.

2. Labeling: The labeling task constitutes an optional step, affording users the discretion to include or omit it. As previously highlighted, the labeling process entails the extraction of hostnames. This extraction is facilitated through the utilization of the Server Name Indication (SNI) field from TLS handshake bytes and/or the Domain Name System (DNS) query name from headers within the payload. These extracted hostnames can subsequently undergo parsing, enabling the assignment of labels based on services or applications.

3.2 Network Feature sets

Network data can be tersely described as either individual packet data or an aggregation of packet data. However, the simplicity of this description belies the intricate and multifaceted nature of the network feature space. A deeper comprehension of this space proves essential for a comprehensive grasp of network data processing.

Broadly classified, network data features can be grouped into three distinct categories. Most contemporary data formats align with these primary categories:

1. Packet Content Features
2. Time Series Features
3. Aggregate Statistics Features

Moreover, network data features can be extracted across three different granularities:

1. Packet Level
2. Burst Level
3. Flow Level Features

This categorization framework furnishes a structured lens through which to perceive the diverse facets of network data.

Let us briefly understand these network data granularities and feature categories.

Network data Granularity

1. Packet Level Granularity: This pertains to features derived from packet-level attributes. Positioned at the lowest stratum within the hierarchy of network data granularity, packet-level features offer a meticulous depiction, encapsulating comprehensive information in intricate detail.

2. Burst Level Granularity: Positioned at an intermediate tier, this level encompasses features pertaining to a temporal grouping of several packets within a network session. Serving as a bridge between packet level and flow level granularity, burst level granularity establishes a vital connection between these two tiers.

3. Flow Level Granularity: Operating at a higher tier of granularity, flow level encompasses the aggregation of features across a singular session. This level of granularity finds frequent application in prominent studies within the field.

Network data Features

1. Packet Content Features: This feature set encompasses a concise array of fields directly extracted from packets. These fields encapsulate headers at the packet level as well as payload bytes. Formulating a unified representation entails the compilation of fields from IP, TCP/UDP, ICMP headers, as well as HTTPS, DNS, and TLS application headers drawn from the payload bytes. Packet content features provide a standardized feature space for the following reasons:

    1. They offer a concise view of the packet-level granularity in its rawest form.

    2. The header fields and payload bytes often contain crucial information for identifying packet details, such as protocols and flags.

    3. Packet content features are straightforward to work with, as they can be easily extracted and processed.

    While packet content features are easy to extract, assembling them in a single representation might be a little tricky. The task of crafting such a representation is inherently challenging, stemming from the variable availability of fields. Amid these challenges, we acknowledge the nPrint approach as a benchmark. nPrint's unique fixed-size, fixed-field representation has emerged as a standard, notable for its consistent implementation across packets, accompanied by appropriate padding to accommodate non-existent fields. This characteristic significantly simplifies developers' engagement with this feature space, negating the necessity for intricate data processing to address inconsistencies.

2. Time Series Features: This feature subset encompasses the time-dependent attributes of network data specific to a given granularity. For instance, at the packet level, time series features comprise packet size, timestamps, and direction.

Renowned for succinctly capturing traffic patterns, time series features provide an accurate reflection of temporal dynamics by unveiling diverse patterns across multiple granularities. Time series features play a pivotal role in the network data feature space for several compelling reasons:

1. Time series features effectively depict the traffic's temporal pattern, offering a precise representation of its dynamic behavior over time.

2. They are easily extracted and processed, requiring no additional tool installation beyond tshark.

3. These features are instrumental in capturing patterns while being protocol-agnostic, making them invaluable for encrypted traffic classification challenges.

4. Time series features can be further transformed into diverse data formats or tokenized for training purposes, leveraging self-supervised and semi-supervised learning tasks.

Typically, time series attributes are extracted at both burst and packet level granularities. At the burst level, attributes such as burst size, timestamps, bytes per second, packets per second, and average packet length are extracted. At the flow level, attributes like packet size, timestamp, and direction are gathered. Extracting time series features from packet traces involves a straightforward process, leveraging raw fields such as timestamps and packet sizes through tools like tshark. Subsequent processing can be seamlessly executed using data processing tools such as pandas and pyspark.

3. Aggregate Statistical Features: This category encompasses statistical attributes such as mean, maximum, minimum, standard deviation, and others, applied to network data features. These aggregate statistical features provide a concise sum-

mary of the features within a given granularity, serving as a vital tool for extracting patterns, particularly in the flow level granularity. Widely employed in prior research for their representative nature, aggregate features contribute significantly to the understanding of the data. The flow-based statistical features emerge as pivotal for three reasons:

1. Flow statistics holistically portray a statistical overview of a flow, enhancing the understanding of its nature.
2. Flow statistical features exhibit a simplified data representation format.
3. Obtaining and training on flow statistical features is comparably straightforward.

For capturing flow level aggregate features, CICFlowmeter has emerged as a prevailing choice. Comprising a standard set of 84 aggregate statistics, CICFlowmeter's offerings are at the forefront of representing flow level granularity. Within our framework, we seamlessly integrate CICFlowmeter to capture aggregate statistical features, a decision guided by the notable effectiveness of this tool.

We make use of the **UCSB dataset** for all our experiments that we describe in our evaluation section. We have obtained the dataset in different sizes to test the efficiency of our framework. Please check chapter 5 for details on this.

3.3 Design goals for PERRY

Selecting appropriate design goals is a pivotal step in constructing any system. When creating our framework, our primary objective was to develop an optimized pipeline capable of fulfilling the data processing needs of networking researchers in a user-friendly manner. In Sections 1.1 and 1.2, we discussed the significance of design for a data processing system and how it impacts the pipeline based on shifts in the problem statement.

To address this, we intentionally made certain decisions to break the interdependence between the data processing and model training steps. With this aim in mind, we aimed to formulate a design that would mitigate this issue, enhancing the framework's flexibility, autonomy, and utility. As a result, we identified two fundamental design goals as the foundation of our framework: Flexibility and Scalability. We firmly believe that our chosen design goals align precisely with the requirements of a framework capable of surpassing existing rudimentary design principles, offering researchers an improved tool to effectively fulfill their data processing needs. Let's delve into each design goal in depth.

1. Flexibility: In the context of our framework, flexibility refers to the extent of freedom granted to users during their interaction with the system. For a data processing system, a flexible framework should empower users to make decisions within the system and maintain control over the entire process. Users should have a significant level of authority over the system, thus enhancing their overall interaction experience. While the notion of flexibility can encompass diverse interpretations and broader perspectives, for the sake of brevity, we will define the aspects of flexibility that characterize PERRY and delve into how we achieve them in the upcoming chapter. The subsequent points outline a few considerations through which we incorporate flexibility into PERRY:

   1. Our framework incorporates an intuitive user interface that ensures ease of use and offers a wide array of options for interaction.

   2. PERRY extends support for diverse feature sets across all granularity levels, saving users substantial time and effort. The data formats generated by PERRY are versatile, enabling users to either utilize them as they are or engage in further processing, tokenization, and more.

   3. Users enjoy comprehensive control over the feature space and data processing

methodologies within our framework. This level of control empowers users to tailor their data processing steps according to their preferences.

2. Scalability: Our second design goal, scalability, is aimed at ensuring the efficiency and reliability of our framework. Even though frameworks with a wide range of features might appear impressive, their performance shortcomings can leave users dissatisfied. Users value a seamless performance experience as much as they appreciate innovation and distinctiveness. Therefore, we have taken care to ensure that PERRY offers outstanding performance and scalability when deployed on the user's end. We have achieved scalability for our framework through the following considerations:

   1. Our scalable pipeline employs a hybrid schema that minimizes unnecessary processing, significantly enhancing framework speed and efficiency.

   2. By adopting a flow-level decomposition design during data processing, we enable easy parallelization, reducing processing times exponentially.

   3. Incorporating various optimization techniques, we ensured expediting the entire process.

These design goals are dedicated to providing an optimal user experience and delivering a state-of-the-art interface that makes data processing tasks easier than ever before. Most importantly, PERRY stands as an intuitive data processing solution for machine learning in network pipelines.

Chapter 4

Proposed Framework

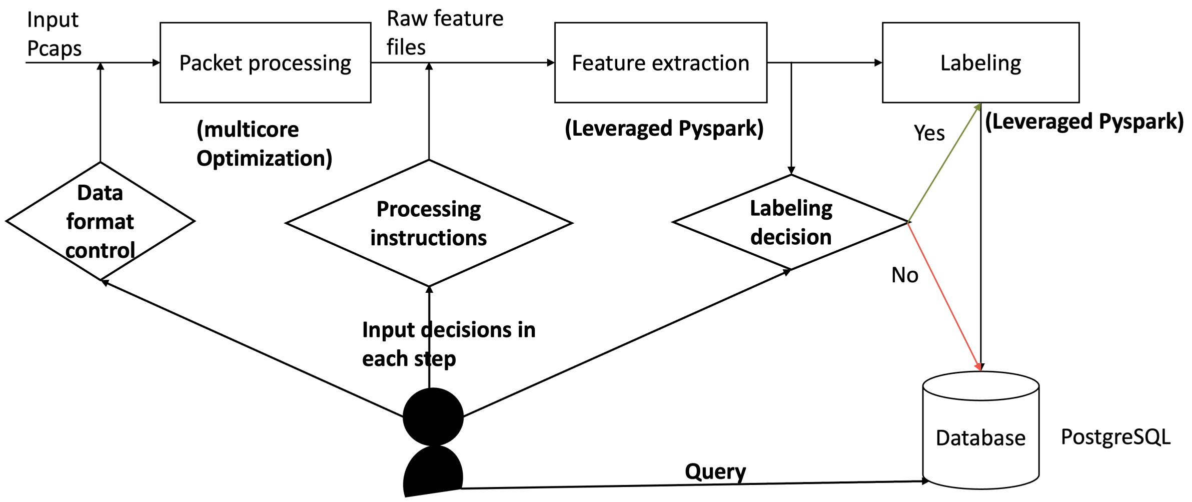
4.1 Architecture

Our proposed framework is underpinned by the dual pillars of scalability and flexibility, incorporating cutting-edge tools such as PySpark and PostgreSQL. Comprising four core modules - Packet Processing, Feature Extraction, Labeling, and Storage & Query - as depicted in Figure 1, users are empowered to make informed decisions at each step of the pipeline. This engagement is facilitated through an intuitive user interface that streamlines interaction with the framework.

The framework harnesses optimization strategies such as multiprocessing and PySpark's features like caching and the omission of User Defined Functions (UDFs), leading to a significant reduction in processing time and a commendable boost in performance. The workflow itself adheres to a straightforward pattern: flow-level data decomposition and feature extraction. This approach streamlines the organization of extracted features in a coherent manner under this uniform level. For instance, in the context of burst-related time series features, data pertaining to all bursts within a single flow can be systematically organized within a CSV file. This method permits the extraction of all features or specifically tailored subsets in line with unique requirements.

The orchestrated arrangement also confers a noteworthy degree of flexibility, substantially enhancing the ease of implementing multiprocessing techniques.

Figure 4.1: The architectural overview of PERRY. Notice the flexibility and Scalability aspects embedded in the framework.

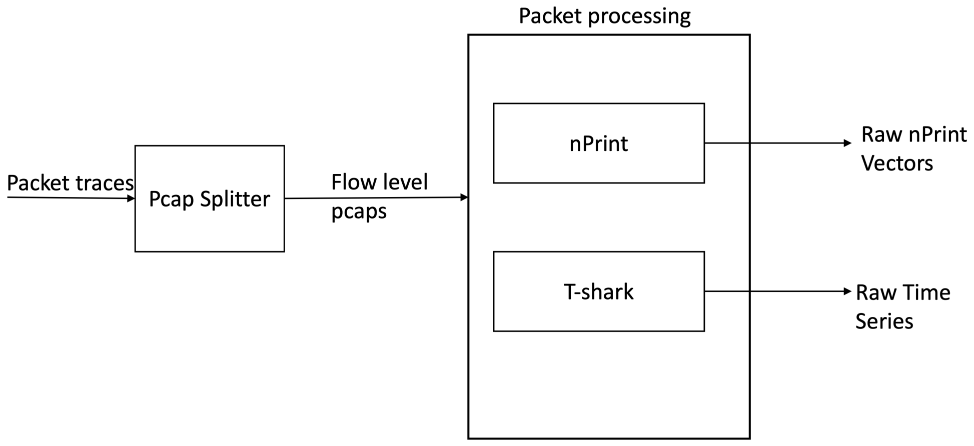


Within the packet processing and feature extraction modules, the focal point lies predominantly on data processing, orchestrating the transformation of raw files into refined and manipulated datasets. Transitioning to the labeling module, users possess the option to imbue their datasets with labels, thus invoking the utilization of Server Name Indication (SNI) and Domain Name System (DNS) protocols to extract host names for individual flows. In the storage and query module, a database framework is employed to house processed data efficiently, laying the foundation for subsequent resourceful information retrieval through queries.

4.2 Packet processing

Within the framework, the packet processing segment undertakes the task of parsing packet traces and extracting rudimentary feature sets from these traces. Depending on specific cases, these raw features can either serve as feature sets themselves or necessitate further transformation into enriched feature sets. The workflow encompassing this packet processing phase leverages straightforward tools such as pcap splitter, nPrint, and tshark. In alignment with our design goals, our framework adopts a flow-level decomposition approach during packet processing. This decomposition facilitates multi-threading and multiprocessing, thereby enhancing the framework's efficiency and expediting its operation. The process entails dividing extensive pcap files into numerous smaller flow-level pcaps, a procedure facilitated by the pcap splitter tool.

Figure 4.2: The packet processing module. The flow level decomposition is taken care of in this section utilizing the Pcap splitter, and feature vectors are extracted using tools like nPrint and tshark.

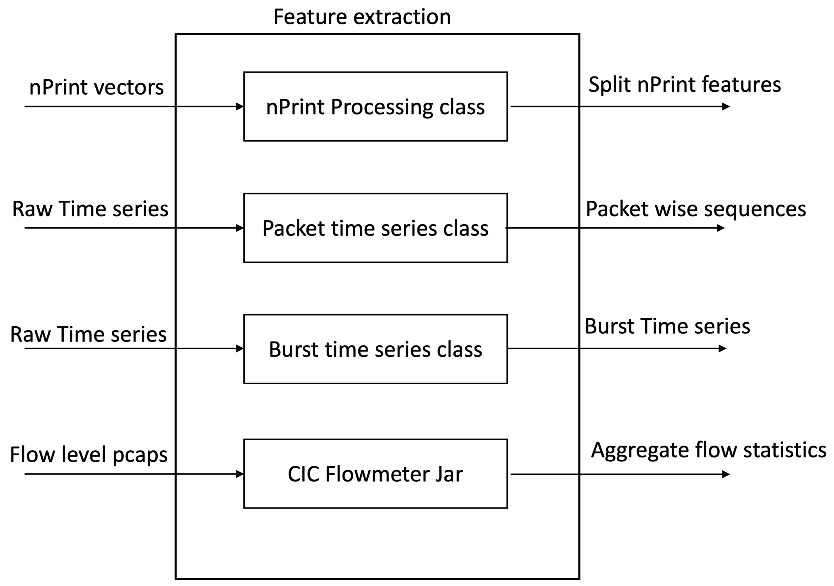


Subsequent to the flow-level decomposition of pcap files, the nPrint and tshark tools come into play, extracting raw nPrint and time series vectors. The integration of the nPrint tool into our framework was accomplished seamlessly, incorporating the necessary commands for efficient extraction of nPrint vectors. This integration was facilitated through an intuitive interface that empowers users to select their preferred vector capture method, offering options such as different flags, payload bytes, and more. Similarly, for the tshark scripts, the commands essential for extracting timestamps, flow IDs, and packet sizes have been incorporated into the scripts. These scripts have the capability to process multiple files concurrently, resulting in the generation of processed raw feature sets. These raw vectors subsequently serve as the foundational materials for the ensuing feature extraction phase. Remarkably swift, the packet processing stage efficiently handles processing for thousands of files within seconds to minutes, a notable achievement attributed to the multi-threading and multiprocessing capabilities embedded in the framework.

4.3 Feature Extraction

Positioned as the second segment in the framework's pipeline, this phase encompasses diverse classes dedicated to processing distinct feature spaces. Within PERRY, the feature extraction module has been extensively crafted, harnessing the capabilities of the advanced analytics tool, PySpark. This tool provides a plethora of optimizations that significantly accelerate the data processing stride.

Figure 4.3: The Feature extraction module supports various data processing tasks.



Within the nPrint processing class, the nPrint vectors undergo transformation to shift from a singular binary vector representation to a more manageable field-wise or byte-wise format. This flexible representation choice facilitates ease of operation compared to the initial single feature vector. Users retain the freedom to selectively opt for specific fields or bytes based on their preferences during feature space selection. The subsequent step involves the horizontal grouping of different bits from the vector representation into distinct fields or bytes, facilitated by the utilization of the PySpark tool. Users are provided the flexibility to select between field-wise and byte-wise formats, which triggers the nPrint class to initiate grouping based on the chosen format. This grouping operation results in the creation of separate columns in the dataframe for each field/byte, including the payload. Furthermore, the processing of the 5-tuple flow ID is conducted, effectively preparing the entire dataframe for writing into the file system.

In the packet time series processing class, raw time series features are harnessed to generate refined packet-level time series attributes, encompassing packet size, timestamps, packet direction, and the 5-tuple flow identifier. A parallel workflow is observed in the burst-wise processing class, with a focus on burst-level features. Here, the raw time series attributes are initially segregated into burst-level attributes, subsequently subject to further processing, leading to diverse fields such as packet count, byte count, packets per second, bytes per second, average packet length, and inter-arrival time between bursts. The processing of packet-wise time series features is executed by first handling the inter-arrival time through the application of the window functionality in PySpark. Subsequently, the direction is determined using a lambda function instead of user-defined functions (UDFs) to enhance processing efficiency significantly. The resulting processed dataframe is then primed for writing into the file system. On the other hand, the processing of burst-wise time series features entails vertical grouping of packets within the dataframe based on the flow ID. These packets are divided into bursts by considering inter-arrival times falling within specified time interval thresholds. Following the segmentation into bursts, window functions are harnessed for performing aggregate operations such as mean, count, lag, sum, etc., thereby generating attributes like average packet length, packets per second, and bursts per second for each burst. Once the processing concludes, the dataframe is prepared for storage in the file system.

Incorporated within the feature extraction phase is the CICFlowmeter jar file, which can be readily invoked to generate aggregate flow-based statistical features. The provided jar file takes flow-level pcap files as input and produces corresponding aggregate flow statistical features in the form of CSV files. Our interface streamlines the process by automatically supplying the necessary pcap files to the command that invokes the jar file. This action results in the generation of these features, which are subsequently stored in our pre-established flow-level directory structure, all without requiring any intervention from the user.

All extracted feature files, alongside their raw counterparts, are systematically stored within the flow-wise directory structure, a strategic arrangement designed for efficient retrieval and grouping purposes.

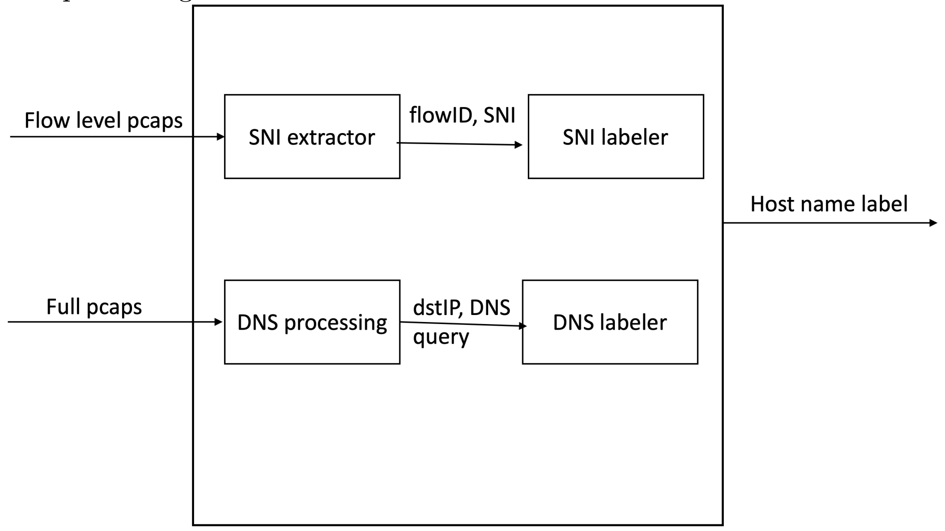
This phase shoulders the heft of processing within the framework, and consequently, it constitutes a significant portion of the overall processing time. To ensure scalability and efficiency, optimizations play a pivotal role. The forthcoming chapter will delve into a comprehensive exploration of these optimizations, accompanied by their associated results.

4.4 Labeling

PERRY offers users the autonomy to opt for labeling or abstain from it. When users choose to label their dataset, the corresponding labeling section activates, initiating the extraction of hostnames using two distinct components: the Server Name Indication

(SNI) extractor and the Domain Name System (DNS) query name resolver.

Figure 4.4: The labeling module along with two of its components - SNI extractor and DNS processing class.



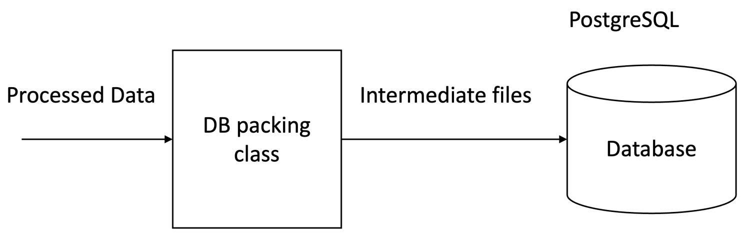
Within this section, the integration of tshark scripts enables the extraction of SNI values for each flow as the flow-wise directory structure is traversed. These extracted SNI values are then employed to generate separate label tags within the respective flow directories. However, the challenge with SNI-based labeling pertains to its limited presence, typically encompassing only 5-10% of the captured traffic's flows. Consequently, the need arises for a more dependable approach to labeling flows devoid of SNI values.

Hence, our framework incorporates an alternative approach by leveraging DNS queries to execute the labeling task. This involves the extraction of all DNS query values and their corresponding resolved addresses, subsequently compiled into a global table. This table then serves as the basis for mapping addresses to hostnames, ensuring a more reliable and comprehensive labeling solution for both SNI-present and SNI-absent flows.

4.5 Storage

Our framework's completeness extends beyond curated datasets, encompassing a systematic approach to data storage and access that enhances efficiency, robustness, and reliability. Recognizing the need for structured data management, we introduced a database system to facilitate seamless querying, filtering, and manipulation of the data with minimal complexity.

Figure 4.5: The storage module showing insertion of data into the database.



In this pursuit, we identified PostgreSQL as the optimal solution, aligning with two crucial considerations. Firstly, its relational database management system (RDBMS) nature simplifies storage and retrieval, especially when managing file locations. This contrasts with a NoSQL document database, as most of our processed files are in CSV format, easily transformed into dataframes with file paths. Such an approach proves more efficient than retrieving entire documents from the database. Secondly, PostgreSQL's support for the JSON data type enhances storage in a queryable format, enabling the embedding of filter options for data manipulation. By leveraging cutting-edge tools like PostgreSQL, we elevate our pipeline's speed and efficiency, further enriching its capabilities.

After processing all the files, we invoke the DB Packing class in our framework that encapsulates all the data into an intermediate CSV file. For this, all the file path information and feature information are inserted into a CSV file progressively and parallely, and the burst-based feature set is wrapped into a JSON type and placed into the CSV file in a type-preserving format without violating the CSV data requirements. Once all the flows have been processed and file paths captured into the intermediate file, the DB Main class is invoked. In this stage, the script expertly handles schema creation, data insertion from the CSV file into the table, and concludes by displaying the finalized table. Remarkably, this entire process takes no longer than a few minutes, even when dealing with a thousand files.

Following the completion of file processing, our framework invokes the DB Packing class, a pivotal step where all data is encapsulated into an intermediate CSV file. During this process, file path information and feature data are progressively inserted into the CSV file. Simultaneously, burst-based feature sets are encapsulated as JSON objects, preserving data types, and seamlessly incorporated into the CSV file, all while adhering to CSV data standards.

Upon processing all flows and systematically capturing file paths in the intermediate file, the DB Main class is engaged. In this stage, the script expertly handles schema creation, data insertion from the CSV file into the table, and concludes by displaying the finalized table. Remarkably, this entire process takes no longer than a few minutes, even when dealing with a thousand files.

4.6 Querying

Within the querying phase, users can leverage the PostgreSQL interface to interact with the database efficiently. Our framework provides an interface through which users can input queries to the PostgreSQL interface, facilitating streamlined data retrieval. The integration of burst features in JSON format enhances querying efficiency, as PostgreSQL effectively processes JSON data. This design choice results in a more simplified schema, supporting layered querying and affording users greater flexibility in customizing their queries. Furthermore, PostgreSQL offers a superior querying experience, boasting built-in optimizations that elevate the overall experience compared to alternative tools.

4.7 Database schema

We also aim to provide an overview of the database schema that we have integrated into our framework. Within this schema, there are up to 15 fields, each encompassing various data types and containing specific information about the dataset. Our schema consists primarily of identifier fields, file path fields, and feature fields, all of which contribute to simplifying the retrieval process. To facilitate comprehension, we have presented a tabular format below, listing all the fields alongside their corresponding data types. This format should aid in your understanding of the schema's composition.

Table 4.1: Table illustrating the database schema for PERRY

Field Name	**Description**	**Datatype**
UID	Unique FlowID + Timestamp	String
Flow ID	5 tuple flow ID	String
Flow Pcap Path	Path to flow wise pacp file	String
Npt path	Path to flow wise nPrint vector file	String
Npt Processed Path	Path to processed nPrint vectors file	string
Timeseries Path	Path to time series features file	String
Timeseries Processed Path	Path to time series features file	String
Burstwise path	Path to Burst features file	String
CICFlow path	Path to CICFlowmeter features file	String
Label	Label value	string
Flow Duration	Total flow duration in seconds	Float/Double
Flow size	Sum of packet sizes in a flow	Float/Double
Flow packets	Number of packets in a flow	Integer
Flow bytes	Number of bytes in a flow	Integer
Burst features	The JSON object of burst features	JSONB

Chapter 5

Addressing the scalability issues

Our framework design diligently adheres to our established design goals, providing a blueprint for an adaptable, user-friendly framework with an intuitive interface that boasts scalability, efficiency, and robustness. While we stand by these commitments, it's imperative that we identify potential challenges that can arise when designing a scalable framework and proactively devise solutions to tackle these challenges.

The data processing phase, in particular, is renowned for presenting scalability challenges due to handling sizable datasets and the limitations of data processing tools in effectively managing such voluminous data while ensuring fault tolerance. These obstacles can impede framework performance, resulting in extended processing times, frustrating user experiences, and, in many cases, reduced reusability.

To address these concerns, the incorporation of optimizations becomes crucial, eradicating these challenges and furnishing an efficient interface that expedites execution, enhances the user experience, and boosts reusability. In this chapter, we succinctly outline these challenges and elaborate on the optimizations we've harnessed to surmount them, presenting outcomes that demonstrate how PERRY achieves tasks within an accelerated timeframe.

5.1 Challenges to Scalability

A major obstacle for data processing frameworks is effectively managing large datasets within limited time constraints. Successful frameworks address this challenge by identifying the underlying issues responsible for this problem. A substantial hindrance in this regard is the scarcity of resources like computational power and memory, which strongly impact overall performance. Hence, leveraging tools like PySpark and PostgreSQL, known for their outstanding performance even in resource-constrained scenarios, offers a significant advantage.

In numerous cases, modern hardware incorporates multi-core architecture, allowing simultaneous execution of multiple threads. Developing a framework should encompass strategies that maximize the utilization of available resources. Additionally, hindrances can arise from poorly designed data processing methods, such as User Defined Functions (UDFs), which are notorious for their resource-intensive nature and time-consuming processing. An efficient data processing solution should effectively tackle these challenges. Another significant hurdle during data processing involves the necessity for extensive shuffling, which is suboptimal for framework optimization. Therefore, effective strategies must be implemented to address this issue.

In the following section, we elaborate on the optimizations we employed to overcome these challenges and showcase the outcomes that highlight our enhanced performance.

5.2 Addressing These Challenges

Our framework embraces four distinct types of optimizations, each significantly influencing runtime performance. For our experimentation, we employed the UCSB dataset, considering various samples comprising 1000 flows, 10000 flows, and 100000 flows. We meticulously detail each optimization, providing dedicated results for every dataset sample, thereby illustrating how these enhancements influenced runtime for each scenario. The ensuing optimizations are integral to PERRY, enhancing its scalability and efficiency:

1. **Multi-processing:** Leveraging multi-processing functionality to handle multiple files using parallel threads concurrently can profoundly enhance the framework's efficiency. Our approach of decomposing larger pcap files into smaller flow-level files through a flow-level decomposition design facilitates this optimization. This strategy efficiently distributes tasks across multiple threads, ensuring reliable and timely completion. Multi-processing has emerged as a significant stride towards minimizing runtime, resulting in exponential reductions in processing time.

2. **Eliminating the UDFs:** As previously mentioned, the usage of User Defined Functions (UDFs) in data processing can be resource-intensive and time-consuming. In certain cases within PERRY, we encountered instances where UDFs were necessary, such as when extracting the 5-tuple flow ID during nPrint vector processing or calculating packet direction within time series features. To mitigate the UDF-related overhead, we developed alternative methods. For instance, we replaced UDFs with simpler lambda functions for processing direction, incurring no additional costs. Moreover, we explored alternative approaches, like directly processing the flow ID during vector curation. These adjustments led to a 50% reduction in processing time.

3. **Caching:** The caching mechanism is a crucial optimization inherent to PySpark, utilized in PERRY to enhance performance. In PySpark, when an action is executed on a DataFrame, the operation's result is stored in memory. Subsequent actions on the same DataFrame would require the DataFrame to be recomputed from the beginning, which could be time-consuming, particularly for sizable datasets.

By implementing the ‘cache()‘ function, PERRY instructs PySpark to retain the DataFrame in memory following the initial action. This enables subsequent actions to leverage the cached data rather than recompute the entire DataFrame, resulting in significant acceleration of data processing tasks.

4. **Reducing the Grouping and Reshuffling:** An additional optimization incorporated into our framework involves minimizing grouping and reshuffling operations whenever feasible. Grouping and reshuffling entail reordering the rows within the dataframe based on the specific requirements of the problem. This process can be resource-intensive, especially for larger datasets, as it necessitates altering the entire dataframe’s structure. Even seemingly straightforward select statements can inadvertently trigger reshuffling to some extent. To mitigate these challenges, we emphasize avoiding reshuffling wherever possible and employing functions that mitigate the need for reshuffling. Throughout the framework’s design, we’ve meticulously identified instances where unnecessary grouping and reshuffling could occur, and we’ve taken measures to curtail these operations, thereby contributing to reduced processing times.

The following tables show the time taken for **1000 flows, 10000 flows, and 100000 flows** of the **UCSB data** to be completely processed by PERRY. For context, **All** refers to **processing nPrint vectors, processing packet time series, processing burstwise time series, processing CICFlowmeter features, and labeling.** The experiments labeled **nPrint** refer to just **extracting the nPrint vectors and horizontally grouping them into fields, which is a costly operation.** Similarly, for **timeseries and burstwise,** it means **extracting raw time series features and processing them in a specific format.** The three tables give a breakdown of the estimated time taken to finish these experiments while using and not using any optimizations.

Table 5.1: Table showing results for 1000 flows with and without optimizations.

Optimization	**Time taken for all**	**Time taken by nPrint**	**Time taken by Time-Series**	**Time taken by Burst wise**
No Multi threading	more than 5hours	more than 5 hours	more than 5 hours	more than 5 hours
With Multi threading	1 hr 55 minutes - 2 hrs	1 hr 2 mins	more than 22 mins	more than 32 mins
Removal of UDF	1 hr 4 mins	45 - 55 mins	less than 10 mins	less than 10 mins
Caching	35 mins	$\leq 15mins$	less than 10 mins	less than 10 mins
Reducing grouping and reshuffling	$\leq 30mins$	$\leq 15mins$	less than 10 mins	less than 10 mins

Table 5.2: Table showing results for 10000 flows with and without optimizations.

Optimization	**Time taken for all**	**Time taken by nPrint**	**Time taken by Time-Series**	**Time taken by Burst wise**
No Multi threading	more than a day	more than a day	more than a day	more than a day
With Multi threading	22 hours 45 minutes	12 hour 45 minutes - 13 hours	more than 3 hours	more than 4 hours
Removal of UDF	12 hours	8 hours	less than hour	less than a hour
Caching	$\leq 7 hours$	$\leq 4 hours$	less than hour	less than hour
Reducing grouping and reshuffling	$\leq 6 hours$	$\leq 4 hours$	less than hour	less than hour

Table 5.3: Table showing results for 100000 flows with and without optimizations.

Optimization	**Time taken for all**	**Time taken by nPrint**	**Time taken by Time-Series**	**Time taken by Burst wise**
No Multi threading	more than a week	more than a week	more than a week	more than a week
With Multi threading	3 - 4 days	2 -3 days	$\leq 1 day$	$\leq 2 days$
Removal of UDF	1 - 2 days	22 hours 45 mins	less than 6 hours	less than 6 hours
Caching	less than 1 day	$\leq 12 hours$	$\leq 3 hours$	$\leq 3 hours$
Reducing grouping and reshuffling	less than 1 day	$\leq 12 hours$	$\leq 3 hours$	$\leq 3 hours$

Chapter 6

Conclusion

6.1 Conclusion and further work

Within this thesis, we have addressed the inherent issue of the tight interdependence between existing data processing techniques and the subsequent model training stages within machine learning pipelines. This constrained linkage has been shown to hinder the development process and impede the exploration of innovative research methodologies aimed at crafting superior models. By examining various works in the field, we have gained insights into specific design goals that offer potential solutions to counteract these challenges.

In the subsequent part of this thesis, we delve into two crucial design goals: Flexibility and Scalability. We extensively elaborate on how these goals materialize through our chosen system design. This design incorporates an interactive user interface, affords users unlimited choices in data processing methods throughout the pipeline, and seamlessly integrates state-of-the-art tools. Subsequently, we provide an in-depth walkthrough of our pipeline framework, PERRY, detailing the role of each individual component. This encompassing analysis covers packet processing, feature extraction, labeling, storage, and querying modules. Together, these components not only streamline data processing tasks but also establish a robust storage and querying infrastructure.

In the concluding segment, we delve into the optimizations we implemented to elevate the framework's performance and present corresponding results across various sample sizes. This thesis endeavors to introduce an efficient data processing solution tailored for network data, aiming to effectively address existing challenges. Our adaptable and scalable design positions this framework as a broadly applicable solution within the network community. By streamlining data processing, we anticipate researchers will be empowered to channel their efforts towards innovation, thereby enhancing their overall experience and productivity.

For future development, the framework could explore various avenues for expansion. Currently, we focus on processing fundamental formats of network data. However, there's potential to extend our approach to incorporate intermediary and derived formats, thereby eliminating the necessity for additional processing steps. Additionally, an interesting direction for improvement could involve utilizing robust languages like Rust to build a lightweight and potentially faster framework compared to Python.

Appendix A

Datasets information

A.1 Feature sets of different network data formats

In this chapter, our aim is to provide a comprehensive breakdown of the various feature sets generated by PERRY. As discussed in earlier chapters, we categorize these feature sets into three main categories. Here, we present detailed information about four distinct formats derived from multiple granularities (for timeseries data) presented in tabular form. It's important to note that certain fields, such as the 5-tuple flow ID, remain consistent across all formats. The nPrint vector representation has a total of 42 fields, including fields from IP, TCP, UDP, and ICMP headers. The packet-wise time series has a total of six features, and the burst-wise time series has seven features, as shown in the figure. The CICFlowmeter features are a collection of 84 features consisting of various aggregate flow statistics.

Table A.1: nPrint vector fields [35].

Begin of Table	
Fields	Number of bits
Source IP	32
Destination IP	32
TCP Source Port	16
TCP Dest Port	16
UDP Source Port	16
UDP Dest Port	16
IP Proto	8
IP ver	4
IP hl	4
IP tos	8
IP tl	16
IP id	16
IP rbit	1
IP dfbit	1
IP mfbit	1
IP foff	13
IP ttl	8
IP cksum	16
IP opt	320
TCP seq	32
TCP ackn	32

Continuation of Table A.1	
Fields	Number of bits
TCP doff	4
TCP res	3
TCP ns	1
TCP cwr	1
TCP ece	1
TCP urg	1
TCP ackf	1
TCP psh	1
TCP rst	1
TCP syn	1
TCP fin	1
TCP wsize	16
TCP cksum	16
TCP urp	16
TCP opt	320
UDP len	16
UDP cksum	16
ICMP type	8
ICMP code	8
ICMP cksum	16
ICMP roh	32
End of Table	

Figure A.1: Packet wise timeseries features

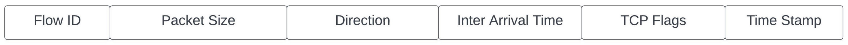


Figure A.2: Burst wise timeseries features

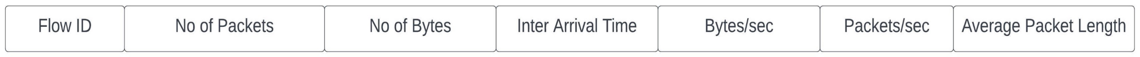


Table A.2: CICFlowmeter feature space fields [10].

Begin of Table	
Field	Description
Flow duration	Duration of the flow in Microsecond
total Fwd Packets	Total packets in the forward direction
total Bwd packets	Total packets in the backward direction
total Length of Fwd Packet	Total size of packet in forward direction
total Length of Bwd Packet	Total size of packet in backward direction
Fwd Packet Length Min	Minimum size of packet in forward direction
Fwd Packet Length Max	Maximum size of packet in forward direction
Fwd Packet Length Mean	Mean size of packet in forward direction
Fwd Packet Length Std	Std deviation size of packet in forward direction
Bwd Packet Length Min	Minimum size of packet in backward direction
Bwd Packet Length Max	Maximum size of packet in backward direction
Bwd Packet Length Mean	Mean size of packet in backward direction
Bwd Packet Length Std	Std deviation size of packet in backward direction
Flow Bytes/s	Number of flow bytes per second
Flow Packets/s	Number of flow packets per second

Continuation of Table A.2

Field	Description
Flow IAT Mean	Mean time btw 2 packets sent in the flow
Flow IAT Std	Std deviation time btw 2 packets sent in the flow
Flow IAT Max	Max time btw 2 packets sent in the flow
Flow IAT Min	Min time btw 2 packets sent in the flow
Fwd IAT Min	Min time btw 2 packets sent in the fwd dir
Fwd IAT Max	Max time btw 2 packets sent in the fwd dir
Fwd IAT Mean	Mean time btw 2 packets sent in the fwd dir
Fwd IAT Std	Std deviation time btw 2 packets sent in the fwd dir
Fwd IAT Total	Total time btw 2 packets sent in the fwd dir
Bwd IAT Min	Min time btw 2 packets sent in the bwd dir
Bwd IAT Max	Max time between 2 packets sent in the bwd dir
Bwd IAT Mean	Mean time btw 2 packets sent in the bwd dir
Bwd IAT Std	Std deviation time btw 2 packets sent in the bwd dir
Bwd IAT Total	Total time btw 2 packets sent in the bwd dir
Fwd PSH flags	No of times the PSH flag set in fwd packets
Bwd PSH Flags	No of times the PSH flag set in bwd packets
Fwd URG Flags	No of times the URG flag set in fwd packets
Bwd URG Flags	No of times the URG flag set in bwd packets
Fwd Header Length	Total bytes used for headers in the fwd dir
Bwd Header Length	Total bytes used for headers in the bwd dir
FWD Packets/s	Number of forward packets per second
Bwd Packets/s	Number of backward packets per second
Packet Length Min	Minimum length of a packet

Continuation of Table A.2	
Field	Description
Packet Length Max	Maximum length of a packet
Packet Length Mean	Mean length of a packet
Packet Length Std	Standard deviation length of a packet
Packet Length Variance	Variance length of a packet
FIN Flag Count	Number of packets with FIN
SYN Flag Count	Number of packets with SYN
RST Flag Count	Number of packets with RST
PSH Flag Count	Number of packets with PUSH
ACK Flag Count	Number of packets with ACK
URG Flag Count	Number of packets with URG
CWR Flag Count	Number of packets with CWR
ECE Flag Count	Number of packets with ECE
down/Up Ratio	Download and upload ratio
Average Packet Size	Mean size of packet
Fwd Segment Size Avg	Mean size observed in the fwd dir
Bwd Segment Size Avg	Mean size observed in the bwd dir
Fwd Bytes/Bulk Avg	Mean bytes bulk rate in the fwd dir
Fwd Packet/Bulk Avg	Mean packets bulk rate in the fwd dir
Fwd Bulk Rate Avg	Mean bulk rate in the fwd dir
Bwd Bytes/Bulk Avg	Mean bytes bulk rate in the bwd dir
Bwd Packet/Bulk Avg	Mean packets bulk rate in the bwd dir
Bwd Bulk Rate Avg	Mean bulk rate in the bwd dir
Subflow Fwd Packets	The mean packets in a sub flow in the fwd

Continuation of Table A.2

Field	Description
Subflow Fwd Bytes	The mean bytes in a sub flow in the fwd
Subflow Bwd Packets	The mean packets in a sub flow in the bwd
Subflow Bwd Bytes	The mean bytes in a sub flow in the bwd
Fwd Init Win bytes	The total bytes sent in initial window in the fwd
Bwd Init Win bytes	The total bytes sent in initial window in the bwd
Fwd Act Data Pkts	No of packets with at least byte of TCP payload
Fwd Seg Size Min	Min segment size observed in the fwd
Active Min	Min time a flow was active before becoming idle
Active Mean	Mean time a flow was active before becoming idle
Active Max	Max time a flow was active before becoming idle
Active Std	Std deviation time a flow was active before idle
Idle Min	Min time a flow was idle before becoming active
Idle Mean	Mean time a flow was idle before becoming active
Idle Max	Maximum time a flow was idle before becoming active
Idle Std	Std deviation time a flow was idle before active

End of Table

Bibliography

[1] “Netmate.” `http://sourceforge.net/projects/netmate-meter/`.

[2] N. Williams, S. Zander, and G. Armitage, *A preliminary performance comparison of five machine learning algorithms for practical ip traffic flow classification*, *SIGCOMM Comput. Commun. Rev.* **36** (oct, 2006) 5–16.

[3] R. Alshammari and A. Nur Zincir-Heywood, *A flow based approach for ssh traffic detection*, in *2007 IEEE International Conference on Systems, Man and Cybernetics*, pp. 296–301, 2007.

[4] R. Alshammari and A. Zincir-Heywood, *Machine learning based encrypted traffic classification: Identifying ssh and skype*, *Computational Intelligence for Security and Defense Applications, 2009. CISDA 2009. IEEE Symposium on* (01, 2009) 1–8.

[5] A. W. Moore and D. Zuev, *Internet traffic classification using bayesian analysis techniques*, in *Proceedings of the 2005 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS ’05, (New York, NY, USA), p. 50–60, Association for Computing Machinery, 2005.

[6] A. Habibi Lashkari., G. Draper Gil., M. S. I. Mamun., and A. A. Ghorbani., *Characterization of tor traffic using time based features*, in *Proceedings of the 3rd International Conference on Information Systems Security and Privacy - ICISSP*, pp. 253–262, INSTICC, SciTePress, 2017.

[7] E. Glatz and X. Dimitropoulos, *Classifying internet one-way traffic*, IMC ’12, (New York, NY, USA), p. 37–50, Association for Computing Machinery, 2012.

[8] s. Ndichu, S. Okoth, H. Okoyo, and C. Wekesa, *Detecting remote access network attacks using supervised machine learning methods*, *International Journal of Computer Network and Information Security* **15** (04, 2023) 48–61.

[9] A. Habibi Lashkari, G. Draper Gil, M. Mamun, and A. Ghorbani, *Characterization of encrypted and vpn traffic using time-related features*, 02, 2016.

[10] “Cicflowmeter.” `https://github.com/ahlashkari/CICFlowMeter`.

[11] C. M. Inacio and B. Trammell, *Yaf: Yet another flowmeter*, in *Proceedings of the 24th International Conference on Large Installation System Administration*, LISA'10, (USA), p. 1–16, USENIX Association, 2010.

[12] K. O'Shea and R. Nash, *An introduction to convolutional neural networks*, 2015.

[13] S. Hochreiter and J. Schmidhuber, *Long short-term memory*, *Neural Comput.* **9** (nov, 1997) 1735–1780.

[14] D. Bank, N. Koenigstein, and R. Giryes, *Autoencoders*, 2021.

[15] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, *Attention is all you need*, 2023.

[16] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, *Empirical evaluation of gated recurrent neural networks on sequence modeling*, 2014.

[17] Y. Zeng, Z. Qi, W. Chen, and Y. Huang, *Test: an end-to-end network traffic classification system with spatio-temporal features extraction*, in *2019 IEEE International Conference on Smart Cloud (SmartCloud)*, pp. 131–136, 2019.

[18] G. Zhao, Z. Wang, and Z. Yang, *An encrypted traffic classification model based on the raw traffic and spatiotemporal characteristics*, EITCE '22, (New York, NY, USA), p. 1208–1213, Association for Computing Machinery, 2023.

[19] L. QOSIENT, "Argus:auditingnetworkactivity." `http://www.qosient.com/argus.`

[20] L. DERI, "nprobe–netflow/ipfixnetworkprobe." `http://www.ntop.org/nProbe.html,Oct2006.`, 2006.

[21] Y. Pan, X. Zhang, H. Jiang, and C. Li, *A network traffic classification method based on graph convolution and lstm*, *IEEE Access* **9** (2021) 158261–158272.

[22] M. S. Towhid and N. Shahriar, *Encrypted network traffic classification using self-supervised learning*, in *2022 IEEE 8th International Conference on Network Softwarization (NetSoft)*, pp. 366–374, 2022.

[23] Z. Shi, N. Luktarhan, Y. Song, and H. Yin, *Tsfn: A novel malicious traffic classification method using bert and lstm*, *Entropy* **25** (05, 2023) 821.

[24] A. Dietmüller, S. Ray, R. Jacob, and L. Vanbever, *A new hope for network model generalization*, in *Proceedings of the 21st ACM Workshop on Hot Topics in Networks*, HotNets '22, (New York, NY, USA), p. 152–159, Association for Computing Machinery, 2022.

[25] L. Vu, H. V. Thuy, Q. U. Nguyen, T. N. Ngoc, D. N. Nguyen, D. T. Hoang, and E. Dutkiewicz, *Time series analysis for encrypted traffic classification: A deep learning approach*, in *2018 18th International Symposium on Communications and Information Technologies (ISCIT)*, pp. 121–126, 2018.

[26] J. Koumar, K. Hynek, and T. Čejka, *Network traffic classification based on single flow time series analysis*, 2023.

[27] Z. Zou, J. Ge, H. Zheng, Y. Wu, C. Han, and Z. Yao, *Encrypted traffic classification with a convolutional long short-term memory neural network*, in *2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pp. 329–334, 2018.

[28] I. Akbari, M. A. Salahuddin, L. Ven, N. Limam, R. Boutaba, B. Mathieu, S. Moteau, and S. Tuffin, *A look behind the curtain: Traffic classification in an increasingly encrypted web*, *Proc. ACM Meas. Anal. Comput. Syst.* **5** (feb, 2021).

[29] R. Schuster, V. Shmatikov, and E. Tromer, *Beauty and the burst: Remote identification of encrypted video streams*, in *26th USENIX Security Symposium (USENIX Security 17)*, (Vancouver, BC), pp. 1357–1374, USENIX Association, Aug., 2017.

[30] H. Singh, *Performance analysis of unsupervised machine learning techniques for network traffic classification*, in *2015 Fifth International Conference on Advanced Computing Communication Technologies*, pp. 401–404, 2015.

[31] T. Auld, A. W. Moore, and S. F. Gull, *Bayesian neural networks for internet traffic classification*, *IEEE Transactions on Neural Networks* **18** (2007), no. 1 223–239.

[32] M. Lotfollahi, M. Jafari Siavoshani, R. Shirali Hossein Zade, and M. Saberian, *Deep packet: a novel approach for encrypted traffic classification using deep learning*, *Soft Computing* **24** (Feb, 2020) 1999–2012.

[33] R. Yuan, Z. Li, X. Guan, and L. Xu, *An svm-based machine learning method for accurate internet traffic classification*, *Information Systems Frontiers* **12** (Apr, 2010) 149–156.

[34] F. Dehghani, N. Movahhedinia, M. R. Khayyambashi, and S. Kianian, *Real-time traffic classification based on statistical and payload content features*, in *2010 2nd International Workshop on Intelligent Systems and Applications*, pp. 1–4, 2010.

[35] J. Holland, P. Schmitt, N. Feamster, and P. Mittal ACM, nov, 2021.

[36] A. Este, F. Gringoli, and L. Salgarelli, *Support vector machines for tcp traffic classification*, *Computer Networks* **53** (2009), no. 14 2476–2490.

[37] T. Shapira and Y. Shavitt, *Flowpic: A generic representation for encrypted traffic classification and applications identification*, *IEEE Transactions on Network and Service Management* **18** (2021), no. 2 1218–1232.

[38] R. Li, X. Xiao, S. Ni, H. Zheng, and S. Xia, *Byte segment neural network for network traffic classification*, in *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*, pp. 1–10, 2018.

[39] S. Ahn, J. Kim, S. Y. Park, and S. Cho, *Explaining deep learning-based traffic classification using a genetic algorithm*, *IEEE Access* **9** (2021) 4738–4751.