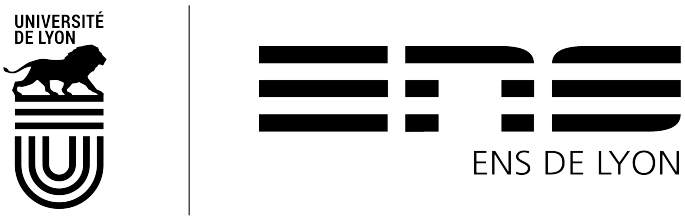
Bridging the Gap Between Machine Learning and Networked Systems

Francesco Bronzino



Mémoire d'Habilitation à Diriger des Recherches

présenté le 30 Septembre 2024

à l'École Normale Supérieure de Lyon

par

Francesco Bronzino

Bridging the Gap Between Machine Learning and Networked Systems

Devant le jury composé de :

Rapporteurs :	Isabelle Chrisment	-	Université de Lorraine
	Mark Crovella	-	Boston University
	Marco Fiore	-	IMDEA Networks
Présidente :	Isabelle Guérin-Lassous	-	Université Claude Bernard Lyon 1
Examinateurs :	Marcelo Dias de Amorim	-	CNRS
	Andrzej Duda	-	INP Grenoble

Abstract

Applications of machine learning to networking, from performance diagnosis to security, have conventionally relied on models that are trained on offline packet traces, without regard to neither the cost of gathering, computing, and storing the corresponding input features nor the performance of the model in its final deployment environment. As a result, there remains a significant gap between the development of statistical models for network operations and their application and systemization in practice. From data collection to feature engineering, from model training to deployment, we argue that all steps of the machine learning development pipeline contain challenges that can hinder the effectiveness of the final model, beyond its accuracy at testing time. The goal of our research is to build new foundational building blocks necessary to address the challenges that emerge when applying machine learning on network traffic deployed in operational networks.

In this manuscript, we present our initial work towards the development of new techniques that make it easier and more effective to develop models that work in real-world network deployments. As a first step, we develop new models to infer quality metrics (*i.e.*, startup delay and resolution) for encrypted streaming video services. We demonstrate the models are practical through a 16-month deployment in 66 homes and provide new insights about the relationships between Internet "speed" and the quality of the corresponding video streams, for a variety of services. Building on the lessons learned from this experience, we build solutions that address various challenges across the model development pipeline: (1) To address the lack of labeled data to train models on, we develop a new technique to generate synthetic traffic for training models. (2) To address the need of designing models that account for both predictive performance and systems costs, we develop new techniques to explore cost-aware data representations and automatically train models. (3) To address the challenge of maintaining the accuracy of models in the face of changing data distributions, we develop a new methodology to mitigate model performance decay over time. Overall, our work demonstrates the benefits of systematically addressing the challenges that arise when applying machine learning to network traffic, reducing the gap that limits network operators from deploying models into production.

Contents

Chapter 1

Introduction

Takeaways

This manuscript summarizes seven years worth of research work initiated while working as a postdoctoral fellow at Inria Paris. During this time window, my work has focused on the development of machine learning-based solutions for network management operations. Our work tackles the challenges that arise when deploying machine learning models in operational networks, from developing new models amenable to modern networks to designing new systems that better support these models. Importantly, our work highlights the need of developing practical solutions at the intersection of networking and machine learning that can be deployed in real-world settings. The goal of this manuscript is to provide a narrative structure to our contributions and delineate open challenges that need to be addressed in future research.

Contents

1.1 ML-Based Network Systems: from Data to Deployment

Network management tasks commonly rely on the ability to classify traffic by type or identify important events of interest from measured network traffic. Over the past 15 years, machine learning (ML) models have become increasingly integral to these tasks [97, 126, 23]. Thanks to the commoditization of advanced data driven technologies, *e.g.*, neural networks, complex tasks such as detecting network failures or extracting information from encrypted traffic streams are now possible. For a variety of different tasks, machine learning models have grown to outperform traditional rule-based heuristics for a variety of traffic analysis applications, from traffic classification [64, 82], intrusion detection [147], and QoE inference [40, 91]. Unfortunately, developing, deploying, and maintaining these models can prove challenging in practice [129].

Figure 1.1 shows the typical ML model development pipeline, from measurement to modeling: The process begins with data collection or a pre-collected dataset (*e.g.*, a raw traffic trace, summary statistics produced by a measurement system). Processed features, also called data *representations*, are then derived from this underlying data (*e.g.*, packet sizes, inter-arrival times, or flow statistics). In the following step, one or more models are trained on these data representations, creating a pool of candidate models to choose from, testing them based on one or more performance metrics (*e.g.*, accuracy, F1 score, precision and recall). Finally, the selected model can be deployed into production. If necessary, due to poor model performance at training time or during the model lifetime, the process might restart, with the model being retrained on new data or with new features. This process is well established and has been successfully employed in a variety of network management tasks, with previous work largely focusing on maximizing a single performance dimension: model accuracy.[1] Unfortunately, the focus on the predictive performance of ML-based solutions has often overshadowed equally critical aspects that arise from deploying these models in production settings.

Consider for example the case of feature selection, a critical aspect of the feature engineering step: Training a machine learning model from network traffic typically involves extracting a set of features that achieve good model performance. This process requires domain knowledge to know the features that are most relevant to prediction, as well as how to transform those features in ways that result in separation of classes in the underlying dataset. Even for cases where the model itself learns the best representation based on its input (*e.g.*, representation learning or deep learning), the designer of the algorithm must still determine the *initial* representation of the data to provide to the model. Yet, for networking tasks, this process might be hampered by the data that is made initially available to design the model, as with existing network traffic measurement systems, the first three steps of this process—collection, cleaning, and feature engineering—are often out of the pipeline designer's control. To date, most network management tasks that rely on machine learning from network traffic have assumed the data to be fixed or given, typically because decisions about measuring, sampling, aggregating, and storing network traffic data are made based on the capabilities (and constraints) of current standards and hardware capabilities (*e.g.*, IPFIX/NetFlow). As a result, a model might be trained with a sampled packet trace or aggregate statistics that describe network traffic—not necessarily because that data representation would result in an efficient model with good overall performance, but rather because the decision about data collection was made well before any modeling

[1] In this document, we use at times the word accuracy as an umbrella term referring to all possible metrics used to determine a model's predictive performance.

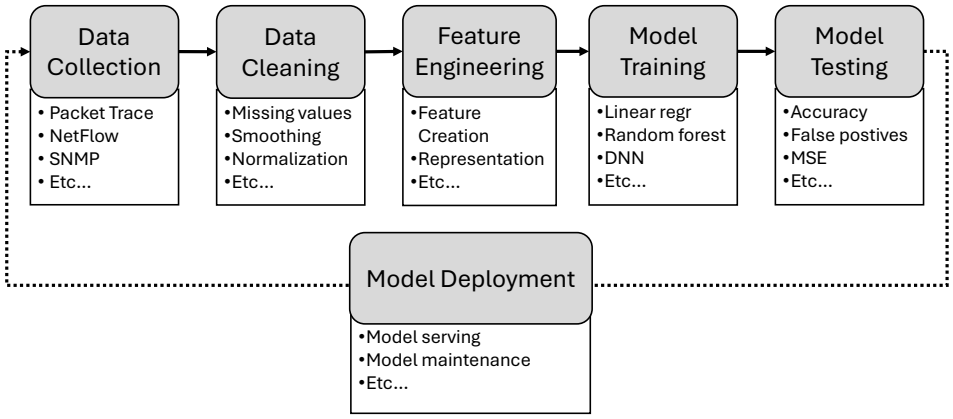


Figure 1.1: Typical pipeline for model design and deployment in network inference.

or prediction problems were considered.

On the surface, the simplest and most effective solution to address these limitations would involve starting the process from raw packets. Any network operator or researcher knows full well that raw packet traces offer maximum flexibility to explore transformations and representations that result in the best model performance. Yet, unfortunately, capturing raw packet traces often proves to be impractical. In large networks, raw packet traces result in massive amounts of data introducing storage and bandwidth requirements that are often prohibitive. Limiting the duration of a pcap collection (*e.g.*, collecting one day's worth of traces) can reduce data storage requirements, but might negatively affect the accuracy of the produced models as the limited capture may not represent network conditions at other times. Conversely, pcaps collected in a controlled laboratory environment might produce models not directly applicable in practice because operational networks include other traffic characteristics that are hard to capture in a lab environment. Due to these reasons, experiments (and much past work) that demonstrate a model's accuracy turn out to be non-viable in practice because the systems costs of deploying and maintaining the model are prohibitive [24].

1.2 The Technical Debt of ML-Based Network Systems

The central premise of this work is motivated by the need for additional awareness in the development and deployment of machine learning models for network management tasks and address the challenges that arise when deploying the models in practice, *i.e.*, *bridging the gap that separates us from the ability to deploy machine learning models that work in operational networking contexts*. To better understand what type of solutions we seek to develop, let's go back to the example of feature selection, particularly focusing on a specific use case: the inference of video streaming quality from encrypted network traffic. As web content has become ubiquitously encrypted, operators are prevented from directly inspecting video streams to troubleshoot performance problems. Instead, operators now rely on statistical learning to match traffic characteristics to the application quality they aim to collect (*e.g.*, resolution).

In general, different network inference tasks use different models, each of which may depend on a unique set of features. For the case of the inference of quality metrics of a video streaming application from encrypted traffic, the task has been commonly modeled

Feature Set	Accuracy
Model 1	0.72
Model 2	0.83
Model 3	0.84

Table 1.1: Models comparison based solely on accuracy

using data representations that are calculated from different networking layers at regular intervals [25, 91, 90, 52]. For instance, one could group data representations based on the three different networking layers they are collected from: Network, Transport, and Application layers. Network-layer features would then consist of lightweight information available from observing network flows (identified by the IP/port four-tuple) and are typically available in monitoring systems (*e.g.*, NetFlow) [7, 6]. Transport-layer features would consist of information extracted from the TCP header, such as end-to-end latency and packet retransmissions. Such features are widely used across the networking space but can require significant resources (*e.g.*, memory) to collect from large network links and are not commonly available across all measurement vantage points. Finally, application-layer metrics are those that include any feature related to the application data that can be gathered by solely observing packet patterns (*i.e.*, without resorting to deep packet inspection). These features capture a unique behavior of the application and have to be designed specifically for every problem in consideration. This uniqueness makes them harder to discover as, to evaluate their effectiveness, a network designer has to build dedicated collection tools that would not be normally deployed anywhere in the network.

Altogether these features compose the set of candidate representations that the model designer would pick from to design the inference model. In particular, in most cases, the model designer would pick different subsets of these features to train multiple models, and evaluate the most effective solution to its problem. Let's consider a simple scenario where the designer has selected three groups of features to train three different models, each of which is evaluated based on its accuracy. Table 1.1 shows the accuracy of each model.

At first glance, Model 3 would seem to be the best candidate for deployment, as it achieves the highest accuracy, even if by a small margin. Unfortunately, this result does not tell the full story of how the model might perform upon its deployment. In practice, the selected model would have to be deployed in a network, where various considerations might impact its final performance. To illustrate this problem, let's now consider what happens when we add a second dimension to the evaluation of the models. Figure 1.2 shows the relationship between model accuracy and the state cost incurred by a measurement system at collection time, *i.e.*, the amount of memory required to store the features used to produce the model's input representations. The figure shows that while Model 3 does achieve the highest accuracy, it does so by requiring two orders of magnitude more state than the second best performing model, *i.e.*, Model 2. This result suggests that collection costs (*e.g.*, required memory) could be significantly reduced with limited impact on model performance, providing important opportunities for in-network reduction and aggregation.

This simple example ultimately shows how the design and evaluation of machine learning models for network management tasks must also consider a variety of operational costs that occur when deploying a model *in practice*. Sculley *et al.* refer to these considerations as "technical debt" [116] and identified a number of hidden costs that contribute to building the technical debt of ML-systems, such as: unstable sources of data, underutilized data, use of generic packages, among others. This problem is vast and complex, and covering

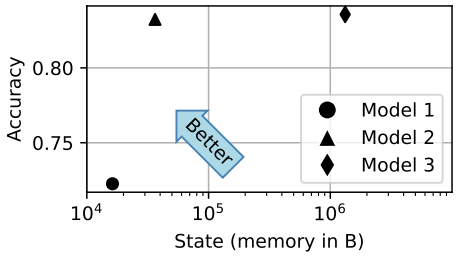


Figure 1.2: The relationship between features state cost and model performance for video streaming resolution inference. Maximizing model performance requires two orders of magnitude more state.

all dimensions of this problem would require a number of contributions that would extend well beyond the size of a single manuscript. For example, we do not investigate practical considerations such as model training time, the energy cost of training, model size, and many other practical considerations. Still, our work is one of the first to systematically explore the challenges and tradeoffs that emerge when applying machine learning on network traffic in real-world environments, which we believe deserves more consideration before machine learning can be more widely adopted in operational networks.

1.3 Cost and Deployment Aware ML for Network Management Tasks

The previous Section introduced the concept of technical debt in the context of machine learning models for network management tasks. However, addressing the challenges that arise due to the technical debt requires answering a number of questions that span the entire machine learning development pipeline. Some of these questions are: How to compensate for the lack of training labeled data? How to systematically develop a methodology that enables the exploration of effective features for a given task? How to automatically navigate the exponential number of combinations possible to create an "optimal" model? How to maintain the model during its lifetime, ensuring that it remains effective as the network changes?

This work aims to address these questions by developing new building blocks tailored at making the process of designing machine learning models that work in real-world deployments simpler and more effective. The rest of this manuscript presents our contributions spanning across all the steps of the machine learning pipeline. First, to introduce our approach, Chapter 2 discusses our work on developing machine learning models for a key inference problem: the development of models that infer quality metrics (*i.e.*, startup delay and resolution) for encrypted streaming video services. The Chapter presents and discusses how our work (my first in the context of ML applied to networking) took a step forward towards making video inference models practical, tackling the challenges that arise when the models must operate on real network traffic traces and across a broad range of services. Beyond contributing the developed models, this work demonstrated the need of designing inference models that can be used in practice, and served as as a learning experience, opening the path to a variety of contributions across the model development pipeline. Starting from this first experience, the rest of the manuscript presents, in the order of the pipeline

shown in Figure 1.1,[2] the rest of our contributions toward the development of practical models that work in realistic network settings:

- **Data Generation (Chapter 3).** Datasets of labeled network traces are essential to develop effective and accurate models for networking tasks. Yet, dataset availability is scarce, due to privacy concerns, outdated data, and more. To overcome this limitation, we present `NetDiffusion`, a tool that uses a finely-tuned, controlled variant of a Stable Diffusion model to generate synthetic network traffic data that is high fidelity and conforms to protocol specifications. The generated synthetic traces are compatible with common network analysis tools and support a myriad of network tasks, suggesting that `NetDiffusion` can serve a broader spectrum of network analysis and testing tasks, extending beyond ML-centric applications.

- **Feature Engineering and Model Training (Chapter 4).** During this introduction, we discussed how the features that a model relies on ultimately determine the model accuracy, as well as where and whether the model can be deployed in practice. In this chapter, we present two solutions that aim to improve the feature engineering process. First, we present `Traffic Refinery`, a new framework and system that enables the joint evaluation of both the conventional notions of machine learning performance (*e.g.*, model accuracy) and the systems-level costs of different representations of network traffic. `Traffic Refinery` both highlights this design space and makes it possible to explore different representations for learning tasks. Second, we develop `CATO`, a framework that addresses the problem of jointly optimizing the predictive performance of models and the associated systems costs of the model serving pipeline.

- **Model Deployment (Chapter 5).** Developing effective ML-based solutions for network traffic does not end with an accurate trained model. During the deployment lifetime of the model, its accuracy can degrade due to *concept drift*, where either the relationships between features and the target to be predicted, or the features themselves change, rendering the model ineffective. In this chapter, we present Local Error Approximation of Features (`LEAF`), a new methodology for concept drift mitigation. `LEAF` works by detecting drift; explaining the features and time intervals that contribute the most to drift; and mitigating it using forgetting and over-sampling. `LEAF`'s approach consistently outperforms state-of-the-art solutions, improving model performance over time, as well as reducing the need of costly retraining operations.

Chapter 6 concludes the manuscript with a discussion on our plans to explore two new research directions within this space: (1) Attack the challenges that limit the deployment of Machine Learning network operations at scale, *i.e.*, over an entire network; and (2) Explore the role that novel generative AI models will have in enhancing networked systems and inference models.

[2] Rather than the chronological order of the contributions.

Chapter 2

Inferring Streaming Video Quality from Encrypted Traffic

i.e., Where It All Started

"Inferring Streaming Video Quality from Encrypted Traffic: Practical Models and Deployment Experience", ACM Sigmetrics 2020 [25]

Takeaways

Inferring the quality of streaming video applications is important for Internet service providers, but the fact that most video streams are encrypted makes it difficult to do so. We develop models that infer quality metrics (*i.e.*, startup delay and resolution) for encrypted streaming video services. Our work builds on previous state-of-the-art, but extends it in several ways. First, our models function in deployment settings where the video session traffic must be identified from all traffic traversing the observed network link and the time precision of the collected statistics is more coarse (*e.g.*, due to data aggregation). Second, our models perform predictions at finer granularity (*e.g.*, the precise startup delay instead of just detecting short versus long delays) allowing to draw better conclusions on the ongoing sessions for a large variety of services. We demonstrate that our models are practical through a 16-month deployment in 66 homes and offer new insights about the relationships between Internet "speed" and the quality of the corresponding video streams.
Beyond these contributions, this work demonstrates the need of designing inference models that can be used in practice, opening the path for a variety of advancements in developing practical models that work in realistic network settings.

Contents

2.1 The Need for Video Quality Inference

Video streaming traffic is by far the dominant application traffic on today's Internet comprising more than 65% of all Internet traffic in 2022 [113]. Optimizing video delivery depends on the ability to determine the quality of the video stream that a user receives. In contrast to video content providers, who have direct access to video quality from client software, Internet Service Providers (ISPs) must infer video quality from traffic as it passes through the network. ISPs need to measure video streaming quality because it represents a more direct metric of customer experience than performance metrics typically extracted from network flows such as data rates. ISPs monitor video quality metrics to detect network issues that affect customer experience and mitigate problems before they generate user complaints. On longer timescales, trends in video quality can facilitate capacity planning. Yet, monitoring video quality is not straightforward for ISPs: With end-to-end encryption becoming more common, as a result of increased video streaming content over HTTPS and QUIC [112, 100], ISPs cannot directly observe video quality metrics such as startup delay and video resolution from the video streaming protocol [17, 43]. The end-to-end encryption of the video streams thus presents ISPs with the challenge of inferring video quality metrics solely from properties of the network traffic that are directly observable.

Previous to our work, existing approaches inferred the quality of a specific video service, typically using offline traces generated in controlled laboratory settings and targeting a single service [91, 40, 70] or inferred performance metrics that are easier to collect within the network, but relate poorly to actual application performance (*e.g.*, average flow throughput is a bad indicator of video resolution) [90]. Unfortunately, these models were not directly applicable in practice due to various factors. First, inference models must take into account the fact that real network traffic traces contain a mix of traffic. Often gathered at coarse temporal granularities due to monitoring constraints in production networks, video session traffic is mixed with non-video cross-traffic. In a real deployment, the models must then identify the video sessions accurately, especially given that errors in identifying applications can propagate to the quality of the prediction models. Second, the prediction models should apply to a range of services, which existing models tend not to do. A model that can predict quality across multiple services is hard to devise because both video streaming algorithms and content characteristics can vary significantly across video services (*e.g.*, buffer-based [58] versus throughput-based [132] rate adaptation algorithms, fixed-size [58] versus variable-size video segments [94]).

Our work takes a step towards making video inference models practical, tackling the challenges that arise when the models must operate on real network traffic traces and across a broad range of services. As a proof of concept that models can be designed and implemented, we studied four major streaming services—Netflix, YouTube, Amazon, and Twitch—across a 16-month period, in 66 home networks in the United States and France, comprising a total of 216,173 video sessions. The rest of the chapter details our methodology to develop video streaming quality inference models (Section 2.2); it discusses the challenges and lessons learned from deploying the models (Section 2.3); and, finally, it elaborates on the implications of our findings and how they inspired the rest of the work described in this manuscript (Section 2.4).

2.2 Methodology

In this section, we define the problem of video quality inference from encrypted network traffic and develop machine learning models to solve it. Following the steps of the machine

learning pipeline presented in Chapter 1, we discuss how we collect a dataset to train the models, what input features we use, and finally how we validate the developed models.

2.2.1 Creating a Labeled Dataset

To collect a labeled dataset of video streaming network traces we instrumented 11 machines to generate video traffic and collect packet traces together with data from a Chrome extension extracting ground truth from the video player. We collected data from November 20, 2017 to May 3, 2019 across both residential locations as well as in our controlled lab environment. We collected in total 13,765 video sessions from four of the main video service providers: Netflix, Amazon, Twitch, and YouTube.[1]

Generating and labeling traffic traces. We generated video sessions automatically using `ChromeDriver` [32]. We played each session for 8 to 12 minutes depending on the video length. For longer videos (*e.g.*, Netflix movies), we varied the playback starting point to avoid always capturing the first portion of the video. We generated five categories of sessions: Netflix, Amazon, Twitch, YouTube TCP, and YouTube QUIC. We randomly selected Netflix and Amazon movies from the suggestions presented by each service in the catalog page. To avoid bias, we selected movies from different categories including action, comedies, TV shows, and cartoons. We ultimately selected 25 movies and TV shows used in rotation from Netflix and 15 from Amazon. Similarly for YouTube, we selected 30 videos from different categories. Twitch automatically starts a live video feed when opening the service home page. Thus, we simply collected data from the automatically selected feed. For each of the streamed sessions, we collected packet traces using `tcpdump` [136] on the network interface the client uses and computed the traffic features presented in the next section.

Labeling: Chrome Extension. For each session we also collected ground truth of the video sessions through a custom browser plugin we developed.[2] The plugin collects two pieces of information: (1) To label traffic traces with the appropriate video quality metrics, the plugin monitors application-level information for the four services as seen by the client. We focus on startup delay and resolution as target quality metrics. Prior work has focused on bitrate as a way to approximate the video resolution, but the relationship between bitrate and resolution is complex because the bitrate also depends on the encoding and the content type [10]. Additionally, other metrics such as resolution switches can be inferred later from the resolution per time slot. (2) To record the client interactions with the content servers, the extension collects browsing history by parsing events available from the Chrome WebRequest APIs [31]. This API exposes all necessary information to identify the start and end of video sessions, as well as the HTTPS requests and responses for video segments.

Emulating Diverse Network Conditions. In the lab environment, we manually varied the network conditions in the experiments using `tc` [26] to ensure that our training datasets captured a wide range of network conditions. These conditions can either be stable for the entire video session or vary at random time intervals. We varied capacity from 50 kbps to 30 Mbps, and introduced loss rates between 0% and 1% and additional latency between 0 ms and 30 ms. All experiments within homes ran with no other modifications of network conditions to emulate realistic home network conditions.

Network Layer	Transport Layer	Application Layer
throughput up/down (total, video, non-video)	# flags up/down (ack / syn / rst /push / urgent)	segment sizes (all previous, last-10, cumulative)
throughput down difference	receive window size up/down	segment requests inter arrivals
packet count up/down	idle time up/down	segment completions inter arrivals
byte count up/down	goodput up/down	# of pending request
packet inter arrivals up/down	bytes per packet up/down	# of downloaded segments
# of parallel flows	round trip time	# of requested segments
	bytes in flight up/down	
	# retransmissions up/down	
	# packets out of order up/down	

Table 2.1: Summary of the extracted features from traffic.

2.2.2 Input Features

For each video session, we compute a set of features from the captured traffic at various levels of the network stack, as summarized in Table 2.1. We consider a super-set of the features used in prior models [40, 91, 90] to evaluate the sub-set of input features that provides the best inference accuracy. We categorize the features into three groups: network-layer features, transport-layer features, and application-layer features.

Network Layer. We define network-layer features as metrics that solely rely on information available from observation of a network flow (identified by the IP/port four-tuple), e.g., throughput, packet counts, and byte counts. These features are lightweight to compute and do not require maintaining per-flow state.

Transport Layer. Transport-layer features include information such as end-to-end latency and packet retransmissions. These metrics reveal possible network problems, such as presence of a lossy link in the path or a link causing high round-trip latencies between the client and the server. Unfortunately, transport metrics suffer two shortcomings. First, due to encryption, some metrics are only extractable from TCP flows and not flows that use QUIC. Second, many transport-layer features require maintaining long-running per-flow state, which is prohibitive at scale.

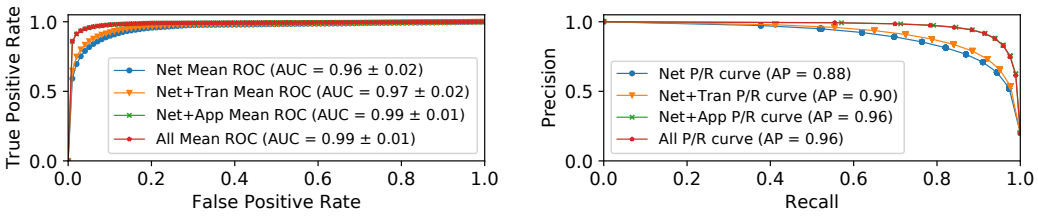
Application Layer. Application-layer metrics include any feature related to the application data. Encryption, however, makes it impossible to directly extract any application-level information from traffic using deep packet inspection. Fortunately, we can still derive some application-level information from encrypted traffic. For example, BUFFEST [70] showed how to identify individual video segments from the video traffic, using the times of upstream requests (*i.e.*, packets with non-zero payload) to break down the stream of downstream packets into video segments. We use sequences of inferred video segment downloads to build up the feature set for the application layer.

2.2.3 Model Validation

We train models considering different sets of input features: network-layer features (*Net*), transport-layer features (*Tran*), application-layer features (*App*), as well as a combination of features from different layers: Net+Tran, Net+App, and all layers combined (*All*). For each target quality metric, we train 32 models in total: (1) varying across these six feature

[1]Dataset available for download at [72].

[2]We make the extension and the tools to generate and label video sessions available as open source software [142].



(a) ROC.

(b) Precision-recall.

Figure 2.1: Resolution inference using different feature sets (For all four video services).

sets and (2) using six different datasets, splitting the dataset with sessions from each of the four video services—Netflix, YouTube, Amazon, and Twitch—plus two combined datasets, one with sessions from all services (which we call *composite*) and one with sessions from three out of four services (which we call *excluded*). For models that rely on transport-layer features, we omit YouTube sessions over UDP as we cannot compute all features. For each target quality metric, we evaluate models using 10-fold cross-validation. For resolution, we trained a classifier with five classes: 240p, 360p, 480p, 720p, and 1080p. We evaluated Adaboost (as in prior work [91]), logistic regression, decision trees, and random forest. We select random forests because it achieves higher precision and recall with lower false positive rates. Similarly for startup delay, we trained on features computed from the first ten seconds of each video session and experimented with different regression methods, including: linear, ridge, SVR, decision tree regressor, and random forest regressor. We evaluate methods based on the average absolute error and conclude that random forest leads to lowest errors.

Main Takeaways. Overall, our results show that models that rely on network- and application-layer features outperform models that rely on network- and transport-layer features across all services. This result is in contrast with prior work [40, 91], which provided models that rely on transport-layer features. For resolution, we observe that the models trained with application layer features consistently achieve the best performance with both precision and recall reaching 91% for a 4% false positive rate (shown in Figure 2.1). Any model not including application features reduces precision by at least 8%, while also doubling the false-positive rate. Similarly for startup delay, our results show that the model using a combination of features from network and application layer yields the highest precision, minimizing the root mean square error (RMSE) across the four services. These results also show that we can exclude Transport-based features, because Net+App models have consistently smaller errors. ISPs may ultimately choose a model that uses only network-layer features, which rely on features that are readily available in many monitoring systems for a small decrease in inference accuracy.

We also evaluate whether our models are general. A composite model—where we train the model with data from multiple services and later predict quality of any video service—is ideal as it removes the requirement to collect data with ground truth for a large number of services. Our evaluation of a *composite model* trained with data from the four video services shows that it performs nearly as well as *specific models* that rely only on sessions from a single service across both quality metrics. This result raises hopes that the composite model can generalize to a wide variety of video streaming services. When we train models using a subset of the services and evaluate it against the left out one (*excluded models*), however, the accuracy of both startup delay and resolution models degrades significantly, rendering the models unusable. This result highlights that although our modeling method is general in that it achieves good accuracy across four video services, the training set used to infer

quality metrics should include all services that one aims to do prediction for.

2.3 Model Deployment

In this section we present the challenges we discovered when applying the methods in practice. We analyze the results collected using the models from a 16-month deployment in 66 homes in the United States and France. We first characterize the deployment and the collected data and then present how we tackle the two challenges. Finally, we present the results obtained from the long term study of video quality in the wild.

2.3.1 Deployment Characterization.

To study the video quality experienced by users in a real-world setting, we have developed a network monitoring system that collects the features described in the previous section. The system collects network and application features aggregated across five-second intervals. For each interval, it reports average statistics for network features divided per flow, together with the list of downloaded video segments.

We use the system to collect data from 66 homes in the United States and France between January 23, 2018, and March 12, 2019. We concluded the data collection in May 2019, at which point we had 60 devices in homes in the United States participating in our study, and an additional 6 devices deployed in France. Downstream throughputs from these homes ranged from 18 Mbps to 1 Gbps. During the duration of the deployment, we have recorded a total of 216,173 video sessions from four major video service providers: Netflix, YouTube, Amazon, and Twitch. Additionally, we periodically (four times per day) record the Internet capacity using active throughput measurements (*i.e.*, speed tests) from the embedded system. We collect this information to understand relationships between access link capacity and video QoE metrics.

2.3.2 Practical Challenges for Robust Models

Testing our models in a long-running deployment raised a new set of challenges that are not faced by offline models that operate on curated traces in controlled lab settings. Two factors, in particular, affected the accuracy of the models: (1) the challenge of accurately detecting the start and end of a video session in the presence of unrelated cross-traffic and attributing the video session traffic to a particular service; and (2) the granularity of training data versus what is practical to collect in an operational system.

Session Identification. Identifying a video session from encrypted network traffic is a challenge as network traffic is noisy. To detect session start and end times, we extend the method from Dimopoulos *et al.* [40], which identifies a spike in traffic to specific YouTube domains to determine the start of a video session and a silent period to indicate the end of a video session. We build on this approach to design a session identification method that generalizes this intuition across video services. Our analysis of video sessions across different services confirms that at the beginning of each video session there is a spike in the volume of traffic that comes from servers that are associated with the video services but are distinct from the servers that deliver video traffic. This activity can correspond to, for example, the download of the player code or thumbnail image downloads as users browse the catalog. Two other features are useful for identifying session boundaries. First, in addition to generating more traffic to other servers, video players also generate new network flows to download audio and video content when moving from one video to another. Second, considering the buffer-based approach of video services, most sessions have no video traffic

at the end, when the player is exclusively rendering the buffered content without having to further retrieve new segments. We rely on these intuitions to design a session identification method that relies on (1) the amount of non-video traffic generated by the service, (2) the time elapsed since the last video traffic activity, and (3) the rate of new flows initiated during the session.

We validate our session identification method against 2,347 streaming sessions from a real-world deployment for which we also had the ground truth available through our browser extension. Our results show that the estimate for session start time is within a few seconds of the actual time for most sessions; in some cases, the technique infers an early start time, perhaps as a result of non-video traffic mistakenly attributed to video traffic. Additionally, our system extracts and reports features at a fixed period (currently every five seconds), which makes it impractical to identify an exact start time.

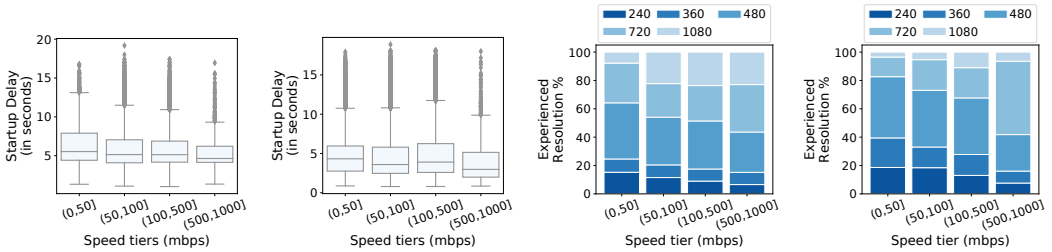
Data Granularity. Operational monitoring system cannot export information about each individual packet due to system various system constraints, *e.g.*, recording traces at modern network speeds would require exporting GBs of data per minute. It is hence common to report traffic statistics in fixed time intervals or time bins (*e.g.*, SNMP or Netflow polling intervals). Our system follows this behavior reporting statistics every five seconds. The training data that we and prior work collect has a precise session start time, whereas the data collected from a deployed system will only have data collected in time intervals, where the session start time might be *anywhere* in that interval, as demonstrated in the previous section. This corresponding mismatch in granularity creates a challenge for the inference models. Furthermore, any error in estimating the start time propagates to the time bins used for inference across the entire session, resulting in a situation where the time bins in the training and deployment data sets do not correspond to one another at all.

Intuitively, our approach to address these challenges involves accounting for additional noise that the practical monitoring introduces that is not present in a lab setting or in the training data. To do so, we introduce noise into the training data so that it more closely resembles the data collected from the deployment. The techniques that we apply are grounded in the general theory of domain adaptation [133]. They work as follows: Because the actual start time can fall anywhere within the five-second interval, we pre-process our training data and artificially adjust each session start time over a window of -5 to +5 seconds from the actual start value in increments of 0.5 seconds. For each new artificial start time, we recalculate all metrics based on this value for the entire session. This technique has two benefits: it makes the model more robust to noise, and it increases the volume of training data. We validated the domain-adapted models against 2,347 sessions collected across five homes of the deployment. We observe that for startup delay, the root mean square error improves—quite significantly for YouTube. We obtain similar improvements for resolution inference, except for YouTube. We posit that this result is attributable to the ground truth dataset collected; the YouTube data is heavily biased towards 360p resolutions (90+%), whereas all other services operated at higher (and more diverse) resolutions. While domain adaptation increases balance across classes, it may slightly impact classes with more prevalence in the training dataset.

2.3.3 Inference Results

We infer the startup delay and resolution for each video session in the deployment dataset in order to pose a question that both ISPs and customers may ask: how does access link capacity relate to startup delay?[3] Answering this question allows us to understand the

[3]We conducted this study in collaboration with *The Wall Street Journal* to understand the effect of home internet speeds on video streaming quality [137].



(a) NF startup time. (b) YT startup time. (c) NF resolution. (d) YT resolution.

Figure 2.2: Resolution and startup time vs. Users' speed tiers (NF is Netflix, YT is YouTube).

tangible benefits of paying for higher access capacity with regard to video streaming. Our results show that the speeds that consumers purchase from their ISPs have considerably diminishing returns with respect to video quality. For example, Figure 2.2 shows plots of the relationship between the startup delay and resolution experience by participant households across different speed tiers. We observe that, while some minimal improvements are observed as speeds increase, Internet speeds higher than about 100 Mbps of downstream throughput offer negligible improvements to video quality metrics such as startup delay and resolution. These results raise important questions for operators and consumers. Ultimately, operators may focus on other aspects of their networks to optimize video delivery; at the same time, consumers can be more informed about what downstream throughput they actually need from their ISPs to achieve acceptable application quality.

2.4 Conclusions and Lessons Learned

Internet service providers increasingly need ways to infer the quality of video streams from encrypted traffic, a process that involves both identifying the video sessions and segments and processing the resulting traffic to infer quality across a range of services. Our work builds on previous work that infers video quality for specific services or in controlled settings, and extends it in several ways, from model granularity, performance, and generality. Further, we applied the new models we developed to 16 months of traffic from 66 homes to demonstrate the applicability of our models in practice, and studied the relationship between access link capacity and video quality. We found, surprisingly, that higher access speeds provide only marginal improvements to video quality, especially at higher speeds.

While our work improves on the state-of-the-art on video quality inference, it also highlighted several weaknesses in our methodology to develop the models. Observing the methodology described in Section 2.2, we can point to several avenues for improvement across the entire machine learning pipeline. First, our work relied on a dataset collected in a controlled lab environment. To collect this dataset, we developed a variety of tools to automate the process and extract labels for the recorded traffic. Yet, while one of the largest datasets of its kind, the data we collected still failed to fully resemble the data observed in the deployment. Second, our work demonstrated the need of identifying data representations (*e.g.*, time series of video segment downloads) that highly enhance the performance of the developed models. However, our methodology fell short of addressing the impact of these representations on the ability of a measurement system to collect these metrics at scale, solely pointing to potential challenges, rather than providing concrete answers. Finding efficient data representations can help strike a balanced trade-off between model

performance and system collection costs. Overall, our work lacked a clear definition of system costs and their impact on model deployment. Third, our work exclusively performed model execution after the fact, *i.e.*, collected measurements were aggregated to a remote server for later use. However, various use cases might require the deployment of real-time execution models, which we did not address. Finally, our work captured data from a specific snapshot in time, and did not consider potential decays of model accuracy over time. Yet, services and networks are in constant evolution. Our dataset was collected over a specific period of time, while the deployment data occurred during a long period occurring well after the moment the models were trained. This difference in time can lead to drift in the data distribution, which could ultimately impact models' performance.

These deficiencies inspired the rest of the work presented in this manuscript. The rest of the manuscript, then, describes how during the following years we addressed these challenges and developed a number of solutions that move us closer to ML-based networked systems that can be deployed in practice.

Chapter 3

Data Generation

"NetDiffusion: Network Data Augmentation Through Protocol-Constrained Traffic Generation", ACM Sigmetrics 2024 [61]

Takeaways

Datasets of labeled network traces are essential for a multitude of machine learning (ML) tasks in networking, yet their availability is hindered by privacy and maintenance concerns, such as data staleness. To overcome this limitation, synthetic network traces can often augment existing datasets. Unfortunately, current synthetic trace generation methods, which typically produce only aggregated flow statistics or a few selected packet attributes, do not always suffice, especially when model training relies on having features that are only available from packet traces. This shortfall manifests in both insufficient statistical resemblance to real traces and suboptimal performance on ML tasks when employed for data augmentation. To solve this problem, we apply diffusion models to generate high-resolution synthetic network traffic traces. We present `NetDiffusion`, a tool that uses a finely-tuned, controlled variant of a Stable Diffusion model to generate synthetic network traffic that is high fidelity and conforms to protocol specifications. Our evaluation demonstrates that packet captures generated from `NetDiffusion` can achieve higher statistical similarity to real data and improved ML model performance than current state-of-the-art approaches (*e.g.*, GAN-based approaches). Furthermore, our synthetic traces are compatible with common network analysis tools and support a myriad of network tasks, suggesting that `NetDiffusion` can serve a broader spectrum of network analysis and testing tasks, extending beyond ML-centric applications.

Contents

3.1 Introduction

Modern networks are increasingly reliant on machine learning (ML) techniques for a wide range of management tasks, ranging from security to performance optimization. A central impediment when training network-focused ML models is the scarcity of labeled network datasets, as their collection and sharing are often associated with high costs and privacy concerns, particularly when data is collected from real-world networks [129, 88, 92, 12, 36]. Unfortunately, existing public datasets rarely receive updates, making them static and unable to reflect evolving network behaviors [67, 110, 73]. These limitations hinder the ability to train robust ML models that accurately reflect evolving real-world network conditions.

These challenges can be addressed through the creation of new synthetic network traces based on existing or new, but small, datasets. This approach aims to preserve the inherent characteristics of network traffic while introducing variations, thereby enhancing dataset size and diversity [121, 151, 99, 155, 130, 143, 78, 157]. Unfortunately, current state-of-the-art synthetic trace generation methods, particularly those based on Generative Adversarial Networks (GANs)-based methods [109, 155, 78, 151, 149], are not always sufficient for producing high-quality synthetic network traffic. Specifically, these approaches tend to focus on a limited set of attributes or statistics, as early machine learning for network tasks often relied on basic flow statistics for classification [102, 39, 34, 74, 75, 21, 66]. With recent ML advancements utilizing detailed raw network traffic to achieve enhanced classification accuracy [108, 160, 83, 14, 154, 146, 120, 35, 27, 86, 134], there is a clear need for synthetic traffic generation that includes the intricate, potentially unforeseen patterns present in full network traces. Yet, existing traffic generation methods face two main issues: (1) a lack of statistical similarity with real data due to the limited attributes in existing methods, making the synthetic data highly sensitive to variations, and (2) unsatisfactory classification accuracy when synthetic statistical attributes are used to augment existing datasets. Moreover, their simplistic attribute focus and disregard for transport and network layer protocol behaviors prevent their use with traditional networking tools such as tcpreplay [37] or Wireshark [20].

Fortunately, the general increase in available computational power and the breakthroughs in high-resolution image generation techniques, particularly diffusion models [128, 111, 106], offer a promising avenue to overcome these challenges. Specifically, we harness the capabilities of text-to-image diffusion models, which execute conditioned generation based on descriptive text prompts. These models are adept at creating detailed, accurate visual representations from textual descriptions. By translating the intricate characteristics of network traffic into an appropriate image format, we can tap into the unique advantages offered by these models. In contrast to GANs, diffusion models are able to capture both broad patterns and detailed dependencies. This inherent generative quality makes them an ideal choice for producing network traces with high statistical resemblance to real traffic and full packet header values. By incorporating conditioning techniques, diffusion models can generate structured data that conforms to specific network properties, which ensures the desired sequential inter-packet characteristics and rough protocol dependencies. Moreover, the gradient dynamics of the training process in diffusion models is a lot more stable than GANs. These attributes collectively position diffusion models as a compelling choice for advancing the state-of-the-art for synthetic network trace generation, addressing the extant limitations of current methodologies. The rest of the Chapter presents `NetDiffusion`, our diffusion-based framework to generate synthetic raw network traffic that complies with transport and network layer protocol rules.

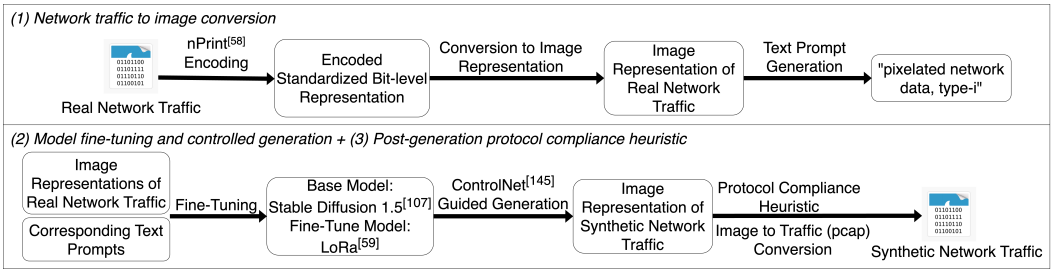


Figure 3.1: Generation Framework Overview.

3.2 NetDiffusion

To address the need of synthetic raw network traffic generation, we introduce NetDiffusion, a framework that harnesses controlled text-to-image diffusion models [111] to generate synthetic raw network traffic that complies with transport and network layer protocol rules.[1]

3.2.1 Network Data Augmentation Through Protocol-Constrained Traffic Generation

NetDiffusion is designed around three main components, shown in Figure 3.1: (1) a conversion process for transforming raw packet captures to image representations (and vice versa); (2) a fine-tuned text-to-image diffusion model based on packet capture-converted images for generating synthetic packet captures; and (3) a post-generation use of domain knowledge-based heuristics to finely check and adjust the generated fields, ensuring their semantic correctness in terms of compliance with transport and network layer protocol rules.

Network Traffic to Image Conversion. Network traffic data, with intricate inter-packet dependencies and vast range of attributes, presents a complex landscape that introduces specific challenges when it comes to accurate representation and efficient learning. Network traffic data exhibits high dimensionality, as well as inherently contains sequential dependencies between packets. Thankfully, images inherently capture spatial hierarchies, which is crucial for representing intricate inter-packet and intra-packet dependencies in network traffic. Pixels in images naturally form patterns and structures. Deep learning models, especially convolutional neural networks (CNNs), are adept at exploiting these structures to capture both local and global dependencies. Unlike traditional tabular formats where data points might be perceived as independent entities, images inherently emphasize the significance of a packet concerning its neighboring packets, preserving crucial contextual information [123, 29, 71, 87].

To arrive at image representations of network traffic, we first encode packet captures (pcaps) using nPrint [56], which converts network traffic into standardized bits where each bit corresponds to a packet header field bit. This binary representation is simple yet effective, where the presence or absence of a bit in the packet header is denoted as 1 or 0 respectively, and a missing header bit is represented as -1. This encoding scheme ensures a standardized representation irrespective of the protocol in use. The payload content is not encoded since it is often encrypted. However, the size of the packet payloads can be inferred from other encoded header fields such as the IP Total Length fields. Following

[1] The source code of NetDiffusion is available at https://github.com/noise-lab/NetDiffusion_Generator/.

this encoding, a sequence of packets in a pcap is converted into a matrix, which is then interpeted as an image. The colors green, red, and gray represent a set bit (1), an unset bit (0), and a vacant bit (-1), respectively. This color coding provides a visually intuitive representation of the network traffic. We finally group the packets in groups of 1024, representing the packet headers of the first 1024 packets in a flow. Through this process, any network traffic in pcap format is transformed into an image with dimensions of 1024 pixels in width and 1024 pixels in height, with each row of pixels representing a packet in the network traffic flow. Further, any image in this format can be converted back to pcaps in a straightforward manner. This representation not only preserves the complexity of the data but also retains the essential sequential relationships among packets, laying a robust foundation for the ensuing steps in the `NetDiffusion` pipeline.

Fine-Tuned Diffusion Model. Diffusion models synthesize data by modeling data generation as the process of noise removal from noisy data (referred to as the reverse process) [131, 54]. Diffusion models excel in capturing and replicating intricate data distributions with remarkable fidelity [55, 117, 101] and are adept at generating and managing high-resolution images [106, 111, 98]. We take advantage of the properties of diffusion models by building upon Stable Diffusion 1.5 [111] and fine-tuning this model on our specific network datasets, making it aptly suited for generating synthetic network traffic that mirrors the complexities and nuances of real-world network traffic. To facilitate this fine-tuning, we employ Low-Rank Adaptation (LoRa) [57], which is a training technique tailored to fine-tune diffusion models, particularly in text-to-image diffusion models. Its crux lies in enabling the diffusion model to learn new concepts or styles effectively, while maintaining a manageable model file size. This is beneficial given the traditionally large sizes of models like Stable Diffusion, which can be cumbersome for storage and deployment. With LoRa, the resultant models are compact, striking a balance between file size and training capability. This compactness does not sacrifice the model's ability but rather applies minute yet effective changes to the base/foundational model, ensuring that the core knowledge remains intact while adapting to new data.

In our fine-tuning process, we start by sampling classes of real network traffic from our dataset that we aim to generate synthetically. These traffic samples are then transformed into their image representations. For each of these images, we craft an unique encoded text prompt (*e.g.*, "pixelated network data, type-0" for Netflix traffic) that succinctly describes its class type. Consequently, this results in a number of text prompt categories corresponding to the variety of network traffic types within the dataset. The choice of our encoded prompt, though seemingly simplistic, achieves two main objectives. It offers a specific vocabulary that reduces ambiguity and ensures the model hones in on the network traffic's nuances. Additionally, it minimizes interference from the base model's original word embeddings, optimizing the generative process. Experimentally, we found that this specific prompt structure provides a balance between specificity and simplicity to prevent overfitting and misinterpretations, leading to better results.

Controlled Generation and Post-Generation Protocol Compliance Heuristics. A challenge arises from the inherent flexibility of general diffusion models: While they are designed to foster creativity in the generated output, it can lead to anomalies in the context of network traffic generation. For example, generated traffic might incorrectly populate packet header fields, leading to protocol distribution discrepancies between synthetic and real traffic. To solve this challenge, we apply a two step process. First, we leverage the controllable nature of diffusion models to incorporate ControlNet [158] into the generation process. ControlNet is a commonly used neural network architecture designed to add spatial conditioning controls to large, pre-trained text-to-image diffusion models. It capitalizes on

the robust encoding layers of these models, which are pre-trained with vast datasets, to learn a diverse set of conditional controls.

Second, we identify a subset of critical header fields that mandate strict adherence to their formatting rules, *e.g.*, sequence and acknowledgement numbers, and we develop a systematic way to calculate their correct values based on other generated fields. This is achieved by constructing two dependency trees—one for intra-packet header field dependencies and another for inter-packet dependencies. These trees are built upon domain knowledge and are sourced from standard network protocol documentation [1, 3, 2, 4, 5, 20]. Although constructing them requires significant manual effort to extract critical protocol rules from these standards, this process is a one-time endeavor.

3.2.2 Main Results

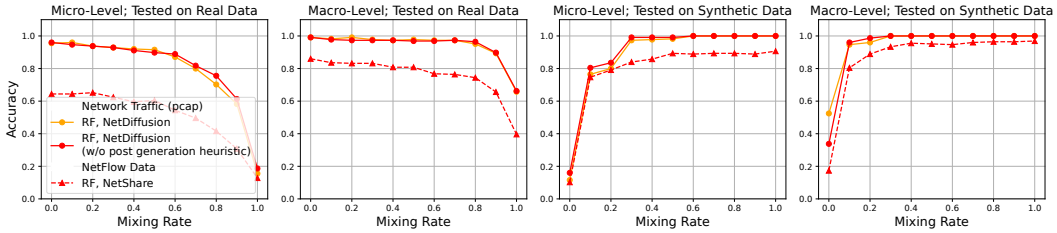
To assess the effectiveness of our generative framework, we applied it to an exemplary real network traffic dataset, generating its synthetic counterpart as a case study. Our ML-oriented evaluation comprises two main analyses: a statistical comparison to gauge the fidelity of the synthetic data and a model accuracy assessment to determine its utility in enhancing ML predictions.

Statistical Similarity Results. A primary measure of synthetic data quality is its statistical resemblance to the original data. This comparison is critical as the essence of synthetic data lies in its ability to represent the statistical properties of the real data without mirroring it exactly. Ensuring statistical similarity ensures that models trained on synthetic data generalize well to real-world scenarios. In our evaluation, we benchmarked our synthetic data against two baselines: the NetShare method, which produces synthetic NetFlow attributes and outperforms most of the other GANs-based methods [155], and a naive random generation approach. The latter, by generating purely random values, acts as a worst-case scenario, illustrating the lower bounds of similarity and underscoring the value added by more sophisticated methods. We present results for both the generation of entire pcaps, as well as only generation of NetFlow attributes. Overall, our experiments showcase the inherent challenges that exist in replicating the complex pcap format. Yet, `NetDiffusion` demonstrates the capability to produce synthetic data with high statistical similarity, even surpassing existing methods on simpler formats, validating its potential as a robust tool for data augmentation in the realm of raw network traffic.[2]

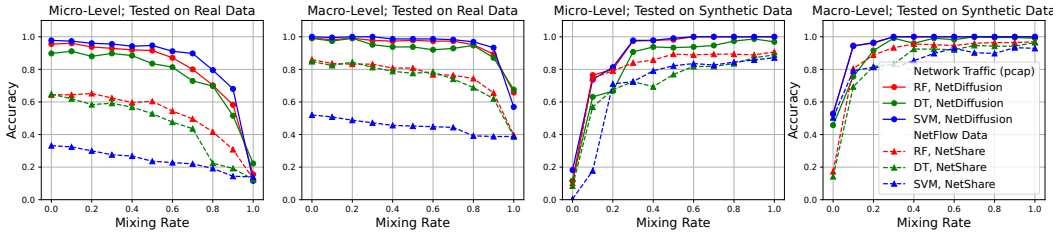
ML Classification Results. To gauge the efficacy of our synthetic network traffic in ML-based data augmentation, we employ it to two classification tasks. The first task aims to categorize network flows at a granular level, aligning them with their corresponding applications (micro-level). The second task operates at a broader scale, classifying flows into their overarching services (macro-level). We conduct the evaluation using three prominent models: random forest (RF), decision tree (DT), and support vector machine (SVM). We explore three distinct augmentation scenarios, utilizing synthetic data: (1) a scenario where either the training or testing set is entirely composed of synthetic data, *e.g.*, training exclusively on synthetic data and testing on real data, and vice versa. (2) A scenario where synthetic data is combined with real data at varying proportions, *e.g.*, a 50-50 split between synthetic and real data during training. (3) A scenario where synthetic data is used to address and rectify class imbalances in the training set.

By integrating `NetDiffusion`-generated network traffic into the real dataset data at varying proportions during training and testing, we observe a general increase in accuracy compared to both baselines. Using the RF model as an example, Figure 3.2a shows

[2]For a detailed evaluation, please refer to our paper [61].



(a) Classification accuracy comparison using the RF model with mixed data proportions. Datasets augmented with `NetDiffusion`-generated traffic consistently outperform those using NetShare-produced NetFlow attributes.



(b) Comparative ML performance across different model choices using `NetDiffusion`-augmented datasets versus NetShare-augmented NetFlow datasets. `NetDiffusion`consistently yields superior results.

Figure 3.2: Evaluation result on mixed data proportions.

that models trained with dataset augmented with `NetDiffusion`-generated traffic consistently achieve higher classification accuracy than those with NetShare-produced NetFlow attributes. When testing on real data, models trained entirely on real network traffic demonstrate notably higher accuracy than those trained solely on real NetFlow. This improvement is attributed to our synthetic data's significantly high statistical resemblance to the real dataset. Additionally, our method shows promise in addressing class imbalance issues, enhancing the accuracy of ML models in such cases.

3.3 Conclusions

Synthetic traces, primarily emphasizing certain flow statistics or packet attributes, are frequently used to support ML tasks in networking. However, their limited alignment with real traces and challenges in converting to raw network traffic hinder both their ML performance and broader applicability in conventional network analyses. In our research, we tap into the promising capabilities of diffusion models, known for their high-quality data generation, to enhance synthetic network traffic production. We present `NetDiffusion`, a tool to produce synthetic network traffic, captured as pcaps covering all packet headers. Our evaluation reveals that NetDiffusion's pcaps closely resemble authentic data and bolster ML model performance, outperforming current methods. These synthetic traces integrate with traditional network tools, support retransmission, and suit a broad range of network tasks. The rich features in NetDiffusion's outputs position it as a new core tool for diverse network analysis and testing tasks.

Looking ahead, several avenues deserve further exploration. First, transformers have shown efficacy in generating sequential data like text, suggesting their potential in network traffic generation. Key challenges include appropriate packet capture tokenization

and maintaining long contexts for generating meaningful flows. `NetDiffusion` can also attempt to address the issue of long contexts by simply increasing image resolution in future work. Second, our current protocol rule-compliance approach is post-generative, given the intricate nature of managing inter-dependent constraints during the diffusion generation process. A future direction is to embed these rules directly within the generation pipeline, eliminating the need for subsequent adjustments. At the same time, although we leverage pre-trained ControlNet models for controlled generation in this implementation of `NetDiffusion`, it is viable to train dedicated ControlNet models from scratch to realized finer-grained control beyond general protocol distribution, which can also serve as a potential solution for avoiding excessive post-generation adjustments. A feasible strategy for achieving this involves curating ControlNet training datasets focused on specific attributes, such as exclusively featuring TCP options. This targeted training may help effectively enforce constraints on particular aspects of the flows. Additionally, as time dependencies play a pivotal role, we aim to refine the diffusion models to directly learn and generate time series, providing a more nuanced approach to inter-packet time dependencies. Our generation's horizon is presently capped at 1,024 packets per flow sample, a limitation we seek to address, possibly through techniques like tabular diffusion that retains packet dependencies or sequential flow generation. Another intriguing prospect is building a network-specific diffusion foundation model, which could further heighten generation accuracy. Lastly, generating semantically meaningful payloads remains a challenge, with potential solutions like autoencoders offering a promising direction for future work.

Chapter 4

Feature Engineering and Model Training

"Traffic Refinery: Cost-Aware Data Representation for Machine Learning on Network Traffic", ACM Sigmetrics 2022 [24]

"CATO: End-to-end Optimization of ML Traffic Analysis Pipelines", Preprint [145]

Takeaways

Network management often relies on machine learning to make predictions about performance and security from network traffic. Often, for these tasks, the representation of the traffic is as important as the choice of the model. The features that the model relies on, and the representation of those features, determine model accuracy, as well as where and whether the model can be deployed in practice. Thus, the design and evaluation of these models ultimately requires understanding not only model accuracy but also the systems costs associated with deploying the model in an operational network. We develop two systems-driven solution to design and train machine learning models that aim to strike the correct balance between systems costs related to feature extraction and model accuracy. First, we develop `Traffic Refinery`, a new framework and system that enables a joint evaluation of both the conventional notions of machine learning performance (*e.g.*, model accuracy) and the systems-level costs of different representations of network traffic. `Traffic Refinery` both highlights this design space and makes it possible to explore different representations for learning tasks. Second, we develop `CATO`, a framework that addresses the problem of jointly optimizing the predictive performance of models and the associated systems costs of the model serving pipeline. `CATO` leverages recent advances in multi-objective Bayesian optimization to efficiently identify Pareto-optimal configurations, and automatically compiles end-to-end optimized serving pipelines that can be deployed in real networks.

Contents

4.1 Introduction

Machine learning (ML) models have grown to outperform traditional rule-based heuristics for a variety of traffic analysis applications, such as traffic classification [64, 82], intrusion detection [147], and QoE inference [25, 90]. Over the past few years, researchers have explored various approaches to developing more accurate models, ranging from better feature selection to employing sophisticated model types and traffic representations [95, 66, 48, 104, 83, 108, 160, 120, 56, 11, 152, 23]. However, the predictive performance of ML-based solutions often overshadows an equally critical aspect—the end-to-end efficiency of the serving pipeline that processes network traffic and executes the model.

For traffic analysis, a significant challenge lies not just in developing accurate models, but in meeting the performance demands of the network. Many network applications must operate in real time with sub-second reaction times and/or process hundreds of gigabits per second of traffic without packet loss [144]. Unfortunately, models developed without consideration of the associated systems costs of serving them in real networks often turn out to be unusable in practice. Current approaches to this problem typically rely on lightweight models [77], programmable hardware [148, 64], or early inference techniques [21, 103], but many of these force developers to compromise on predictive performance instead [127, 28, 148].

Two central obstacles emerge in building models that aim to strike the right balance between model performance and systems constraints. First, the choice of features and their representation in the model is often as important as the choice of the model itself. The features that the model relies on, and the representation of those features, ultimately determine model accuracy, as well as where and whether the model can be deployed in practice. On the surface, choosing representations that maximize model performance seems straightforward: Any network operator or researcher knows full well that collecting raw packet traces can attain maximum flexibility to explore transformations and representations that result in the best model performance. Yet, unfortunately, capturing raw packet traces often proves to be impractical. In large networks, raw packet traces produce massive amounts of data introducing storage and bandwidth requirements that are often prohibitive. Limiting the duration of a pcap collection can reduce data storage requirements, but might negatively affect the accuracy of the produced models as the limited capture may not represent network conditions at different times. Conversely, pcaps collected in a controlled laboratory environment might produce models not directly applicable in practice because operational networks include other traffic characteristics that are hard to capture in a lab environment. Due to these reasons, experiments (and much past work) that demonstrate a model's accuracy turn out to be non-viable in practice because the systems costs of deploying and maintaining the model are prohibitive. An operator may ultimately need to explore costs across state, processing, storage, and latency to understand whether a given pipeline can work in its network.

Second, even when the costs of feature extraction are well understood, the end-to-end serving pipeline that processes network traffic and executes the model still needs to be optimized, balancing the accuracy-performance tradeoff. However, achieving this balance is difficult. The end-to-end latency and throughput costs of the serving pipeline, which includes packet capture, feature extraction, *and* model inference, are difficult to approximate without real measurements. Furthermore, the search space over optimal feature representations is exponential in the number of available candidate features, and also includes considerations for how far into a flow to wait before making a prediction. The added complexity of not just considering one objective, but two, makes end-to-end optimization of such systems an open challenge.

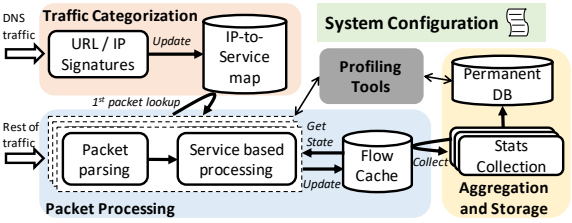


Figure 4.1: Traffic Refinery system overview.

```
{
  "Name": "ServiceName",
  "Filter": {
    "DomainsString": ["domain.x",
        ...],
    "Prefixes": ["10.0.0.0/18", ...
        ]
  },
  "Collect": [FeatureSetA,
      FeatureSetB, ...],
  "Emit": 10
}
```

Listing 4.1: Configuration example.

The rest of the Chapter presents Traffic Refinery and CATO, two systems driven solutions to explore cost-aware data representations and automatically train machine learning models that strike optimal accuracy-performance balance.

4.2 Traffic Refinery

To explore network traffic feature representations and its subsequent effect on both the performance of prediction models and collection cost, we need a way to easily collect different representations from network traffic. To enable such exploration, we implement Traffic Refinery [9],[1] which works both for *data representation design*, helping network operators explore the accuracy-cost tradeoffs of different data representations for an inference task; and for *customized data collection in production*, whereby Traffic Refinery can be deployed online to extract custom features.

4.2.1 Joint Exploration of Cost and Model Performance

Figure 4.1 shows an overview of the system. Traffic Refinery is implemented in Go [8] to exploit performance and flexibility, as well as its built-in benchmarking tools. The system design revolves around three guidelines: (1) Detect flows and applications of interest early in the processing pipeline to avoid unnecessary overhead; (2) Support state-of-the-art packet processing while minimizing the entry cost for extending which features to collect; (3) Aggregate flow statistics at regular time intervals and store for future consumption. The pipeline has three components:

Traffic Categorization. The first module is a traffic categorization module responsible for associating network traffic with applications of interest. Traffic Refinery implements a cache to map remote IP addresses to services accessed by users. The map supports identifying the services flows belong to by using one of two methods: (1) Using the domain name of the service captured from DNS queries or (2) Using exact IP prefixes.

Packet Capture and Processing. The second module is a packet capture and processing module that collects network flow statistics and tracks their state at line rate. Traffic Refinery implements parallel traffic processing through a pool of worker processes, allowing the system to scale capacity and take advantage of multicore CPU architectures. We implement a flow cache used to store a general data structure containing state and statistics related to a network flow. The general flow data structure allows storing different flow types, and differing underlying statistics using a single interface. Furthermore, it includes,

[1] The source code of Traffic Refinery is available at https://traffic-refinery.github.io

if applicable, an identifier to match the services the flow belongs to. This information permits the system to determine early in the pipeline whether a given packet requires additional processing. Finally, the workers pool processes all non-DNS packets. Each worker has a dedicated capture interface to read incoming packets. As a first step, each worker pre-parses MAC, network, and transport headers, which yields useful information such as the direction of the traffic flow, the protocols, and the addresses and ports of the traffic. The system then performs additional operations on the packet depending on the service category assigned to the packet by inspecting the flow's service identifier in the cache.

Aggregation and Storage. The third module queries the flow cache to obtain features and statistics about each traffic flow and stores higher-level features concerning the applications of interest for later processing. `Traffic Refinery` exports high-level flow features and data representations at regular time intervals. Upon firing collection events, the system loops through the flows belonging to a given service class and performs the required transformations (*e.g.*, aggregation or sampling) to produce the data representation of the traffic class.

`Traffic Refinery` is customizable through a configuration file written in JSON and enables the collection of user defined data representations. The configuration provides a way to tune system parameters (*e.g.*, which interfaces to use for capture) as well as the definitions of service classes to monitor. A service class includes three pieces of information that establish a logical pipeline to collect the specified feature sets for each targeted service class: (1) which flows to monitor; (2) how to represent the underlying flows in terms of features; (3) at what time granularity features should be represented. Listing 4.1 shows the JSON format used.

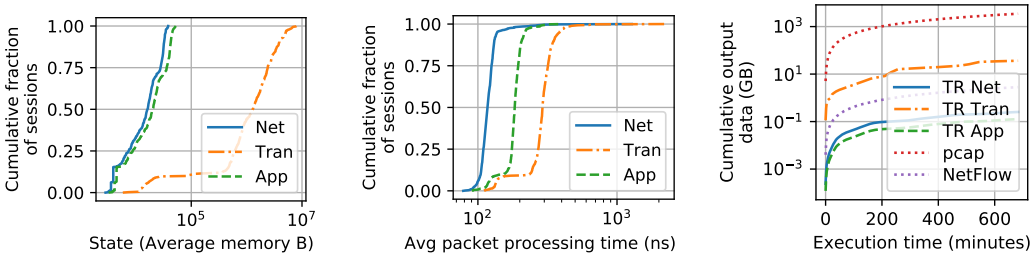
Further, we design `Traffic Refinery` to facilitate the exploration of how different representations affect model performance and collection cost. To do so, we design `Traffic Refinery` to use convenient flow abstraction interfaces to allow for quick implementation of user-defined collection methods for features and their aggregated statistics. Each flow data structure implements two functions that define how to handle a packet in the latter two steps of the processing pipeline: (1) an `AddPacket` function that defines how to update the flow state metrics using the pre-processed information parsed from the packet headers; and (2) a `CollectFeatures` function that allows the user to specify how to aggregate the features collected for output when the collection time interval expires.

4.2.2 Main Results

We use `Traffic Refinery` to demonstrate the value of jointly exploring data representations for modeling and their associated costs for two supervised learning problems in networking: video quality inference from encrypted traffic and malware detection.[2] As discussed in Chapter 2, our previous work [25] categorized useful features for video quality inference into three groups that correspond to layers of the network stack: Network, Transport, and Application Layer features. We add approximately 100 lines of Go code to implement in `Traffic Refinery` the feature calculation functions to extract application features (*i.e.*, `VideoSegments`). Further, we use built-in feature classes to collect network (*i.e.*, `PacketCounters`) and transport (*i.e.*, `TCPCounters`) features.

Data Representation Costs. We evaluate system-related costs of the three classes of features used for the video quality inference problem: network, transport, and application features. First we use `Traffic Refinery`'s profiling tools to quantify the fine-grained costs imposed by tracking video streaming sessions. To do so, we profile the per-feature state

[2]Please refer to the full paper [24] for malware detection results.



(a) State required for different representations.

(b) Processing costs for different representations.

(c) Storage costs for different representations.

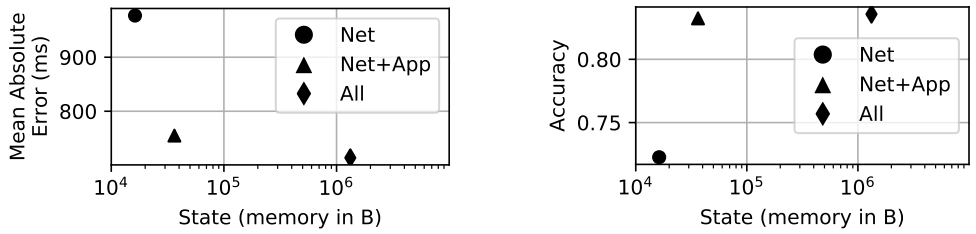
Figure 4.2: Cost profiling for video inference models.

and processing costs for pre-recorded packet traces with 1,000 video streaming sessions split across four major video streaming services (Netflix, YouTube, Amazon Prime Video, and Twitch). Then, we study the effect of collecting the different classes of features at scale by deploying the system in a 10 Gbps interconnect link.

We find that while some features add relatively little state (*i.e.*, memory) and long-term storage costs, others require substantially more resources. Conversely, processing requirements are within the same order of magnitude for all three classes of features. Figure 4.2a shows the cumulative distribution of memory in Bytes across all analyzed video streaming sessions. The reported results highlight how collecting transport layer features can heavily impact the amount of memory used by the system. In particular, collecting transport features can require up to three orders of magnitude more memory compared to network and application features. Transport features require historical flow information (*e.g.*, all packets) in contrast with network features that require solely simple counters. As expected, Figure 4.2c shows that storage costs follow similar trends as the state costs previously shown. This is not surprising as the exported information is a representation of the state contained in memory. In contrast, Figure 4.2b shows distributions of the time required to process different feature classes. Collecting simple network counters requires the least processing time, followed by application and transport features. While there are differences among the three classes, the difference is relatively small and within the same order of magnitude.

Model Performance. We then study the relationship between model performance and system costs for online video quality inference. We use previously developed models but explicitly explore how *data representation* affects model performance. We focus on state-related costs (*i.e.*, memory), as for video quality inference, state costs mirror storage costs and the differences in processing costs of the feature classes is not significant. Interestingly, we find that the relationship between state cost and model performance is not proportional. More importantly, we find that it is often possible to significantly reduce the state-related requirements of a model without significantly compromising prediction performance, further bolstering the case for systems like `Traffic Refinery` that allow for flexible data representations.

Figure 4.3 shows the relationship between model performance and state costs. As shown, network features alone can provide a lightweight solution to infer both startup delay and resolution but this yields the lowest model performance. Adding application layer features contributes to a very small additional memory overhead. This result is particularly important for resolution where models with video segments alone perform basically as well as combining all others. Further, adding transport features (labeled "All" in the figure)



(a) Startup delay (lower is better).

(b) Resolution (higher is better).

Figure 4.3: The relationship between features state cost and model performance for video streaming quality inference (marker shapes identify layers used).

provides limited benefits in terms of added performance—40 ms on average lower errors for startup delay and less than 0.5% higher accuracy for resolution. Even for startup delay where using transport features can improve the mean absolute error by a larger margin, this comes at the cost of two orders of magnitude higher memory usage.

4.3 CATO

`Traffic Refinery` demonstrates both the need and the *potential* for exploring how different data representations can affect model accuracy and systems costs. However, achieving a perfect balance between these two dimensions is difficult. The search space over optimal feature representations is exponential in the number of available candidate features, and also includes considerations for how far into a flow to wait before making a prediction. The added complexity of not just considering one objective, but two, makes end-to-end optimization of such systems an open challenge. To solve this challenge we present `CATO`, the first framework that systematically optimizes the systems costs and model performance of ML-based traffic analysis pipelines.

4.3.1 Cost-Aware Model Training

The goal of `CATO` is to automatically construct traffic analysis pipelines that jointly minimize the end-to-end systems costs of model serving while maximizing the model's predictive performance. At its core, `CATO` combines a multi-objective Bayesian optimization-guided search with a novel pipeline generator and feature representation profiler to produce serving pipelines suitable for deployment in real networks. `CATO` constructs end-to-end optimized traffic analysis pipelines according to the systems cost and model performance objective functions. It does so by efficiently identifying Pareto-optimal feature representations, and generating ready-to-deploy serving pipelines for a given model from those feature representations. Note that `CATO` is agnostic to the type of model used in the pipeline.

Figure 4.4 depicts the high-level design, which consists of an Optimizer and a Profiler:

- The Optimizer takes the set of candidate features and maximum connection depth (*i.e.*, the number of flow packets to use for inference), and performs a multi-objective Bayesian optimization-guided search over the feature representation space. It periodically queries the Profiler for the systems costs and model performance of its sampled feature representations, which it uses to further refine the search for the Pareto front.

- The Profiler accepts queries from the Optimizer, generates compiled binaries for the

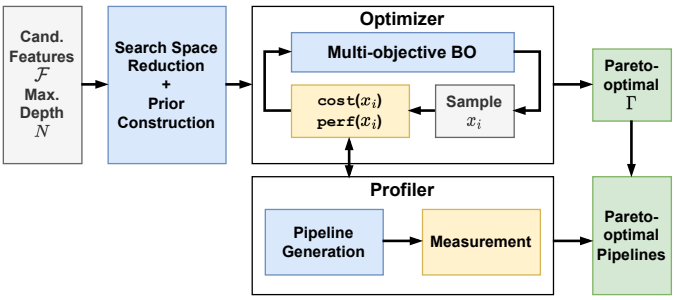


Figure 4.4: CATO combines a multi-objective BO-based Optimizer and a realistic pipeline Profiler to construct and validate efficient ML-based traffic analysis serving pipelines.

end-to-end serving pipeline, and runs them to accurately measure systems costs and model performance. These measurements serve the dual purpose of guiding the Optimizer towards Pareto-optimal solutions and validating the in-network performance of the resulting traffic analysis pipelines.

We now describe the two components in more detail.

The CATO Optimizer. In general, measuring systems costs and model performance for an arbitrary feature representation is computationally expensive. It involves generating the serving pipeline, training and evaluating the ML model, and measuring performance costs either through simulation or in physical testbeds. The massive size of the search space and the computational cost of evaluating the objective functions precludes the possibility of exhaustively searching all possible configurations. To handle this intractability, CATO leverages Bayesian Optimization (BO), building on recent developments in multi-objective design space exploration [96] and sample-efficient BO [60] to efficiently estimate the Pareto front of solutions. However, in its basic form, BO has several limitations. Conventional applications of BO typically involve single-objective, low-dimensional (fewer than 20) search spaces [46, 119]. Unfortunately, our traffic analysis problem is inherently multi-objective, high-dimensional, and involves a complex search space with mixed categorical (features) and numerical (connection depth) variables.

To address this, we augment the CATO Optimizer with two preprocessing techniques to improve its sample efficiency (Figure 4.4). The first is a dimensionality reduction step that strategically discards candidate features that are unlikely to improve the model's predictive performance regardless of its impact on the end-to-end systems costs. By default, we exclude features with a mutual information [141] score of zero, which indicates no direct informational relationship with the target variable. The second technique incorporates prior probabilities into the BO formulation, accelerating the search by providing the Optimizer with "hints" about the approximate locations of Pareto-optimal feature representations. To account for both objectives, CATO constructs two sets of priors: one over the feature space that targets model performance, and one over the connection depth that targets systems costs. The set of priors over the feature space encodes each feature's relative contribution to the model's performance, and are derived from the mutual information scores computed in the dimensionality reduction step. These adaptations encourage CATO to more frequently explore regions of the search space that include features with higher predictive power.

The CATO Profiler. The CATO Profiler evaluates the feature representations sampled by the Optimizer based on the concrete definitions of systems costs and model performance. To accomplish this, it compiles customized code for the packet capture and feature extraction stages of each sampled point, trains the model, and runs the full serving pipeline to *directly*

measure its end-to-end systems costs and model performance. This measurement serves two purposes: (1) guiding the search process of the Optimizer, and (2) validating the in-network performance of identified solutions.

CATO employs *conditional compilation* to build and run customized end-to-end serving pipelines tailored to each configuration. The resulting binary matches the performance of a manually implemented pipeline, containing only the set of operations needed to collect traffic data up to the specified connection depth, extract the corresponding features, and execute the model inference. This technique not only constructs fully operational traffic analysis pipelines, but also provides the flexibility to accurately measure any point in the search space. Ultimately, CATO presents a testbed interface that replicates a real-world deployment scenario of the pipeline.

For model performance measurements, the Profiler trains a fresh model for each feature representation sampled by the Optimizer and directly measures its predictive performance to account for any interaction effects between features. The final performance metric is derived from a hold-out test set, which ensures an unbiased evaluation. This approach is inspired by established wrapper methods [69] in feature selection. For systems cost measurements, CATO either simulates traffic inputs from the training data, or, when feasible, deploys the full serving pipeline in its target network environment for end-to-end measurements. While each measurement is expensive, the Optimizer is intentionally designed to minimize the number of measurements needed to approximate the Pareto front.

4.3.2 Main Results

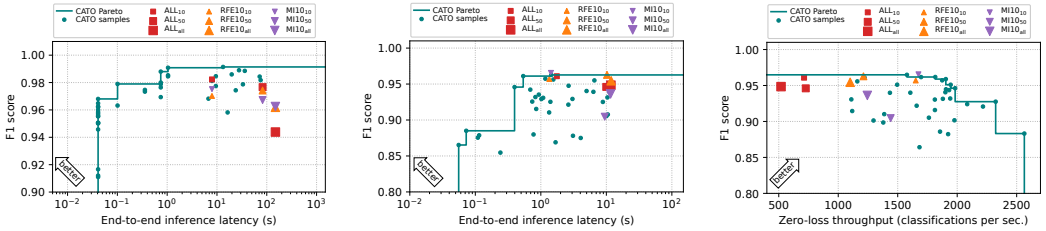
We consider two typical traffic analysis use cases, web application classification and IoT device recognition, and evaluate CATO's ability to identify optimal models over a variety of configurations. We show that CATO's joint optimization of systems costs and model performance can help traffic analysis applications achieve substantially lower inference latency and higher throughput without compromising model performance, and in many cases improve upon both metrics.[3]

Model Serving Performance. We examine the end-to-end efficiency and predictive performance of serving pipelines optimized by CATO. We compare the F1 score, inference latency, and zero-loss classification throughput to popular feature optimization methods:

- ALL: Use all available features.
- RFE10: Select the top ten features by recursive feature elimination [53]. RFE trains a model using all available features, then iteratively removes the least important feature and retrains until the desired number remains.
- MI10: Select the top ten features based on mutual information [141]. This is a model agnostic algorithm that measures how much information each feature contributes to the target variable and picks the most relevant ones.

Figures 4.5a and 4.5b show the end-to-end inference latency and F1 scores for IoT device recognition and Application classification, respectively. Note that each of the baseline methods must pre-specify a packet depth and only produces a single solution, whereas CATO estimates a Pareto front of optimal solutions. This Pareto front is the set of non-dominated feature representations sampled by the Optimizer. We can see that for IoT recognition, all points in CATO's Pareto front dominate the baseline solutions. In other words, for any given representation chosen by one of the baselines, CATO finds a different one that achieves the

[3]Please refer to the full paper for all evaluation results [145].



(a) IoT identification latency (b) Application classification latency (c) Application classification throughput

Figure 4.5: Comparison of F1 score vs. end-to-end inference latency / zero-loss throughput (single-core) for IoT identification and Application classification serving pipelines. CATO identifies multiple solutions on its Pareto front that dominate those found by traditional optimization techniques, and can achieve significantly better systems performance with a similar or higher F1 scores.

same or better F1 score while also reducing the inference latency. Compared to solutions that need information from the entire connection, CATO can reduce the inference latency by over 3600×, from several minutes to under 0.1 seconds, while simultaneously improving the F1 score. Compared to solutions that use the first 50 packets in the connection, CATO reduces the latency by 817–2000×, and 11–79× for those that use up to the first 10. Since end-to-end inference latency is largely dominated by packet inter-arrival times, this huge improvement can be attributed to CATO's ability to find alternative sets of features using the minimum number of packets necessary without compromising the predictive performance of popular feature selection methods. For example, RFE10 using the first 10 packets achieves an F1 score of 0.970 with an inference latency of 7.9 seconds. However, CATO identifies a different set of features using just the first 3 packets for a better F1 score of 0.979 and an inference latency of 0.1 seconds. We find a similar pattern for Application classification with pipelines optimized with CATO outperforming most baseline methods across both objectives.

After, we compare the predictive performance and classification throughput of solutions found by CATO with those found by the same baseline methods for Application classification. For a realistic assessment, we use live traffic from our campus network, but restrict all experiments to a *single* core to avoid saturating our network's maximum ingress throughput. In an actual deployment scenario, the throughput can be easily scaled up by adding more cores, owing to to the per-core scalability of Retina [144]. Figure 4.5c shows that CATO's solutions outperform most of the baselines across both throughput and F1 score, with the exception of MI10 over the first 10 packets. Despite this, CATO successfully identifies the feature representation that achieves the highest overall F1 score and the one with the highest zero-loss throughput. For a decrease in F1 score from 0.96 to 0.93, CATO can increase the throughput by 37%. Compared to solutions that wait until the end of the connection, CATO can improve the zero-loss throughput by a factor of 1.6–3.7×, and 1.3–2.7× for those that require the first 50 packets while also achieving higher model performance. It is noteworthy that CATO achieves these results after exploring just 50 feature representations out of $2^{67} \times 50 = 7 \times 10^{21}$ potential combinations (67 candidate features, up to a maximum packet depth of 50).

4.4 Conclusions

While it is well-known that *in general* different data representations can both affect model accuracy and introduce variable systems costs, network research has left this area relatively under-explored. In this chapter, we have presented `Traffic Refinery` and `CATO`, two systems that support the design of machine learning models that account for both model accuracy and systems costs. `Traffic Refinery` permits the exploration of both model accuracy and the systems-related costs of machine learning models trained on network traffic representations to make predictions concerning performance and security. `CATO` enables end-to-end optimization of such ML-based traffic analysis pipelines.

Our investigation both constitutes an important re-assessment of previous results and lays the groundwork for new directions in applying machine learning to network traffic modeling and prediction problems. From a scientific perspective, our work explores the robustness of previously published results. From a deployment standpoint, our results also speak to systems-level deployment considerations, and how those considerations might ultimately affect these models in practice, something that has been often overlooked in previous work. Looking ahead, we believe that incorporating these types of deployment costs as a primary model evaluation metric should act as a rubric for evaluating models that rely on machine learning for prediction and inference from network traffic.

Chapter 5

Model Deployment

"LEAF: Navigating Concept Drift in Cellular Networks", ACM CoNEXT 2023 [79]

Takeaways

Operational networks commonly rely on machine learning models for many tasks, including detecting anomalies, inferring application performance, and forecasting demand. Yet, model accuracy can degrade due to *concept drift*, where either the relationships between features and the target to be predicted, or the features themselves change. Mitigating concept drift is an essential part of operationalizing machine learning models in general, but is of particular importance in networking's highly dynamic deployment environments. In this work, we first characterize concept drift in a large cellular network for a major metropolitan area in the United States. We find that concept drift occurs across many important key performance indicators (KPIs), independently of the model, training set size, and time interval—thus necessitating practical approaches to detect, explain, and mitigate it. We then show that frequent model retraining with newly available data is not sufficient to mitigate concept drift, and can even degrade model accuracy further. Finally, we develop a new methodology for concept drift mitigation, Local Error Approximation of Features (`LEAF`). `LEAF` works by detecting drift; explaining the features and time intervals that contribute the most to drift; and mitigating it using forgetting and over-sampling. We evaluate `LEAF` against industry-standard mitigation approaches (notably, periodic retraining) with more than four years of cellular KPI data. Our tests with a major cellular provider in the US show that `LEAF` consistently outperforms periodic and triggered retraining on complex, real-world data while reducing costly retraining operations.

Contents

5.1 Introduction

Network operators rely on machine learning (ML) models to perform many tasks, including anomaly detection [124], performance inference [49] and diagnosis, and forecasting [93, 30, 80]. Unfortunately, deploying and maintaining these models can prove challenging in practice [129]. One significant operational challenge is *concept drift*, whereby changes to the data distribution (*virtual drift*) or its relationship with the target to be predicted (*real concept drift*), deteriorate model performance over time [50, 84]. Previous work in applying ML models to network management tasks has typically trained and evaluated models on fixed, offline datasets [125, 41, 124, 18, 93, 30], demonstrating the ability to predict various network features at fixed points in time on a static dataset. Yet, a model that performs well offline on a single dataset may not in fact perform well in practice, especially over time as characteristics change.

Concept drift is a relatively well-understood phenomenon in ML for other prediction problems (*e.g.*, image and text classification [114, 84, 50]). Yet, mitigating concept drift for networking problems, such as forecasting Key Performance Indicators (KPIs) in a cellular network introduces fundamentally new challenges that make previous approaches from other domains inapplicable. Networks have unique characteristics, such as dynamic signal interference due to environment changes (*e.g.*, weather, seasonality, etc.) [159], which calls for new approaches. In contrast to previous tasks, where the semantics of prediction occurs on a fixed object and characteristics of the features change relatively slowly over time [44, 161, 50], predictions of network characteristics occur continuously and occur within the context of a system that changes over time due to periodicity (*e.g.*, seven-day period of volume), gradual evolution (*e.g.*, the constant addition of capacity by new equipment installations), and exogenous shocks (*e.g.*, a software upgrade or a sudden change in traffic patterns such as the COVID-19 pandemic which resulted in significant changes in user behaviors [85, 81]).

Beyond simply detecting concept drift, operators may also want to interpret why a model has become less accurate and mitigate it. Previous research has developed explanation methods for concept drift in classification problems [63, 153], but prediction in cellular networks is often a regression problem. The absence of distinct classes rules out the direct use of existing methods. In addition, the subtleties of drift in regression make its impact more nuanced.

5.1.1 Model Drift Characterization

As a first step, to quantify the need for solutions for model drift, we explore how trained forecasting models are affected by concept drift in the context of a typical cellular operator forecasting task.

Dataset. We study the extent of model drift relying on more than four years (January 1, 2018 to March 28th, 2022) of daily measurements of LTE cellular network performance indicators collected at the eNodeB-level (evolved NodeBs, or the "base station" in 4G LTE) from a major wireless carrier in the United States. The dataset contains information from 898 eNodeBs from a large city and surrounding metropolitan area (rural, suburban, and urban included) in the United States. The dataset contains 1,084,837 daily eNodeB-level logs. Each log includes 224 Key Performance Indicators (KPIs) collected for a base station on a particular date. KPIs are statistics collected and used by the operator of the network to monitor and assess network performance. The 224 KPIs fall into three categories: (1) resource utilization (*e.g.*, data volume, peak active users, active session time, cell availability rate), (2) access network performance (*e.g.*, throughput, connection establishment success,

congestion, packet loss), and (3) user experience features (*e.g.*, call drop rate, RTP gap duration ratio, abnormal UE releases). Further, some of the KPIs have separate directional measurements.[1]

Forecasting Problem. We focus our study of concept drift in the context of network forecasting. Network forecasting (load, performance, user experience) is an important problem for operators as it sets the foundation to guide infrastructure configuration [68], management [118], and augmentation [150, 107]. We focus on per-eNodeB level KPI forecasting which can be used as a foundation for capacity adjustment, deployment, maintenance, and operation in large cellular networks (not a focus of this work). The nature of this problem is regression with time-series information. Regression models are a better fit than classification because we aim to provide fine-grained forecasting of numerical KPIs that can have wide ranges. In line with multivariate regression modeling used in previous network forecasting applications [150, 107, 68, 118], which uses time-series of KPI histories, we use historical data—*i.e.*, all available KPIs and dates (as features) up to a given day—to forecast one or more target KPIs of interest 180 days in the future. We employ a 180-day forecast window, as operators need this duration for planning and executing long-term network infrastructure augmentation. This 180-day gap also makes it more challenging to explain and mitigate drift.

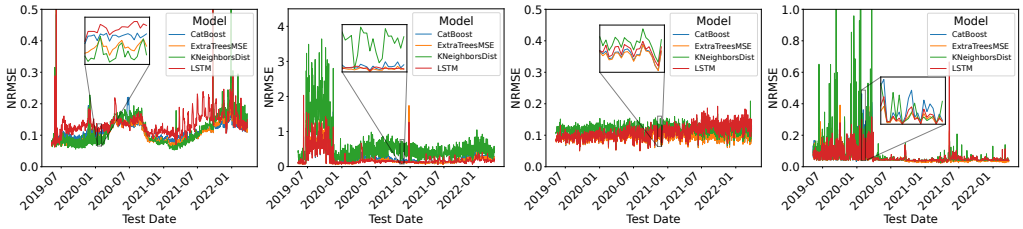
Model Evaluation. To explore the performance of different widely adopted regression techniques, we select *four different families of models*: (1) gradient boosting algorithms like LightGBM, CatBoost, and XGBoost; (2) bagging algorithms such as Random Forest and Extra Trees; (3) distance-based algorithms like KNeighbors; (4) recurrent neural networks such as LSTM. All models either incorporate temporal features (*e.g.*, time stamps, day of the week, month, year), or are time-series models (LSTM). Although it is feasible to fine-tune each model's hyperparameters by hand, we rely on an auto-selection pipeline [13] with the goal of a fair comparison and to make training scalable and efficient. For all experiments, we develop a model for one target KPI per category. As the input of the models, we use a portion of the history of all categorical and numerical KPIs from all eNodeBs up to the date when the model is generated. We generate a single model for the entire network, *i.e.*, we create a model capable of forecasting values for samples collected from each base station.

Consistent Drift Occurs, Independent of Model, Target KPI, and Training Set Size. We demonstrate concept drift in a large cellular network comparing different KPIs, model families, training set sizes, and periods across many regression models and tasks. Concept drift occurs consistently and independently of both the size and period of the training set. Ultimately, the diverse, longitudinal nature of the dataset used presents a challenging concept drift problem.

As an example,[2] Figure 5.1 presents the concept drift across time for the three categories of KPIs. Overall, drift patterns are quite unique for each class, and they vary in two aspects. First, deviations in Normalized Root Mean Squared Error (NRMSE) occur at different periods of time. For instance, in Figure 5.1a, the NRMSE of downlink volume, specifically in the CatBoost model, experiences a substantial increase from 0.102 to 0.136 in April 2020 due to the COVID-19 lockdown—a clear example of sudden concept drift. However, it reverts back to more normal values, achieving an average of 0.089 by October 2020. From March 2021, the NRMSE starts to rise once more, peaking at 0.145 in January 2022. For the prediction of peak users, Figure 5.1b demonstrates that July 2019 to November 2019 are more challenging to predict (0.789 NRMSE for CatBoost), because of lost data.

[1] We release a normalized version of the dataset spanning from from May 2019 to May 2020: https://forms.gle/g5pbB5qRHeBsEmZJ6.

[2] Please refer to the full paper for the complete characterization [79].



(a) Volume. (b) Peak active UEs. (c) Throughput. (d) Gap duration ratio.

Figure 5.1: Drift of different models for KPIs of interest. Inset figures exhibit a 3-week view (all starting from Sunday) of NRMSE for the box-selected period. Some data is lost between July, 2019 and January, 2020 for Peak active UEs. Note that the y-axes are scaled to different range to accommodate larger errors in Fig.5.1b, 5.1d.

Moreover, short-lived, abrupt increases in error are more frequent than other KPIs, due to the burstiness of GDR. Second, the high-frequency components have different patterns for KPIs. Using signal processing techniques like STFT, we found no clear weekly pattern in the NRMSE of CDR and GDR, while other KPIs exhibit such patterns, as the 3-week insets show.

Naïve Retraining in Practice. In operational networks, a common approach to counteract potential concept drift is to retrain models regularly. Retraining using the latest data is often considered an effective way to deal with concept drift. Many existing solutions [65, 138] adopt this approach, which outperforms recent dedicated drift mitigation methods [89, 156]. To understand the effectiveness of this approach, we retrain a number of different models using different retraining frequencies. For this experiment, we use a training set of 14 days (the best performing size in our characterization) to forecast traffic volume 180 days in the future, using a CatBoost model. Given a retrain frequency N, a model is retrained using the latest 14-day data. It is evaluated using the NRMSE for the next N days and is then replaced every N days. Somewhat counter to conventional practice, we find that simply retraining the model at regular intervals is insufficient to efficiently combating concept drift for a diverse, longitudinal dataset. Naïve retraining is either less effective, or is effective but inefficient, requiring frequent retraining.

Intuitively, a key reason that naïve retraining may not work is that it is triggered at regular intervals, even though drift occurrences are irregular. It does not take into account when, where, and why drift is occurring. Thus, at times retraining is not necessary, or it is planned before drift actually occurs. This strategy ignores the fact that models trained on more recent periods do not necessarily result in better performance. Further, complete data replacement neglects finer-grained error information across samples, throwing away useful samples from the past. Overall, we conclude that although retraining is essential, naïvely performing it at regular intervals is not sufficient for efficient and explainable drift mitigation. It works best at high retraining frequencies and requires specific tests across different KPIs to at best tuning its performance. Both are challenging when run at scale in operational networks. Also, naïve retraining neglects finer-grained temporal error information across samples and thus loses explainability. Based on the need to develop a more generally applicable solution, the rest of the Chapter presents `LEAF`, a framework that detects, explains, and mitigates concept drift in cellular networks.

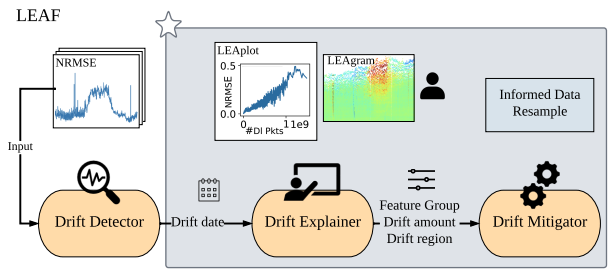


Figure 5.2: The LEAF framework that detects, explains, and mitigates concept drift. Our main contributions are in the gray box marked with a star.

5.2 LEAF

To tackle the model drift problem in cellular networks, we introduce LEAF, a framework for drift detection, explanation, and mitigation. LEAF leverages explainable AI methods to provide us with insights about concept drift and mitigate it effectively.

5.2.1 Navigating Concept Drift in Cellular Networks

LEAF is designed to work with any supervised regression model and provide explanations for black box models. Based on the explanations provided, targeted mitigation strategies are applied to compensate for concept drift. Existing solutions face two limitations: (1) previous concept drift explanation and mitigation techniques are limited to classification problems (*e.g.*, [63, 153]); (2) concept drift mitigation is often coarse-grained, only focused on the global performance metrics (*e.g.*, [19]).

LEAF overcomes these limitations by implementing a pipeline of three components: (1) a drift *detector*; (2) a set of tools to *explain* drift for features; and (3) a drift *mitigator*. LEAF works in a black-box manner, only requiring access to the previously used training set data, new data as it arrives, and the generated model. Figure 5.2 shows the three steps in LEAF's pipeline:

1. **Drift Detector:** The detector ingests the outputs of the model in the form of NRMSE time-series to determine whether drift is occurring. The detector applies the well-known Kolmogorov-Smirnov Windowing (KSWIN) method [105, 139] on the time-series to identify a change in the distribution of the output error, providing an indicator of whether drift is occurring. Drift detection is critical but we mainly focus on drift explanation and informed mitigation in this work, as there is significant prior work on detection.

2. **Drift Explainer:** The explainer is triggered at the time instances at which drift is detected. We design LEAF's explainer around the goal of characterizing errors of a regression model simply based on the model input and output. We extend global model-agnostic explanation methods [47, 15, 16] to identify and visualize the effect that different features have on prediction errors of black-box regression models. To do so, the explainer determines representative features that contribute to drift, uses Local Error Approximation (LEA) to characterize the drift, and offers guidance for model compensation. The process generates LEAplots and LEAgrams, which provide insights for operators to understand drift events.

3. **Drift Mitigator:** Based on the error distribution and statistical patterns, the mitigator *automatically* forgets previous data, and performs informed replacement by sampling/over-sampling targeted regions. Our insight is that while the global error metrics provide a good measure of the performance over time, *the distribution of local errors across samples at each given time instance may be uneven.* Using this intuition, `LEAF`'s mitigator better compensates for occurring drift.

As shown, at the core of the LEAF framework is the drift explainer, which relies on LEA to provide insights into the drift. We detail the LEA technique next.

Local error approximation (LEA). To analyze the error of a model on any corresponding dataset, we use selected representative features R to inspect based on feature importance and mutual information. For each feature f_i in R, we group samples based on the value of the representative feature into N bins (*i.e.*, quantiles $\{q_j\}_{j=1}^{N}$). The higher N is, the finer the granularity of local errors that can be observed. Next, a specified error metric (NRMSE by default) is computed for samples within each bin. We compile these N measurements into a vector, representing the error distribution of each quantile. This technique approximates local errors over the range of the most sensitive and representative features for a specific model and dataset:

$$LEA(N, M) = \{\mathbf{e}_i | \mathbf{e}_{i,j} = Err(Q(f_i, q_j, (\mathbf{X}, \mathbf{y}))), f_i \in R\} \tag{5.1}$$

Here, $LEA(N, M)$ is the list of local error approximations for a given feature matrix $\mathbf{X}$, target vector $\mathbf{y}$,[3] number of bins N, and model M. The function $Q(f_i, q, (\mathbf{X}, \mathbf{y}))$ returns samples from quantile q_j of feature f_i, with $\mathbf{e}_i$ calculated for each representative feature from the set $R(\rho)$. LEA characterizes the extent of drift and focuses targeted drift mitigation, providing the foundation of LEAF.

5.2.2 Main Results

We evaluate how the `LEAF` framework mitigates drift compared to existing techniques as baselines. We present how different mitigation schemes improve end-to-end model performance.[4]

End-to-End Comparison Across Mitigation Schemes. We compare the average NRMSE over the duration of the dataset against three baselines: ADaptive WINdowing (ADWIN) [22], which uses a variable-length window to detect and adjust for changes; the naïve retraining scheme that retrains the model every 30 or 90 days (we choose these two frequencies because they present the best performance versus cost tradeoff); and a triggered retraining scheme, which only utilizes the KSWIN detections, *i.e.*, retrain the model using the latest available data whenever drift is detected. To keep the evaluation fair, we use the same amount (14 days) of data for each retrain, which also controls the amount of time needed for a single retrain across schemes. The mitigation effectiveness is compared against a static model (trained on 14 days of data before July 1st, 2018) for each target KPI.

We investigate the trade-off between $\Delta\overline{NRMSE}$, the average distance between the error for a mitigated model against a static model, and the number of retrains required for selected schemes and present in Table 5.1 a summary of the results. Such trade-off provides insights not only on the performance of each mitigation scheme, but also on its applicability in practice in an operational network where each retrain operation might come at a cost. We use the number of retrains as a proxy to the system cost, as the operational overhead

[3] We suppress dependence on the data for ease of notation.
[4] Please refer to the full paper for the complete evaluation [79].

Table 5.1: Effectiveness of mitigation schemes measured in $\Delta\overline{NRMSE}$ and $\#Retrains$. We include models from different model families over a variety of KPIs. The scheme with the highest performance is shaded in gray.

Model	**KPIs**	$\Delta\overline{NRMSE}$ **(#Retrains) of Mitigation Schemes**					
		Naïve_{30}		Triggered		LEAF	
CatBoost	DVol	−29.62%	(39)	−31.80%	(27)	−32.67%	(28)
	PU	−44.88%	(39)	−35.06%	(25)	−46.59%	(35)
	DTP	−20.02%	(39)	−23.84%	(28)	−24.30%	(31)
	REst	−35.41%	(39)	−38.38%	(25)	−38.44%	(31)
	CDR	2.35%	(39)	−4.21%	(17)	−3.63%	(9)
	GDR	3.37%	(39)	44.56%	(17)	−6.24%	(19)
ExtraTrees	DVol	−24.77%	(39)	−28.17%	(32)	−30.64%	(32)
	PU	−44.26%	(39)	−50.76%	(26)	−45.83%	(27)
	DTP	−18.13%	(39)	−21.63%	(32)	−22.59%	(23)
	REst	−31.95%	(39)	−34.29%	(22)	−36.13%	(29)
	CDR	2.10%	(39)	8.08%	(20)	−0.20%	(11)
	GDR	−0.58%	(39)	33.67%	(17)	−14.26%	(19)
LSTM	DVol	0.54%	(39)	14.12%	(21)	2.67%	(19)
	PU	37.11%	(39)	3.76%	(25)	−20.48%	(18)
	DTP	17.08%	(39)	−0.82%	(14)	−37.13%	(20)
	REst	6.78%	(39)	5.78%	(27)	4.21%	(26)
	CDR	−33.22%	(39)	−21.39%	(10)	−71.52%	(11)
	GDR	0.41%	(39)	−8.58%	(14)	−16.29%	(13)
KNeighbors	DVol	−8.26%	(39)	−4.11%	(16)	−4.47%	(24)
	PU	−34.09%	(39)	−37.99%	(16)	−18.11%	(20)
	DTP	−4.73%	(39)	−4.03%	(18)	−1.53%	(22)
	REst	−26.69%	(39)	−25.86%	(25)	−22.10%	(16)
	CDR	9.44%	(39)	7.35%	(11)	4.69%	(12)
	GDR	−8.13%	(39)	−23.40%	(19)	−6.12%	(13)

of implementing LEAF is minimal, especially when contrasted against the time-consuming process of retraining models. For example, running a CatBoost model on a 14-day training set averages around 113.688 seconds, whereas employing LEAF on the same volume of data takes merely 0.657 seconds—just about 0.578% of the time needed for CatBoost. Our goal is to find the scheme that achieves the best mitigation effectiveness first, while balancing the alternative goal of few retrains.

Despite tuning its confidence parameters and providing sufficient window size, ADWIN struggles to effectively mitigate NMRSE induced by drift. For CatBoost, it is only triggered 7 times, achieving a −9.75% mitigation effectiveness on DVol predictions. It even fails to detect any drift in the error time series of highly dispersed KPIs like CDR and GDR. This is largely due to the cyclical patterns and noise irregularities inherent in our real-world data streams.

Naïve retraining every 30 days requires the highest number (39 for all KPIs) of retrains, yet its mitigation effectiveness never outperforms LEAF. Naïve retraining every 90 days requires fewer retrains (13). However, the mitigation effectiveness is frequently inferior to LEAF's, except for the CDR KPI. The triggered mitigation scheme rarely outperforms other schemes on either metric. It also has exponential errors for KPIs like GDR (44.56% increase on $\Delta\overline{NRMSE}$), making it less practical among the schemes since it does not guarantee performance improvements after mitigation.

Finally, LEAF consistently outperforms all baseline schemes considering the trade-off. Due to its error explanation and informed mitigation approach, LEAF already exceeds the

performance of other methods across KPIs (*e.g.*,−46.69% for PU, −38.44% for REst), except for CDR, even when only using one representative feature. Compared to triggered retraining, `LEAF` can mitigate more errors, with a similar or slightly higher number of retrains (*e.g.*, all but CDR). For CDR, `LEAF` is able to mitigate a similar amount of errors with 30.8% fewer retrains.

5.3 Conclusions

An important step in deploying machine learning models for networking tasks in practice is dealing with concept drift. Although this phenomenon has been explored in other contexts, it has received limited attention in the networking domain. To address it, our work characterizes drift patterns across multiple models and KPIs and has developed, presented, and evaluated `LEAF`, a framework to detect, explain, and mitigate drift for black-box models applied to cellular demand forecasting. The `LEAF` framework employs explainable AI and informed mitigation. Our results based on more than four years of KPI data from a large cellular network show that `LEAF` consistently outperforms both periodic and triggered retraining, while reducing the cost of retraining.

In future work, we plan to explore how the `LEAF` framework could also be applied to other network management problems modeled as regression-based predictions, as well as explore extensions to classification problems. Another area for exploration lies in the use of ensemble models across modeling frameworks in `LEAF`, as they could potentially enhance resilience against concept drift. Finally, the evaluation of `LEAF` to date has been conducted on fully labeled datasets, due to the forecasting problem nature, where the ground truth is available once the future is observed; thus, a promising direction could be to improve `LEAF` to cope with semi-supervised or unsupervised regressors.

Chapter 6

Conclusions and Future Work

Takeaways

The work presented in this manuscript presents solutions aimed at the development and deployment of machine learning models on network traffic that are both accurate, as well as effective when deployed in practice. However, this problem is vast and complex, and this manuscript does not explore all dimensions of this problem. In this regard, we plan to explore two new research directions that we believe are key in further moving forward this topic: first, we will attack the challenges that limit the deployment of machine learning network operations at scale, *i.e.*, over an entire network; second, we will explore the role that novel generative AI models will have in enhancing networked systems and inference models.

Contents

6.1 Conclusions

Over the past 15 years, machine learning models have become increasingly integral to both network performance and security tasks. Yet, by and large, existing approaches have aimed to maximize model prediction performance (*e.g.*, accuracy), without consideration of deployment costs or constraints that arise from deploying these models in production settings. The work presented in this manuscript tackles various challenges that close the gap between accurate machine learning models for network operations and their deployability in practice. However, this problem is vast and complex, and this manuscript does not explore all dimensions of this problem. For our future work, we plan to continue expanding on the directions opened by this manuscript. In this regard, we plan to explore two new research directions: (1) Attack that challenges that limit the deployment of machine learning network operations at scale, *i.e.*, over an entire network; and (2) Explore the role that novel generative AI models will have in enhancing networked systems and inference models.

6.2 ML at Network Operations Scale

Network management typically relies on machine learning to make predictions pertaining to both network performance and security. As discussed throughout this manuscript, existing approaches by and large aim to maximize model prediction performance (*e.g.*, accuracy), without consideration of deployment costs or constraints that arise from deploying these models in production settings. Our work tackled various challenges to close this gap. However, most existing models still operate on timescales that do not correspond to operational decisions taken by network operators. When considering that the response to prediction outcomes often involves a human in the loop, different tradeoffs between accuracy and timeliness of prediction may be important. For example, models that can produce continuous predictions, even if occasionally inaccurate, may be more useful than models that produce accurate predictions only at periodic, discrete points in time.

To attack this challenge, we plan to build systems and models tailored around the time scales at which network management occurs, maximizing the accuracy of the output of the models rather than the single inference instance. We plan to explore a new paradigm for machine learning models for networking, where latency is paramount and decisions are ultimately based on aggregate streams of data where any single prediction may be inaccurate, but the aggregate stream of predictions can still yield valuable information for decision-making. To design models that can operate under these conditions, we will explore the use of machine learning techniques that leverage both the spatial and temporal correlations that exist in network data. To achieve this, we will explore two main directions: First, we will investigate how to design models that efficiently serve models that best fit the current network conditions, maximizing at any time instance the accuracy vs systems cost tradeoff. Second, we will explore how to sample and aggregate data to reduce the amount of data that we need to process, based on the insight that predictions that occur close together in time or space are likely to exhibit some level of similarity.

Dynamic Model Serving. Nowadays, ML techniques have become a common solution used to solve a variety of network management tasks. To meet different tradeoffs across model performance and system constraints, different models are available for each task. For example, `CATO` [145] identifies the set of models that are Pareto optimal across both accuracy and systems costs. However, the deployment of these models in real-world settings is still a challenge. The users of these models have to decide a priori which model might best fit their deployment, configuring their measurement system accordingly [145, 24]. Yet,

network traffic is inherently dynamic and varies throughout the day due to continuous shifts in usage, making it hard, if not impossible, to select an optimal configuration that can work throughout these changes. Further, substituting such configurations requires to first observe that the system is not capable of processing the traffic, and then manually change the configuration causing further loss due to reboot times.

We argue that the best vantage point to understand a system's ability to select the optimal model to serve is *the system itself*. Instead of selecting the optimal candidate model based on offline information, systems should adapt based on up-to-date information of the traffic observed as well as on the system's ability to extract the requested features for a specific traffic load. Towards this, we plan to design a monitoring system that takes as input a set of candidate ML models, and the features they require as input, and adaptively selects the better fitting configuration as a function of the network and the system conditions. Our initial results [62, 59] show that systems that adapts in real time can be used to simplify the deployment of ML models in concrete applications, enabling network administrators to more easily rely on machine learning solutions to drive their networks. This early work presents several interesting challenges to overcome and future work directions to explore. One key challenge involves expanding the array of feature sets to satisfy the requirements of various machine learning models. Furthermore, exploring more complex profiling metrics beyond the scope of packet loss alone promises to be an interesting avenue of research. With the inclusion of these metrics, our system will be more proficient in extracting the most suitable feature set for the current network context, thus ensuring the choice of the best model.

Data Sampling and Aggregation. The ability to process and analyze network traffic data is crucial for network operators to make informed decisions about network management. Thanks to recent advances in model development and algorithms, inference techniques can now provide fine grained information about single network flows or application sessions, *e.g.*, extract real-time resolution of video streaming sessions solely using encrypted traffic [25]. Unfortunately, the ever increasing amounts of network traffic make it unfeasible for modern traffic analytics architectures to appropriately scale and process all incoming flows. Historically, network operators relied on sampling techniques to reduce the amount of data that needs to be processed. However, these techniques are often suboptimal, as they negatively impact the quality of the model output [24]. Still, limited by processing bottlenecks and to cope with the amounts of traffic to process, measurement infrastructure has to either select a small subset of flows or reduce the quality of the models served.

To address this challenge, we plan to explore new ways to sample flows to process and later aggregate data to make better informed decisions in network operations. First, we plan to study how to enable measurement infrastructure to select the most informative flows to process, *i.e.*, develop admission control techniques tailored for network analytics. Admission control techniques can be crucial for ensuring an effective real-time monitoring by selecting flows of interest from the network's traffic. In this scenario, a controller becomes in charge of evaluating the relevancy of each new incoming flow and decide whether to admit it to the processing pipeline. In case a flow is admitted, compute and memory resources are allocated to compute statistics used by the ML model for the treatment of the flow. Such admission decisions can be taken based on the intuition that predictions that occur close to each other in time or space are likely to exhibit some level of similarity. Our initial results in developing a constrained Markovian framework for the decentralized admission control of varied information flows shows that it is indeed attainable to achieve an optimal admission policy [45].

Second, we plan to explore the use of statistical techniques to aggregate point measure-

ments over a geography to obtain large scale insights on an operator network. Even with smarter sampling techniques, measurements might end up concentrated irregularly over space, rending necessary to understand how these sparsely-selected samples can be used to make generalizations about the entire network. A key challenge towards this goal is to identify the extent of noise associated with a single measurement. It is therefore necessary to also understand the extent and the spatial granularity up to which these measurements can be de-noised, and aggregated to form conclusions about the performance of the network over a specific geography. Our initial work on spatial interpretation of measurements [122] suggests that a combination of statistical techniques can achieve gains in similarity score over traditional methods that solely rely on aggregates over raw measurement values for performance summarization.

6.3 The Role of Generative Models

Throughout this manuscript, we discussed how modern network operations increasingly rely on machine learning (ML) and artificial intelligence (AI) to solve a variety of management problems. In particular, we focused numerous times on a specific use case: the inference of services quality of service, *e.g.*, video streaming quality. Through the years, a myriad of proposals have emerged to provide a solution to map service quality to network traffic and the resulting models have proved to be both accurate and reliable in deployment settings (*e.g.*, [25, 90, 40, 91]). However, while the developed QoS inference models perform well in the environments they are tested on, these solutions have yet to see wide adoption outside of the academic space.

We identify two open questions at the root of this lack of adoption that have yet to be addressed: First, existing models do not work in network environments where abundant training data is not available. Existing models are commonly trained on network data, *i.e.*, packet traces, collected in the sole environments accessible to researchers: controlled laboratories. This translates in the key challenge of collecting representative data to train models that generalize well to networks different from the ones where they were initially trained and tested on. Second, inference models produce metrics that are hard to action upon. Existing models are now capable of producing metrics that more closely relate to the actual application quality. For example, in the case of video quality inference, models provide operators information that relates to actual application performance (*e.g.*, video resolution instead of average flow throughput). Yet, while these metrics well correlate to experienced service quality, they do not provide an easy-to-use input for network management tasks (*e.g.*, traffic engineering).

Potential research directions. Fortunately, recent breakthroughs in AI-based generation techniques offer a promising avenue to overcome these challenges. New model architectures (*e.g.*, transformer [140] and diffusion models [128, 54]) are capable of capturing complex structures, broad patterns, and detailed dependencies from unlabeled data. Thanks to these capabilities, they have been successfully used to solve a number of complex tasks across a variety of domains, particularly in natural language processing [38] and computer vision [42]. Yet, the potential of generative AI techniques remains mostly underexplored in the networking context, particularly in aiding to move quality inference-based network management to the next level. To this end, we aim to study the role of generative AI techniques in solving the unanswered problems of (1) scarcity of data to train quality inference models and (2) translate inference models' output into actions. We plan to tap into the unique advantages offered by generative AI models to develop new tools aimed at

the deployment end-to-end quality inference models.

Generative Models for Data Augmentation. A central impediment when training network-focused ML models is the scarcity of labeled network datasets, as their collection and sharing are often associated with high costs and privacy concerns, particularly when data is collected from real-world networks. Moreover, existing public datasets rarely receive updates, making them static and unable to reflect evolving network behaviors. Further, they often lack ground truth for application quality behavior. Unfortunately, these limitations hinder the ability to train robust ML models that accurately reflect evolving real-world network conditions.

In Chapter 3 we discussed our `NetDiffusion` [61] solution, a framework that employs controlled text-to-image diffusion models [128] to generate synthetic raw network traffic that complies with transport and network layer protocol rules. Our results show `NetDiffusion`'s ability to generate high-fidelity packet traces that resemble real-world traffic. However, several limitations remain open for exploration, especially when targeting quality inference ML models. First, diffusion models depend on a constant image size for both training and generation, constraining synthetic traces to a fixed length. Second, while diffusion models are highly expressive, resulting in synthetic traces that more accurately mimic real network dynamics, they produce noisy outputs that can compromise the correctness of generated traces and do not account for inter packet-timings. Third, diffusion models are incapable of capturing complex correlations between traces and their associated metadata, *i.e.*, the application quality associated with a network trace. To address these challenges, we aim to explore generative AI techniques that are more suited for the generation of complex sequential data. For example, state-space models (SSMs) (*e.g.*, Mamba [51]) have shown to be effective in generating sequential data like text, suggesting their potential for network traffic generation. Unfortunately, to apply them to networking data, there remain open key challenges to address, including appropriate packet capture tokenization and maintaining long contexts for generating meaningful flows, generating semantically meaningful traffic "payloads" (*i.e.*, data within each packet), and generate multi-dimensional time series of collection of flows and associated metadata. Our initial results [33] on applying SSMs to network traffic generation are promising, showing that SSMs better capture complex intra-packet dependencies with additionally higher fidelity, while requiring lower training and inference resource footprint than existing, comparable approaches.

Generative Models for Automated Operations. Network operators need to be reactive in case of problems like outages or sudden QoS drops, and change their configuration to adapt to these events. As operators do not want to push wrong configurations that could increase the blast radius of an outage, they emulate possible failures, measure the potential impact, change the configuration, and evaluate whether the submitted changes answer their objective and mitigate detected problems. Most of this process is still largely human-driven, thus remaining error prone and inefficient. One of the core reason for this is that existing network verifiers (*e.g.*, [115, 135]) are based on synthesis techniques that typically require extensive human intervention.

We aim to explore a more general approach that targets recent advances on LLMs as the baseline to address the limitations of traditional ML/NLP/synthesis-based verification techniques. Specifically, as LLMs are trained on massive amounts of unlabeled public data, including configuration data, they have the capability of performing configuration validation without the need for manual rule engineering. Ciri [76] has been recently presented as a first solution to address this challenge by integrating different LLMs and verifying the output for potential model hallucinations. Yet, the simplistic approach used (*i.e.*, simply feeding a full configuration to the model) is solely capable of capturing syntactical errors, as more

complex correctness errors are dispersed in distant positions within the configuration file, rendering the model ineffective in their detection. To overcome this limitation, we plan to exploit the natural tree-like structure of configuration files to provide a properly formatted input to the model. Exploiting this structure, terms re-utilized at different distances within the text will be easily identifiable as recurrent and connected, enabling the model to better understand their contextual meaning.

Bibliography

[1] User Datagram Protocol. RFC 768, Aug. 1980. (Cited on page 21.)

[2] Internet Control Message Protocol. RFC 792, Sept. 1981. (Cited on page 21.)

[3] Internet Protocol. RFC 791, Sept. 1981. (Cited on page 21.)

[4] Transmission Control Protocol. RFC 793, Sept. 1981. (Cited on page 21.)

[5] An Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware. RFC 826, Nov. 1982. (Cited on page 21.)

[6] Deepfield. https://www.nokia.com/networks/solutions/deepfield/, 2019. (Cited on page 4.)

[7] Kentik. https://kentik.com/, 2019. (Cited on page 4.)

[8] Go language. https://golang.org/, 2020. (Cited on page 27.)

[9] Traffic Refinery. https://github.com/traffic-refinery/traffic-refinery, 2021. (Cited on page 27.)

[10] A. Aaron, Z. Li, M. Manohara, J. De Cock, and D. Ronca. Per-title encode optimization. *The Netflix Techblog*, 2015. (Cited on page 9.)

[11] M. Abbasi, A. Shahraki, and A. Taherkordi. Deep learning for network traffic monitoring and analysis (NTMA): A survey. In *Computer Communications*, 2021. (Cited on page 26.)

[12] S. Abt and H. Baier. Are we missing labels? a study of the availability of ground-truth in network security research. In *2014 third international workshop on building analysis datasets and gathering experience returns for security (badgers)*, pages 40–55. IEEE, 2014. (Cited on page 18.)

[13] A. AI. *AutoGluon: AutoML for Text, Image, and Tabular Data*, accessed July, 2021. https://auto.gluon.ai/stable/index.html. (Cited on page 37.)

[14] 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. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 5(1):1–26, 2021. (Cited on page 18.)

[15] D. W. Apley and J. Zhu. Visualizing the effects of predictor variables in black box supervised learning models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 82(4):1059–1086, 2020. (Cited on page 39.)

[16] B. Arzani, K. Hsieh, and H. Chen. Interpretable feedback for automl and a proposal for domain-customized automl for networking. In *Proceedings of the Twentieth ACM Workshop on Hot Topics in Networks*, pages 53–60, 2021. (Cited on page 39.)

[17] G. Association. Network Management of Encrypted Traffic: Version 1.0. https://www.gsma.com/newsroom/wp-content/uploads/WWG-04-v1-0.pdf, Feb. 2015. (Cited on page 8.)

[18] S. Ayoubi, N. Limam, M. A. Salahuddin, N. Shahriar, R. Boutaba, F. Estrada-Solano, and O. M. Caicedo. Machine learning for cognitive network management. *IEEE Communications Magazine*, 56(1):158–165, 2018. (Cited on page 36.)

[19] S. H. Bach and M. A. Maloof. Paired learners for concept drift. In *2008 Eighth IEEE International Conference on Data Mining*, pages 23–32. IEEE, 2008. (Cited on page 39.)

[20] J. Beale, A. Orebaugh, and G. Ramirez. *Wireshark & Ethereal network protocol analyzer toolkit*. Elsevier, 2006. (Cited on pages 18 and 21.)

[21] L. Bernaille, R. Teixeira, and K. Salamatian. Early application identification. In *Proceedings of the 2006 ACM CoNEXT conference*, pages 1–12, 2006. (Cited on pages 18 and 26.)

[22] A. Bifet and R. Gavalda. Learning from time-changing data with adaptive windowing. In *Proceedings of the 2007 SIAM international conference on data mining*, pages 443–448. SIAM, 2007. (Cited on page 40.)

[23] R. Boutaba, M. A. Salahuddin, N. Limam, S. Ayoubi, N. Shahriar, F. Estrada-Solano, and O. M. Caicedo. A comprehensive survey on machine learning for networking: evolution, applications and research opportunities. *Journal of Internet Services and Applications*, 9(1):16, 2018. (Cited on pages 2 and 26.)

[24] F. Bronzino, P. Schmitt, S. Ayoubi, H. Kim, R. Teixeira, and N. Feamster. Traffic refinery: Cost-aware data representation for machine learning on network traffic. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 2021. (Cited on pages 3, 25, 28, 44 and 45.)

[25] F. Bronzino, P. Schmitt, S. Ayoubi, G. Martins, R. Teixeira, and N. Feamster. Inferring streaming video quality from encrypted traffic: Practical models and deployment experience. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 2019. (Cited on pages 4, 7, 26, 28, 45 and 46.)

[26] M. Brown. Traffic control howto. `http://tldp.org/HOWTO/Traffic-Control-HOWTO/index.html`, 2006. (Cited on page 9.)

[27] Z. Bu, B. Zhou, P. Cheng, K. Zhang, and Z.-H. Ling. Encrypted network traffic classification using deep and parallel network-in-network models. *IEEE Access*, 8:132950–132959, 2020. (Cited on page 18.)

[28] C. Busse-Grawitz, R. Meier, A. Dietmüller, T. Bühler, and L. Vanbever. pforest: In-network inference with random forests. *arXiv preprint arXiv:1909.05680v2*, 2022. (Cited on page 26.)

[29] Z. Chen, K. He, J. Li, and Y. Geng. Seq2img: A sequence-to-image based approach towards ip traffic classification using convolutional neural networks. In *2017 IEEE International conference on big data (big data)*, pages 1271–1276. IEEE, 2017. (Cited on page 19.)

[30] S. Chinchali, P. Hu, T. Chu, M. Sharma, M. Bansal, R. Misra, M. Pavone, and S. Katti. Cellular network traffic scheduling with deep reinforcement learning. In *Thirty-second AAAI conference on artificial intelligence*, 2018. (Cited on page 36.)

[31] Chrome webRequest API. https://developer.chrome.com/extensions/webRequest, 2018. (Cited on page 9.)

[32] ChromeDriver - WebDriver for Chrome. https://sites.google.com/a/chromium.org/chromedriver/, 2018. (Cited on page 9.)

[33] A. Chu, X. Jiang, S. Liu, A. Bhagoji, F. Bronzino, P. Schmitt, and N. Feamster. Feasibility of state space models for network traffic generation. *arXiv preprint arXiv:2406.02784*, 2024. (Cited on page 47.)

[34] K. C. Claffy. Internet traffic characterization. 1995. (Cited on page 18.)

[35] S. Cui, B. Jiang, Z. Cai, Z. Lu, S. Liu, and J. Liu. A session-packets-based encrypted traffic classification using capsule neural networks. In *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pages 429–436. IEEE, 2019. (Cited on page 18.)

[36] F. De Keersmaeker, Y. Cao, G. K. Ndonda, and R. Sadre. A survey of public iot datasets for network security research. *IEEE Communications Surveys & Tutorials*, 2023. (Cited on page 18.)

[37] T. Developers. Tcpreplay. https://tcpreplay.appneta.com/, 2023. (Cited on page 18.)

[38] J. Devlin et al. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018. (Cited on page 46.)

[39] C. Dewes, A. Wichmann, and A. Feldmann. An analysis of internet chat systems. In *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, pages 51–64, 2003. (Cited on page 18.)

[40] G. Dimopoulos, I. Leontiadis, P. Barlet-Ros, and K. Papagiannaki. Measuring video QoE from encrypted traffic. In *Proceedings of the 2016 Internet Measurement Conference*, pages 513–526. ACM, 2016. (Cited on pages 2, 8, 10, 11, 12 and 46.)

[41] B. Dong and X. Wang. Comparison deep learning method to traditional methods using for network intrusion detection. In *2016 8th IEEE International Conference on Communication Software and Networks (ICCSN)*, pages 581–585. IEEE, 2016. (Cited on page 36.)

[42] A. Dosovitskiy et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020. (Cited on page 46.)

[43] K. Dyer. How encryption threatens mobile operators, and what they can do about it. http://the-mobile-network.com/2015/01/how-encryption-threatens-mobile-operators-and-what-they-can-do-about-it/, Jan. 2015. (Cited on page 8.)

[44] F. Fdez-Riverola, E. L. Iglesias, F. Díaz, J. R. Méndez, and J. M. Corchado. Applying lazy learning algorithms to tackle concept drift in spam filtering. *Expert Systems with Applications*, 33(1):36–48, 2007. (Cited on page 36.)

[45] A. Fox, F. De Pellegrini, F. Faticanti, E. Altman, and F. Bronzino. Optimal flow admission control in edge computing via safe reinforcement learning. *arXiv preprint arXiv:2404.05564*, 2024. (Cited on page 45.)

[46] P. I. Frazier. A tutorial on Bayesian optimization. *arXiv preprint arXiv:1807.02811*, 2018. (Cited on page 31.)

[47] J. H. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001. (Cited on page 39.)

[48] C. Fu, Q. Li, M. Shen, and K. Xu. Realtime robust malicious traffic detection via frequency domain analysis. In *ACM SIGSAC Conference on Computer and Communication Security (CCS)*, 2021. (Cited on page 26.)

[49] Futuriom. *Verizon Applies Machine Learning to Operations*, accessed August, 2021. https://www.futuriom.com/articles/news/verizon-applies-machine-learning-to-operations/2018/08. (Cited on page 36.)

[50] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia. A survey on concept drift adaptation. *ACM computing surveys (CSUR)*, 46(4):1–37, 2014. (Cited on page 36.)

[51] A. Gu and T. Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023. (Cited on page 47.)

[52] C. Gutterman, K. Guo, S. Arora, X. Wang, L. Wu, E. Katz-Bassett, and G. Zussman. Requet: Real-time qoe detection for encrypted youtube traffic. In *ACM Conference on Multimedia Systems*, MMSys '19, Amherst, MA, USA, February 2019. (Cited on page 4.)

[53] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik. Gene selection for cancer classification using support vector machines. In *Machine Learning*, 2002. (Cited on page 32.)

[54] J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020. (Cited on pages 20 and 46.)

[55] J. Ho, C. Saharia, W. Chan, D. J. Fleet, M. Norouzi, and T. Salimans. Cascaded diffusion models for high fidelity image generation. *The Journal of Machine Learning Research*, 23(1):2249–2281, 2022. (Cited on page 20.)

[56] J. Holland, P. Schmitt, N. Feamster, and P. Mittal. New directions in automated traffic analysis. In *ACM Conference on Computer and Communication Security (CCS)*, 2021. (Cited on pages 19 and 26.)

[57] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021. (Cited on page 20.)

[58] T. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. In *ACM SIGCOMM*, Chicago, IL, aug 2014. (Cited on page 8.)

[59] J. Hugon, G. Nodet, A. Busson, and F. Bronzino. Towards adaptive ml traffic processing systems. *Proceedings of the ACM CoNEXT Student Workshop 2023*, 2023. (Cited on page 45.)

[60] C. Hvarfner, D. Stoll, A. Souza, M. Lindauer, F. Hutter, and L. Nardi. πBO: Augmenting acquisition functions with user beliefs for Bayesian optimization. In *International Conference on Learning Representations (ICLR)*, 2022. (Cited on page 31.)

[61] X. Jiang, S. Liu, A. Gember-Jacobson, A. Nitin Bhagoji, P. Schmitt, F. Bronzino, and N. Feamster. Netdiffusion: Network data augmentation through protocol-constrained traffic generation. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 2024. (Cited on pages 17, 21 and 47.)

[62] X. Jiang, S. Liu, S. Naama, F. Bronzino, P. Schmitt, and N. Feamster. Ac-dc: Adaptive ensemble classification for network traffic identification. *arXiv preprint arXiv:2302.11718*, 2023. (Cited on page 45.)

[63] R. Jordaney, K. Sharad, S. K. Dash, Z. Wang, D. Papini, I. Nouretdinov, and L. Cavallaro. Transcend: Detecting concept drift in malware classification models. In *26th {USENIX} Security Symposium ({USENIX} Security 17)*, pages 625–642, 2017. (Cited on pages 36 and 39.)

[64] R. Kamath and K. M. Sivalingam. Machine learning based flow classification in DCNs using P4 switches. In *International Conference on Computer Communications and Networks*, 2015. (Cited on pages 2 and 26.)

[65] A. Kantchelian, S. Afroz, L. Huang, A. C. Islam, B. Miller, M. C. Tschantz, R. Greenstadt, A. D. Joseph, and J. Tygar. Approaches to adversarial drift. In *Proceedings of the 2013 ACM workshop on Artificial intelligence and security*, pages 99–110, 2013. (Cited on page 38.)

[66] T. Karagiannis, K. Papagiannaki, and M. Faloutsos. Blinc: multilevel traffic classification in the dark. In *Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 229–240, 2005. (Cited on pages 18 and 26.)

[67] A. Kenyon, L. Deka, and D. Elizondo. Are public intrusion datasets fit for purpose characterising the state of the art in intrusion event datasets. *Computers & Security*, 99:102022, 2020. (Cited on page 18.)

[68] M. B. Knebl and A. Albanna. Systems and methods for network performance forecasting, Sept. 6 2016. US Patent 9,439,081. (Cited on page 37.)

[69] R. Kohavi and G. H. John. Wrappers for feature subset selection. In *Artificial Intelligence*, 1997. (Cited on page 32.)

[70] V. Krishnamoorthi, N. Carlsson, E. Halepovic, and E. Petajan. BUFFEST: Predicting buffer conditions and real-time requirements of HTTP(S) adaptive streaming clients. In *MMSys'17*, Taipei, Taiwan, jun 2017. (Cited on pages 8 and 10.)

[71] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012. (Cited on page 19.)

[72] Labeled video sessions dataset. https://nm-public-data.s3.us-east-2.amazonaws.com/dataset/all_traffic_time_10.pkl, 2019. (Cited on page 10.)

[73] C. Labovitz, S. Iekel-Johnson, D. McPherson, J. Oberheide, and F. Jahanian. Internet traffic and content consolidation. *77th Internet Engineering Task Force*, 2010. (Cited on page 18.)

[74] T. Lang, G. Armitage, P. Branch, and H.-Y. Choo. A synthetic traffic model for half-life. In *Australian Telecommunications Networks & Applications Conference*, volume 2003, 2003. (Cited on page 18.)

[75] T. Lang, P. Branch, and G. Armitage. A synthetic traffic model for quake3. In *Proceedings of the 2004 ACM SIGCHI International Conference on Advances in computer entertainment technology*, pages 233–238, 2004. (Cited on page 18.)

[76] X. Lian, Y. Chen, et al. Configuration validation with large language models. *arXiv preprint arXiv:2310.09690*, 2023. (Cited on page 47.)

[77] E. Liang, H. Zhu, X. Jin, and I. Stoica. Neural packet classification. In *Proceedings of the ACM Special Interest Group on Data Communication*, 2019. (Cited on page 26.)

[78] Z. Lin, A. Jain, C. Wang, G. Fanti, and V. Sekar. Using gans for sharing networked time series data: Challenges, initial promise, and open questions. In *Proceedings of the ACM Internet Measurement Conference*, pages 464–483, 2020. (Cited on page 18.)

[79] S. Liu, F. Bronzino, P. Schmitt, A. Nitin Bhagoji, N. Feamster, H. G. Crespo, T. Coyle, and B. Ward. Leaf: Navigating concept drift in cellular networks. *Proceedings of the ACM on Networking*, 2023. (Cited on pages 35, 37 and 40.)

[80] S. Liu, T. Mangla, T. Shaowang, J. Zhao, J. Paparrizos, S. Krishnan, and N. Feamster. Amir: Active multimodal interaction recognition from video and network traffic in connected environments. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 7(1):1–26, 2023. (Cited on page 36.)

[81] S. Liu, P. Schmitt, F. Bronzino, and N. Feamster. Characterizing service provider response to the covid-19 pandemic in the united states. In *PAM 2021-Passive and Active Measurement Conference*, 2021. (Cited on page 36.)

[82] Y. Liu, W. Li, and Y. Li. Network traffic classification using K-means clustering. In *International Multi-Symposiums on Computer and Computational Sciences*, 2007. (Cited on pages 2 and 26.)

[83] 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(3):1999–2012, 2020. (Cited on pages 18 and 26.)

[84] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang. Learning under concept drift: A review. *IEEE Transactions on Knowledge and Data Engineering*, 31(12):2346–2363, 2018. (Cited on page 36.)

[85] A. Lutu, D. Perino, M. Bagnulo, E. Frias-Martinez, and J. Khangosstar. A characterization of the covid-19 pandemic impact on a mobile network operator traffic. In *Proceedings of the ACM Internet Measurement Conference*, pages 19–33, 2020. (Cited on page 36.)

[86] Q. Ma, W. Huang, Y. Jin, and J. Mao. Encrypted traffic classification based on traffic reconstruction. In *2021 4th International Conference on Artificial Intelligence and Big Data (ICAIBD)*, pages 572–576. IEEE, 2021. (Cited on page 18.)

[87] A. L. Maas, A. Y. Hannun, A. Y. Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3. Atlanta, GA, 2013. (Cited on page 19.)

[88] M. V. Mahoney. Network traffic anomaly detection based on packet bytes. In *Proceedings of the 2003 ACM symposium on Applied computing*, pages 346–350, 2003. (Cited on page 18.)

[89] A. Mallick, K. Hsieh, B. Arzani, and G. Joshi. Matchmaker: Data drift mitigation in machine learning for large-scale systems. *Proceedings of Machine Learning and Systems*, 4:77–94, 2022. (Cited on page 38.)

[90] T. Mangla, E. Halepovic, M. Ammar, and E. Zegura. Using session modeling to estimate http-based video qoe metrics from encrypted network traffic. *IEEE Transactions on Network and Service Management*, 16(3):1086–1099, 2019. (Cited on pages 4, 8, 10, 26 and 46.)

[91] M. H. Mazhar and Z. Shafiq. Real-time video quality of experience monitoring for https and quic. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pages 1331–1339. IEEE, 2018. (Cited on pages 2, 4, 8, 10, 11 and 46.)

[92] J. McHugh. Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory. *ACM Transactions on Information and System Security (TISSEC)*, 3(4):262–294, 2000. (Cited on page 18.)

[93] L. Mei, R. Hu, H. Cao, Y. Liu, Z. Han, F. Li, and J. Li. Realtime mobile bandwidth prediction using lstm neural network and bayesian fusion. *Computer Networks*, 182:107515, 2020. (Cited on page 36.)

[94] A. Mondal, S. Sengupta, B. R. Reddy, M. Koundinya, C. Govindarajan, P. De, N. Ganguly, and S. Chakraborty. Candid with youtube: Adaptive streaming behavior and implications on data consumption. In *NOSSDAV'17*, Taipei, Taiwan, June 2017. (Cited on page 8.)

[95] A. Moore, D. Zuev, and M. Crogan. Discriminators for use in flow-based classification. Technical report, 2005. (Cited on page 26.)

[96] L. Nardi, A. Souza, D. Koeplinger, and K. Olukotun. HyperMapper: a practical design space exploration framework. In *IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2019. (Cited on page 31.)

[97] T. T. Nguyen and G. Armitage. A survey of techniques for internet traffic classification using machine learning. *IEEE communications surveys & tutorials*, 10(4):56–76, 2008. (Cited on page 2.)

[98] A. Nichol, P. Dhariwal, A. Ramesh, P. Shyam, P. Mishkin, B. McGrew, I. Sutskever, and M. Chen. Glide: Towards photorealistic image generation and editing with text-guided diffusion models. *arXiv preprint arXiv:2112.10741*, 2021. (Cited on page 20.)

[99] S. K. Nukavarapu, M. Ayyat, and T. Nadeem. Miragenet-towards a gan-based framework for synthetic network traffic generation. In *GLOBECOM 2022-2022 IEEE Global Communications Conference*, pages 3089–3095. IEEE, 2022. (Cited on page 18.)

[100] Openwave Mobility. Mobile Video Index. https://landing.owmobility.com/mobile-video-index/, 2018. (Cited on page 8.)

[101] K. Pandey, A. Mukherjee, P. Rai, and A. Kumar. Diffusevae: Efficient, controllable and high-fidelity generation from low-dimensional latents. *arXiv preprint arXiv:2201.00308*, 2022. (Cited on page 20.)

[102] V. Paxson. Empirically derived analytic models of wide-area tcp connections. *IEEE/ACM transactions on Networking*, 2(4):316–336, 1994. (Cited on page 18.)

[103] L. Peng, B. Yang, and Y. Chen. Effective packet number for early stage internet traffic identification. In *Neurocomputing*, 2015. (Cited on page 26.)

[104] J. Piet, D. Nwoji, and V. Paxson. GGFAST: Automating generation of flexible network traffic classifiers. In *ACM Special Interest Group on Data Communication (SIGCOMM)*, 2023. (Cited on page 26.)

[105] C. Raab, M. Heusinger, and F.-M. Schleif. Reactive soft prototype computing for concept drift streams. *Neurocomputing*, 416:340–351, 2020. (Cited on page 39.)

[106] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, and M. Chen. Hierarchical text-conditional image generation with clip latents, 2022. *URL https://arxiv.org/abs/2204.06125*, 7, 2022. (Cited on pages 18 and 20.)

[107] J. Riihijarvi and P. Mahonen. Machine learning for performance prediction in mobile cellular networks. *IEEE Computational Intelligence Magazine*, 13(1):51–60, 2018. (Cited on page 37.)

[108] V. Rimmer, D. Preuveneers, M. Juarez, T. Van Goethem, and W. Joosen. Automated website fingerprinting through deep learning. *arXiv preprint arXiv:1708.06376*, 2017. (Cited on pages 18 and 26.)

[109] M. Ring, D. Schlör, D. Landes, and A. Hotho. Flow-based network traffic generation using generative adversarial networks. *Computers & Security*, 82:156–172, 2019. (Cited on page 18.)

[110] M. Ring, S. Wunderlich, D. Scheuring, D. Landes, and A. Hotho. A survey of network-based intrusion detection data sets. *Computers & Security*, 86:147–167, 2019. (Cited on page 18.)

[111] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022. (Cited on pages 18, 19 and 20.)

[112] Sandvine. Global Internet Phenomena Spotlight: Encrypted Internet Traffic. https://www.sandvine.com/hubfs/downloads/archive/global-internet-phenomena-spotlight-encrypted-internet-traffic.pdf, 2015. (Cited on page 8.)

[113] Sandvine. Phenomena—THE GLOBAL INTERNET PHENOMENA REPORT JANUARY 2023. https://www.sandvine.com/hubfs/Sandvine_Redesign_2019/Downloads/2023/reports/Sandvine%20GIPR%202023.pdf, 2023. (Cited on page 8.)

[114] J. C. Schlimmer and R. H. Granger. Incremental learning from noisy data. *Machine learning*, 1(3):317–354, 1986. (Cited on page 36.)

[115] T. Schneider, R. Birkner, and L. Vanbever. Snowcap: Synthesizing network-wide configuration updates. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, 2021. (Cited on page 47.)

[116] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison. Hidden technical debt in machine learning systems. In *Advances in neural information processing systems*, pages 2503–2511, 2015. (Cited on page 4.)

[117] V. Sehwag, C. Hazirbas, A. Gordo, F. Ozgenel, and C. Canton. Generating high fidelity data from low-density regions using diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11492–11501, 2022. (Cited on page 20.)

[118] J. Serra, I. Leontiadis, A. Karatzoglou, and K. Papagiannaki. Hot or not? forecasting cellular network hot spots using sector performance indicators. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pages 259–270. IEEE, 2017. (Cited on page 37.)

[119] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas. Taking the human out of the loop: A review of bayesian optimization. In *Proceedings of IEEE, vol. 104, no. 1*, 2016. (Cited on page 31.)

[120] T. Shapira and Y. Shavitt. FlowPic: A generic representation for encrypted traffic classification and applications identification. In *IEEE Transactions on Network and Service Management*, 2021. (Cited on pages 18 and 26.)

[121] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. *ICISSp*, 1:108–116, 2018. (Cited on page 18.)

[122] T. Sharma, P. Schmitt, F. Bronzino, N. Feamster, and N. Marwell. Spatial models for crowdsourced internet access network performance measurements. *arXiv preprint arXiv:2405.11138*, 2024. (Cited on page 46.)

[123] B. Shi, X. Bai, and C. Yao. An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition. *IEEE transactions on pattern analysis and machine intelligence*, 39(11):2298–2304, 2016. (Cited on page 19.)

[124] T. Shon and J. Moon. A hybrid machine learning approach to network anomaly detection. *Information Sciences*, 177(18):3799–3821, 2007. (Cited on page 36.)

[125] C. Sinclair, L. Pierce, and S. Matzner. An application of machine learning to network intrusion detection. In *Proceedings 15th Annual Computer Security Applications Conference (ACSAC'99)*, pages 371–377. IEEE, 1999. (Cited on page 36.)

[126] J. Singh and M. J. Nene. A survey on machine learning techniques for intrusion detection systems. *International Journal of Advanced Research in Computer and Communication Engineering*, 2(11):4349–4355, 2013. (Cited on page 2.)

[127] G. Siracusano, S. Galea, D. Sanvito, M. Malekzadeh, G. Antichi, P. Costa, H. Haddadi, and R. Bifulco. Re-architecting traffic analysis with neural network interface cards. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2022. (Cited on page 26.)

[128] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International conference on machine learning*, pages 2256–2265. PMLR, 2015. (Cited on pages 18, 46 and 47.)

[129] R. Sommer and V. Paxson. Outside the closed world: On using machine learning for network intrusion detection. In *2010 IEEE symposium on security and privacy*, pages 305–316. IEEE, 2010. (Cited on pages 2, 18 and 36.)

[130] J. Sommers, H. Kim, and P. Barford. Harpoon: a flow-level traffic generator for router and network tests. *ACM SIGMETRICS Performance Evaluation Review*, 32(1):392–392, 2004. (Cited on page 18.)

[131] Y. Song and S. Ermon. Generative modeling by estimating gradients of the data distribution. *Advances in neural information processing systems*, 32, 2019. (Cited on page 20.)

[132] T. Stockhammer. Dynamic adaptive streaming over HTTP: standards and design principles. In *ACM Conference on Multimedia Systems*, MMSys '11, San Jose, CA, February 2011. (Cited on page 8.)

[133] B. Sun, J. Feng, and K. Saenko. Return of frustratingly easy domain adaptation. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016. (Cited on page 13.)

[134] B. Sun, W. Yang, M. Yan, D. Wu, Y. Zhu, and Z. Bai. An encrypted traffic classification method combining graph convolutional network and autoencoder. In *2020 IEEE 39th International Performance Computing and Communications Conference (IPCCC)*, pages 1–8. IEEE, 2020. (Cited on page 18.)

[135] A. Tang, R. Beckett, et al. Lightyear: Using modularity to scale bgp control plane verification. In *Proceedings of the ACM SIGCOMM 2023 Conference*, 2023. (Cited on page 47.)

[136] tcpdump - dump traffic on a network. `https://www.tcpdump.org/manpages/tcpdump.1.html`, 2017. (Cited on page 9.)

[137] The Truth About Faster Internet: It's Not Worth It. `https://www.wsj.com/graphics/faster-internet-not-worth-it/`, 2019. (Cited on page 13.)

[138] K. Thomas, C. Grier, J. Ma, V. Paxson, and D. Song. Design and evaluation of a real-time url spam filtering service. In *2011 IEEE symposium on security and privacy*, pages 447–462. IEEE, 2011. (Cited on page 38.)

[139] M. U. Togbe, Y. Chabchoub, A. Boly, M. Barry, R. Chiky, and M. Bahri. Anomalies detection using isolation in concept-drifting data streams. *Computers*, 10(1):13, 2021. (Cited on page 39.)

[140] A. Vaswani, N. Shazeer, et al. Attention is all you need. *Advances in neural information processing systems*, 2017. (Cited on page 46.)

[141] J. R. Vergara and P. A. Estévez. A review of feature selection methods based on mutual information. In *Neural Computing and Applications*, 2014. (Cited on pages 31 and 32.)

[142] Video Collection Tools. `https://github.com/inria-muse/video_collection`, 2019. (Cited on page 10.)

[143] K. V. Vishwanath and A. Vahdat. Swing: Realistic and responsive network traffic generation. *IEEE/ACM Transactions on Networking*, 17(3):712–725, 2009. (Cited on page 18.)

[144] G. Wan, F. Gong, T. Barbette, and Z. Durumeric. Retina: Analyzing 100 GbE traffic on commodity hardware. In *ACM Special Interest Group on Data Communication (SIGCOMM)*, 2022. (Cited on pages 26 and 33.)

[145] G. Wan, S. Liu, F. Bronzino, N. Feamster, and Z. Durumeric. Cato: End-to-end optimization of ml traffic analysis pipelines. *arXiv preprint arXiv:2402.06099*, 2024. (Cited on pages 25, 32 and 44.)

[146] M. Wang, K. Zheng, D. Luo, Y. Yang, and X. Wang. An encrypted traffic classification framework based on convolutional neural networks and stacked autoencoders. In *2020 IEEE 6th International Conference on Computer and Communications (ICCC)*, pages 634–641. IEEE, 2020. (Cited on page 18.)

[147] W. Wang, M. Zhu, X. Zeng, X. Ye, and Y. Sheng. Malware traffic classification using convolutional neural network for representation learning. In *International Conference on Information Networking*, 2017. (Cited on pages 2 and 26.)

[148] Z. Xiong and N. Zilberman. Do switches dream of machine learning?: Toward in-network classification. In *ACM Workshop on Hot Topics in Networks*, 2019. (Cited on page 26.)

[149] L. Xu, M. Skoularidou, A. Cuesta-Infante, and K. Veeramachaneni. Modeling tabular data using conditional gan. *Advances in neural information processing systems*, 32, 2019. (Cited on page 18.)

[150] Q. Xu, S. Mehrotra, Z. Mao, and J. Li. Proteus: network performance forecast for real-time, interactive mobile applications. In *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*, pages 347–360, 2013. (Cited on page 37.)

[151] S. Xu, M. Marwah, M. Arlitt, and N. Ramakrishnan. Stan: Synthetic network traffic generation with generative neural models. In *Deployable Machine Learning for Security Defense: Second International Workshop, MLHat 2021, Virtual Event, August 15, 2021, Proceedings 2*, pages 3–29. Springer, 2021. (Cited on page 18.)

[152] H. Yang, Q. He, Z. Liu, and Q. Zhang. Malicious encryption traffic detection based on NLP. In *Security and Communication Networks*, 2021. (Cited on page 26.)

[153] L. Yang, W. Guo, Q. Hao, A. Ciptadi, A. Ahmadzadeh, X. Xing, and G. Wang. {CADE}: Detecting and explaining concept drift samples for security applications. In *30th {USENIX} Security Symposium ({USENIX} Security 21)*, 2021. (Cited on pages 36 and 39.)

[154] H. Yao, C. Liu, P. Zhang, S. Wu, C. Jiang, and S. Yu. Identification of encrypted traffic through attention mechanism based long short term memory. *IEEE Transactions on Big Data*, 2019. (Cited on page 18.)

[155] Y. Yin, Z. Lin, M. Jin, G. Fanti, and V. Sekar. Practical gan-based synthetic ip header trace generation using netshare. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pages 458–472, 2022. (Cited on pages 18 and 21.)

[156] X. You, M. Zhang, D. Ding, F. Feng, and Y. Huang. Learning to learn the future: Modeling concept drifts in time series prediction. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 2434–2443, 2021. (Cited on page 38.)

[157] S. Zander, D. Kennedy, and G. Armitage. Kute a high performance kernel-based udp traffic engine. 2005. (Cited on page 18.)

[158] L. Zhang, A. Rao, and M. Agrawala. Adding conditional control to text-to-image diffusion models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3836–3847, 2023. (Cited on page 20.)

[159] G. Zheng, I. Krikidis, C. Masouros, S. Timotheou, D.-A. Toumpakaris, and Z. Ding. Rethinking the role of interference in wireless networks. *IEEE Communications Magazine*, 52(11):152–158, 2014. (Cited on page 36.)

[160] W. Zheng, J. Zhong, Q. Zhang, and G. Zhao. Mtt: an model for encrypted network traffic classification using multi-task transformer. *Applied Intelligence*, pages 1–16, 2022. (Cited on pages 18 and 26.)

[161] I. Žliobaitė. Adaptive training set formation. 2010. (Cited on page 36.)