University of California

Santa Barbara

Credible Machine Learning for Networking: Diagnosis, Data Collection, and Network-Native Representation Learning

A dissertation submitted in partial satisfaction

of the requirements for the degree

Doctor of Philosophy

in

Computer Science

by

Roman Beltiukov

Committee in charge:

Professor Arpit Gupta, Chair

Professor Wenbo Guo

Professor Walter Willinger, Northwestern University

June 2026

The Dissertation of Roman Beltiukov is approved.

Professor Wenbo Guo

Professor Walter Willinger, Northwestern University

Professor Arpit Gupta, Committee Chair

May 2026

Credible Machine Learning for Networking: Diagnosis, Data Collection, and Network-Native Representation Learning

by

Roman Beltiukov

Acknowledgements

This Ph.D. started, probably, at the most uncertain time of my life, when everyone around me was still fighting the pandemic while simultaneously being on the verge of another, more dangerous, war. Across several countries, societal shakeups, and bureaucracy, the opportunities and actions barely aligned enough to let me overcome everything and pursue my research program here, on the coast of the Pacific Ocean at UC Santa Barbara, which resulted not only in obtaining knowledge in my research area, but also in reshaping and changing many things about my life and myself for the better. This path was interesting but complicated, and I would like to acknowledge many people who helped and supported me through all these years.

First, I want to express my deepest gratitude and appreciation to my advisor, Arpit Gupta. We spent countless hours and days discussing ideas, problems, fixes, and solutions, which later resulted in successful projects and papers. Sometimes it was not clear what we wanted to achieve, and sometimes we seemed to disagree on everything, drawing things on a whiteboard or exchanging messages at 4 a.m. trying to decide on the best course of action, but in the end this turned out to be an insanely productive collaboration, which shaped my research trajectory and helped immensely.

I also want to thank two other members of my committee: Wenbo Guo and Walter Willinger. They both provided a lot of feedback for most of my projects and created a productive discussion space, challenging many of the definitions and improving our ideas. Special thanks here goes to their ability to notice all the small mistakes in writing that my brain was ignoring after the 17th time reading the paper: the skill I still wish to learn someday. :)

Besides my direct research committee, I want to acknowledge all my labmates, from both SNL and MOMENT labs, for their endless support and help. From struggling over

courses to chatting at netBrunch, we shared a lot of common knowledge and memories, and their kind words and actions were always helpful throughout all these years.

A separate thanks goes to all my friends, partners, neurodiversity groups, and queer groups at UCSB, who helped me build a community around me and stay positive despite all the problems, as I knew I was not alone and could rely on the people around me. Genuinely, you all mean a lot to me.

And finally, a special one: to my close friend, Liu Kurafeeva, who shared with me so many challenges that I cannot enumerate, or even remember, them all anymore, and made my life so much easier that I sincerely cannot imagine how much more difficult it would have been otherwise. I really appreciate this, and you helped me a lot through all these years.

Thank you all again. ♡

Curriculum Vitæ

Roman Beltiukov

Education

2026	Ph.D. in Computer Science (Expected), University of California, Santa Barbara.
2019	M.S. in Computer Science, Peter the Great St. Petersburg Polytechnic University.
2017	B.S. in Computer Science, Peter the Great St. Petersburg Polytechnic University.

Selected publications

- Beltiukov, Roman and Guthula, Satyandra and Guo, Wenbo and Willinger, Walter and Gupta, Arpit. "Demystifying Network Foundation Models". Advances in neural information processing systems (NeurIPS), 2025
- Beltiukov, Roman and Guthula, Satyandra and Manda, Haarika and Daneshamooz, Jaber and Guo, Wenbo and Willinger, Walter and Gupta, Arpit and Monga, Inder. "netFound: From Diagnostics-Informed Design to Production-Ready Network Foundation Model". arXiv preprint arXiv:2310.17025, 2023
- Beltiukov, Roman and Guo, Wenbo and Gupta, Arpit and Willinger, Walter. "In Search of netUnicorn: A Data-Collection Platform to Develop Generalizable ML Models for Network Security Problems". Proceedings of the 2023 ACM SIGSAC CCS, 2023
- Beltiukov, Roman and Chandrasekaran, Sanjay and Gupta, Arpit and Willinger, Walter. "Pinot: Programmable infrastructure for networking". Proceedings of the 2023 Applied Networking Research Workshop, 2023
- Jacobs, Arthur S and Beltiukov, Roman and Willinger, Walter and Ferreira, Ronaldo A and Gupta, Arpit and Granville, Lisandro Z. "Ai/ml for network security: The emperor has no clothes". Proceedings of the 2022 ACM SIGSAC CCS, 2022
- Beltiukov, Roman. "Optimizing Q-learning with K-FAC algorithm". International Conference on Analysis of Images, Social Networks and Texts, 2019

Awards

- Resonant Fellowship, University of California, Santa Barbara 2026
- IETF Applied Networking Prize 2023
- UCSB CS Department Summer Fellowship Award 2023
- World IT Planet Winner, Cloud Computing Track by Huawei 2018
- Special Award, Institute of Bioinformatics, EPAM Systems 2018

- World IT Planet Winner, Cloud Computing Track by Huawei 2017
- "Robots and Learning Machines Track" Winner, Junction Hackathon 2017

Volunteering

- President of Computer Science Graduate Student Association, UC Santa Barbara 2025-2026
- President of Virtual Reality club, UC Santa Barbara 2025-2026
- Mentor for Early Research Student Program, UC Santa Barbara 2023-2026

Abstract

Credible Machine Learning for Networking: Diagnosis, Data Collection, and Network-Native Representation Learning

by

Roman Beltiukov

Machine learning promises to help networks monitor, diagnose, secure, and optimize themselves under conditions that increasingly exceed the reach of static rules and manual operation. Yet this promise in academia is limited by a credibility crisis: many network ML systems report excellent benchmark performance while remaining opaque, brittle under deployment shifts, and dependent on artifacts of the data-collection or evaluation pipeline. This dissertation argues that credible machine learning for networking requires treating diagnosis, data collection, and model design as a connected problem rather than as isolated stages.

The dissertation studies this problem through three complementary methods. First, it develops diagnostic techniques for scrutinizing what network ML models learn. Trustee extracts high-fidelity, low-complexity decision-tree explanations and trust reports from black-box models, and the Intrinsic Evaluation Framework probes network foundation models through embedding geometry, alignment with domain-expert traffic features, and controlled sensitivity tests. Second, it builds data-collection infrastructure for realistic and reusable experimentation: PINOT combines passive monitoring and active measurements on the UCSB production network, while netUnicorn provides a thin-waist, intent-based platform for closed-loop data collection across heterogeneous physical and virtual environments. Third, it translates the diagnostic findings into model design through netFound, a network-native foundation model with protocol-aware tokenization, operational context

embeddings, burst-flow hierarchical attention, and privacy-by-construction inputs, and explores context-grounded LLMs for interpreting network measurements.

Together, these studies show that benchmark accuracy alone is a poor proxy for deployment credibility. Published network ML models can learn shortcuts, spurious correlations, and out-of-distribution vulnerabilities; existing network foundation models often exhibit collapsed embeddings, inconsistent alignment with expert features, and unwanted payload dependence. The systems contributions show that realistic, iterative data collection can expose and reduce these failures, improving generalization across environments and attack variants while reducing experiment implementation effort. Finally, netFound demonstrates that diagnostics-informed design yields more reusable traffic representations, stronger intrinsic behavior, and competitive or superior downstream performance across diverse networking tasks. Overall, the dissertation establishes credibility as the organizing principle for evaluating, collecting data for, and designing machine learning systems for networking.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Computer networks are expected to support increasingly heterogeneous applications, stronger privacy guarantees, and tighter performance and security requirements, even as their internal behavior becomes harder to observe and reason about directly. Large portions of traffic are encrypted [6], operational conditions change rapidly, and the traffic patterns that matter for diagnosis or control are often only indirectly visible. These realities have long motivated the vision of self-driving networks: infrastructures that can monitor, reason about, and adapt to their own behavior with minimal human intervention [87, 163, 172]. Realizing that vision requires learned traffic representations[1] that remain useful under real deployment constraints [18], and recent production-oriented model designs [265] make clear that such representations often must be inferred from packet headers, timing, and flow behavior rather than payload alone.

Machine learning appears to offer exactly this kind of capability. Across networking and network security, researchers have used ML to classify traffic [6, 144, 263], detect anomalies and attacks [19, 23, 78, 166], infer application QoE [7, 158, 258], and optimize operational decisions [157, 251], often substantially outperforming static heuristics on

[1] Learned embeddings that map high-dimensional data into lower-dimensional spaces.

benchmark datasets. More recently, network foundation models [144, 241, 264, 265] have renewed this promise by suggesting that large-scale self-supervised pretraining can produce reusable traffic representations that transfer across tasks. If successful, this shift would move networking away from narrowly engineered task-specific models and toward general-purpose learned representations for monitoring, diagnosis, and control.

Yet this promise has not translated cleanly into practice. Network operators remain reluctant [248] to relinquish control to black-box models whose behavior they do not understand and whose robustness outside the original training setting is unclear. This reluctance is especially pronounced because wrong decisions can disrupt service, degrade user experience, or interfere with security-critical operations. The result is a widening gap between what the literature celebrates as success and what production environments are willing to trust [249, 250]. High benchmark scores, even when reproducible, often create the appearance of progress without yielding artifacts that can be depended on in deployment [18, 126, 148, 267].

This gap is best understood as a credibility crisis [248] rather than merely a reproducibility problem. Reproducibility asks whether a reported result can be recreated from the original artifacts. Credibility asks a harder question: whether the resulting model deserves to be trusted as a representation of the underlying networking problem, rather than as a solver for a narrow benchmark. In this dissertation, we use *credibility* as the combined term for *generalizability* and *trust*. A credible model is not simply accurate on held-out data from the same pipeline; it captures patterns that survive changes in environment, dataset, and operational context, and it exposes consistent behavior that experts can rely on when the stakes of deployment are high [64]. In networking, this credibility problem is amplified by the way data is collected and reused. Traffic is difficult to label, operational environments differ substantially, and privacy and encryption constrain what can be observed. As a result, many published datasets represent only a narrow slice of

the real problem, making underspecification[2] and brittle model behavior especially likely.

Across the works that motivate this dissertation, the same failure modes [64, 126] recur: shortcut learning, where a model relies on artifacts that correlate with labels but are not causally tied to the underlying phenomenon; spurious correlations, where accidental cues preserve performance while obscuring true signal; and out-of-distribution failure, where even modest shifts in traffic mix or environment break model behavior. In networking, these are not corner cases: small emulated testbeds, one-off collections, narrow train/test splits, and leakage through timestamps, protocol fields, or traffic-generation artifacts make them unusually likely [18, 148, 265]. For that reason, improving network ML cannot begin with optimization alone; it must begin by diagnosing what current models have actually learned.

Recent excitement around network foundation models [36] does not eliminate this challenge; it reframes it. Foundation models are appealing because they learn from large-scale unlabeled traffic, reduce dependence on task-specific feature engineering, and promise reusable representations across diverse downstream problems. But the evidence in Chapter 3 of this dissertation shows that pretraining at scale is not a substitute for credibility. Existing network foundation models can still produce collapsed embedding spaces[3], align weakly or inconsistently with domain-expert traffic features, miss exogenous network context, and depend too heavily on payload information that is often encrypted or unavailable in production. Resulting diagnostics-informed model-design work makes the broader lesson clear: better network models must be built in response to diagnosed failures, not merely in pursuit of larger benchmarks.

This dissertation responds to the credibility crisis along three connected fronts. First, it develops tools and methodologies for scrutinizing existing ML models and network

[2]Sensitivity to arbitrary training choices unpredictably influencing performance on held-out data.
[3]Defined as low-variable and poorly separable embeddings for different data.

foundation models, making failure modes visible rather than hidden behind aggregate metrics. Second, it argues that credibility is inseparable from data and therefore builds infrastructure that makes realistic, diverse, and reusable network data collection practical at scale. Third, it translates those diagnostic and data-centric insights into model design, culminating in *netFound* as a network-native foundation model and complemented by an exploration of context-grounded LLMs for networking tasks. The result is not a single technique, but a research program for making machine learning in networking more believable, more inspectable, and more deployable.

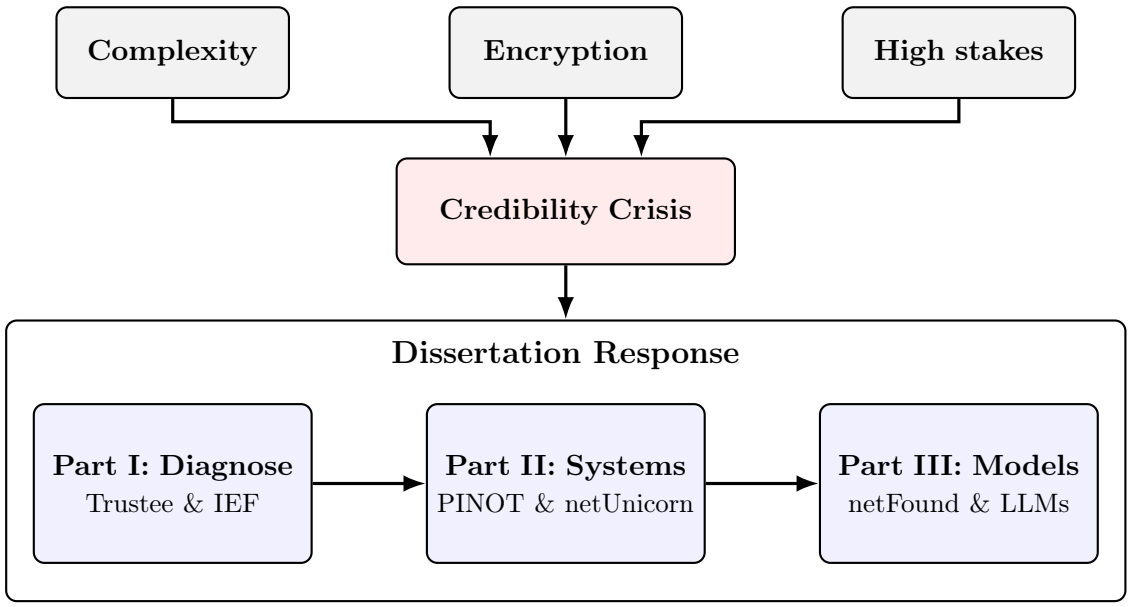


Figure 1.1: Dissertation overview from the credibility crisis in network ML to diagnosis, infrastructure, and model design.

Figure 1.1 summarizes this arc: from diagnosing why network ML often fails, to building the means to collect better data, to designing models that better respect the realities of deployment. The central claim of this dissertation is that these steps cannot be treated as separate concerns. Machine learning will become genuinely useful for networking only when credibility, rather than benchmark accuracy alone, becomes the organizing principle for how we evaluate models, collect data, and design network-native

representations.

1.1 Thesis Statement

This dissertation argues that the central challenge for machine learning in networking is credibility. Modern network ML solutions routinely report high benchmark scores while remaining brittle, underspecified, and difficult to trust beyond the narrow conditions in which they were trained and evaluated. To make these solutions genuinely useful in practice, we must treat diagnosis, data collection, and model design as a single connected problem: we must identify why existing models fail, build infrastructure that makes higher-quality and more diverse network data practical to collect, and design network-native models that translate those lessons into more generalizable and trustworthy representations.

1.2 Key Contributions

1.2.1 Credibility of Machine Learning for Networking

To understand and improve the credibility of existing machine learning solutions for network data, we pursue three complementary steps.

1. **Credibility analysis and formulation of the problem.** We revisit published, reproducible ML artifacts for networking and network security and analyze their models, datasets, and evaluation pipelines to understand whether the reported performance is actually credible.

   Contributions: By reproducing and scrutinizing representative case studies, we show that benchmark accuracy alone is insufficient: a model is credible only when its behavior reflects patterns that generalize beyond the narrow conditions of training

and evaluation, rather than artifacts of the dataset or pipeline. From this analysis, we formulate the core credibility problem of ML for networking as a problem of underspecification and organize its most common manifestations into three categories: shortcut learning, spurious correlations, and vulnerability to out-of-distribution data. This contribution establishes the conceptual foundation for the rest of the dissertation by defining what must be diagnosed, explained, and ultimately fixed before ML solutions for networking can be trusted in practice.

2. **Explainability framework for models analysis.** We develop Trustee and use it to make the credibility problem actionable in practice.

   Contributions: We design and implement Trustee as a reusable framework for analyzing black-box ML models for networking and network security, and release it together with reproducibility artifacts. We then apply Trustee to published, reproducible ML case studies to demonstrate how the framework can be used in practice. Through these case studies, we show how model analysis can move from abstract concerns about trustworthiness to concrete diagnosis of problems in the data, features, and evaluation pipeline. This contribution turns the credibility problem into a practical and reproducible workflow for scrutinizing ML solutions for networking.

3. **Intrinsic representation analysis.** We complement whitebox- and downstream task-based analysis with an orthogonal methodology for assessing what network ML models learn before task-specific fine-tuning.

   Contributions: We design an Intrinsic Evaluation Framework (IEF) that evaluates learned representations directly, independent of downstream tasks and their labels, and use it to benchmark state-of-the-art network foundation models across diverse datasets. Through this analysis, we show how to uncover what these models encode

about network traffic, including representation geometry, alignment with established domain-expert metrics, and sensitivity to protocol- and context-level changes. Our analysis reveals limitations and failure modes that conventional downstream benchmarks miss, including anisotropy, inconsistent encoding of established network metrics, and strong dependence on payload information. We release the framework and benchmarking artifacts as reproducible code, turning intrinsic representation analysis into a practical methodology for comparing and understanding network ML solutions.

1.2.2 Infrastructure for Networking Research

Due to complexity of credible data collection for networking research, which results in a lack of high-quality diverse datasets for both training and evaluation of machine learning solutions, we propose and implement large-scale data collection infrastructure and a thin-waist intent-based framework, aimed to facilitate the collection of high-quality datasets and enable the development of generalizable machine learning solutions for networking.

1. **Data collection infrastructure.** We develop PINOT as a programmable campus-scale infrastructure for collecting realistic network data for ML-based networking research.

   Contributions: We design, implement, and deploy PINOT at UCSB as a combined active and passive data-collection platform built on top of the production campus network. On the passive side, we implement privacy-preserving packet-level collection at the campus border gateway and augment the resulting traces with logs from production appliances to support labeling for learning problems. On the active side, we deploy a fleet of more than 50 measurement devices across diverse campus

locations to collect representative measurements from different physical vantage points and network conditions. We use this infrastructure to curate realistic datasets for multiple networking problems, including QoE inference for video applications, traffic detection for security appliances, and device or application fingerprinting, and make the platform publicly available as an open research infrastructure. This contribution establishes the dissertation's first systems path toward improving data quality and, in turn, the credibility of network ML solutions.

2. **Thin-waist intent-based framework.** We develop a thin-waist, intent-based framework for flexible and reusable network data collection across disparate environments.

   Contributions: We design and implement netUnicorn as a new programming abstraction and data-collection platform that decouples high-level data-collection intent from infrastructure-specific mechanisms and decomposes experiments into reusable tasks and pipelines. We show that this thin-waist abstraction enables the same learning problem to be instantiated across multiple network environments, allows multiple learning problems to reuse a common set of tasks, and supports iterative data collection as part of a closed-loop workflow for improving model generalizability. Through experiments over different networking problems and physical and virtual infrastructures, we demonstrate that netUnicorn substantially reduces the effort required to express, port, reproduce, and share data-collection experiments. We release the full system, task and pipeline library, and supplemental artifacts, turning the collection of diverse, high-quality networking datasets into a practical and reusable systems capability.

1.2.3 Towards Generalizable Models

To address the generalizability issues in the ML models themselves, we pursue two complementary directions. First, we develop *netFound*, a network-native foundation model that learns reusable traffic representations from large-scale unlabeled network data and translates the diagnostic insights from Part I into concrete model design choices. Second, we explore how pretrained text-based LLMs can be adapted to networking tasks when grounded with additional domain context. Together, these efforts show two paths toward more generalizable solutions: building domain-specific foundation models for network traffic and carefully constraining broad pretrained models with networking-specific evidence.

1. **Network foundation model.** We develop *netFound* as the main model-side contribution of the dissertation for learning reusable and generalizable traffic representations from large-scale unlabeled data.

   Contributions: We translate the credibility and representation failures identified earlier in the dissertation into concrete design principles for model construction and use them to design and implement *netFound*. We train the model on large-scale real network traffic collected through the infrastructure developed in Part II and evaluate it using both intrinsic analysis and downstream tasks. Through this study, we show that *netFound* learns higher-quality and more reusable traffic representations than prior network foundation models and results in stronger generalization across diverse networking tasks. We also release the model implementation and training artifacts as reproducible research artifacts, turning *netFound* into a practical foundation for future work on generalizable ML for networking.

2. **Context-grounded large language models.** We explore a complementary route toward generalizable networking solutions by leveraging pretrained text-based LLMs

together with additional contextual grounding.

Contributions: We design and implement a proof-of-concept framework that combines network measurements with geolocation and retrieval over historical measurements to produce grounded interpretations of speed test results for non-expert users. This work shows that broad pretrained LLMs can be adapted to networking problems without task-specific model training when they are supplied with relevant external context, and provides an example of how contextual grounding can improve the correctness and usefulness of LLM-based networking applications.

1.3 Broader Impact

The impact of this dissertation extends beyond its individual technical contributions. At a high level, the work helps shift machine learning for networking away from narrow benchmark-driven progress and toward practices that are more credible, reusable, and deployable. Its broader impact is therefore not only scientific, but also methodological and infrastructural: it has influenced how the community evaluates network ML models, how researchers build data-collection workflows, and how subsequent efforts frame representation learning for network telemetry.

One form of impact has been community recognition and dissemination of the dissertation's core ideas. Trustee, which turns black-box network ML models into inspectable decision processes, received the Applied Networking Research Prize (ANRP) from the IETF/IRTF and a Best Paper Honorable Mention at ACM CCS 2022. The same line of work also helped motivate the ACM SIGCOMM 2023 tutorial on closed-loop "ML for Networks" pipelines, where ideas from Trustee and netUnicorn were presented as part of a broader methodology for building, diagnosing, and iteratively improving learning-based networking systems. These outcomes indicate that the dissertation's central argument,

namely that accuracy alone is not enough without scrutiny and iteration, resonates with a wider research community.

A second form of impact is practical uptake through shared infrastructure. The systems work in this dissertation was designed to lower the barrier to collecting realistic and diverse networking data, and that goal has already extended beyond the original deployment environment. netUnicorn has been deployed at UC Santa Barbara, IIT Delhi, and ETH Zurich, demonstrating that the abstractions developed here are useful across distinct research settings rather than only within a single local testbed. The follow-on NetReplica [65] system was also directly inspired by netUnicorn, providing an additional signal that the ideas introduced in this dissertation are already informing subsequent systems for network experimentation and data generation.

Finally, the dissertation's model-analysis and representation-learning agenda has influenced ongoing work beyond the immediate projects reported here. The intrinsic evaluation line of work has already shaped follow-on efforts recognized through a Cisco award, reflecting continued interest in principled methods for understanding learned network representations. More broadly, *netFound* and the related NetBurst [110] effort are serving as intellectual foundations for representation learning on network telemetry data in two major DOE efforts led by Lawrence Berkeley National Laboratory: the Genesis project on AIOps for networking and the American Science Cloud project. Together, these downstream uses suggest that the dissertation contributes not only point solutions, but also a longer-term research direction for building trustworthy, generalizable machine learning systems for networking.

1.4 Dissertation Outline

The dissertation proceeds in three parts that mirror the credibility loop: *Part I* diagnoses why current network ML and NFMs fail, *Part II* builds the infrastructure needed to collect realistic and reusable data, and *Part III* translates those lessons into models that generalize across deployment conditions.

Credibility of Machine Learning for Networking explores the state of the art in networking research and underlying reasons for high performance metrics, demonstrating problems with shortcuts, generalizability, and overall network solutions credibility. Chapter 2 establishes common generalizability problems in network data (shortcuts, *o.o.d.* issues, and spurious correlations) and demonstrates that lack of attention to these details leads to overinflated performance metrics and overpromises in generalizability of these solutions. It also proposes *Trustee*, a framework for network ML models analysis and understanding, which turns black-box solutions into explainable decisions to verify their correctness. Chapter 3 continues the credibility analysis and proposes a completely different angle for comparison of network ML solutions, which does not utilize fine-tuning datasets to avoid the very problems with shortcuts propagation, but instead explores the internal representations of traffic from the models to verify its geometric, expressive, and behavioral features. This allows not only to verify correctness and lack of shortcuts for network ML models, but also to compare them and explore their capabilities beyond downstream datasets.

Infrastructure for Networking Research proposes the first path for improving generalizability of network ML solutions based on improving the data collection efforts to obtain more diverse, shortcut-free data. Chapter 4 presents a physical infrastructure, deployed at University of California, Santa Barbara, that allows us to collect real-world network data, both from active measurement experiments created by researchers and passive network

traffic monitoring of dozens of thousands of real-world users. This effort significantly increases the quality of the data and removes most of the issues connected to small-scale lab setups, which are common for data collection in the field. Chapter 5 further expands the scalable data collection initiative, creating a Docker-based, infrastructure-agnostic, thin waist between researcher intent expression and physical world setups, allowing users to quickly implement and modify network measurement experiments and deploy them on various platforms, from local setups to hundreds of cloud nodes. Both these solutions aim to improve the data quality for public network datasets, partially solving the credibility problem.

Next, *Towards Generalizable Models* describes the second path for solving the credibility problem via using foundation models exposed to huge diverse unlabeled datasets to avoid problems of small data scale and downstream task specific shortcuts. Chapter 6 proposes the network foundation model which design is motivated by failure analysis described in Chapter 2 and Chapter 3, and trained on data collected from infrastructure described in Chapter 4. We demonstrate that this approach results in significantly higher generalizability of the ML solution on variety of downstream tasks compared to competitors. Besides network-specific models, we also explore usage of common text-based Large Language Models in Chapter 7, demonstrating their generalizability on previously unseen tasks of speedtest results explanation.

Finally, Chapter 8 and Chapter 9 discuss findings and results and provide recommendation and directions for future expansion.

Part I

Credibility of ML for Networking

This part establishes the dissertation's diagnostic foundation by asking what it would take for machine learning for networking to be credible enough for deployment. Rather than treating high benchmark accuracy as sufficient evidence of progress, the chapters in this part examine whether network ML models have learned robust and meaningful signal or whether they rely on artifacts of the data and evaluation setup. Together, they develop the dissertation's core view of credibility as the combination of trust and generalizability: a model should expose behavior that experts can scrutinize and should capture patterns that survive beyond narrow benchmark settings.

Chapter 2 studies black-box ML models for network security and introduces Trustee, a framework that extracts compact decision-tree explanations and trust reports to uncover shortcut learning, spurious correlations, and out-of-distribution vulnerabilities in published systems. Chapter 3 extends this diagnostic perspective to network foundation models by proposing an intrinsic representation analysis framework based on embedding geometry, alignment with domain-expert traffic features, and controlled sensitivity tests. In the overall dissertation, these two works make the credibility problem concrete and motivate the later parts: before we can build better data-collection infrastructure or more generalizable models, we must first understand what existing models are actually learning and why current evaluations so often overstate their reliability.

Chapter 2

AI-ML and Cybersecurity: The Emperor Has No Clothes

2.1 Introduction

In the last few years, we have witnessed a growing tension in the network-security community. Recent research has demonstrated the benefits of Artificial Intelligence (AI) and Machine Learning (ML) models over simpler rule-based heuristics in identifying complex network traffic patterns for a wide range of network security problems (see recent survey articles such as [35, 171, 215, 252]). At the same time, we have seen reluctance among network security researchers and practitioners when it comes to adopting these ML-based research artifacts in production settings (*e.g.,* see [15, 18, 225]). The black-box nature of most of these proposed solutions is the primary reason for this cautionary attitude and overall hesitance. More concretely, the inability to explain how and why these models make their decisions renders them a hard sell compared to existing simpler but typically less effective rule-based approaches.

This tension is not unique to network security problems but applies more generally

to any learning models, especially when their decision-making can have serious societal implications (*e.g.,* healthcare, credit rating, job applications, and criminal justice system). At the same time, this basic tension has also driven recent efforts to "crack open" the black-box learning models, explaining why and how they make their decisions (*e.g.,* "interpretable ML" [206], "explainable AI (XAI)" [233], and "trustworthy AI" [41]). However, to ensure that these efforts are of practical use in particular application domains of AI/ML such as network security is challenging and requires further qualifying notions such as (model) interpretability or trust (in a model) [145] and also demands solving a number of fundamental research problems in these new areas of AI/ML.

In this paper, we first provide such a qualification that is motivated by the needs of the field of network security as application domain of AI/ML and equates *"an end user having trust in an AI/ML model"* with *"an end user being comfortable with relinquishing control to the model"* [145]. Given this specific definition of what it means for an AI/ML model to engender trust, we next address a number of fundamental research challenges related to the problem of quantitatively deciding when an end user is comfortable with relinquishing control to a given AI/ML model. To this end, a particular focus of this paper is on determining whether or not a given AI/ML model suffers from the *problem of underspecification* [64].

Here, the problem of underspecification in modern AI/ML refers to determining whether the success of a trained model (*e.g.,* high accuracy) is indeed due to its innate ability to encode some essential structure of the underlying system or data or is simply the result of some inductive biases that the trained model happens to encode. In practice, inductive biases typically manifest themselves in instances of shortcut learning strategies [93], signs of spurious correlations [17], or an inherent inability for out-of-distribution (o.o.d.) generalizations (*i.e.,* test data distribution is different from training data distribution). The implication of such inductive biases is that their presence in

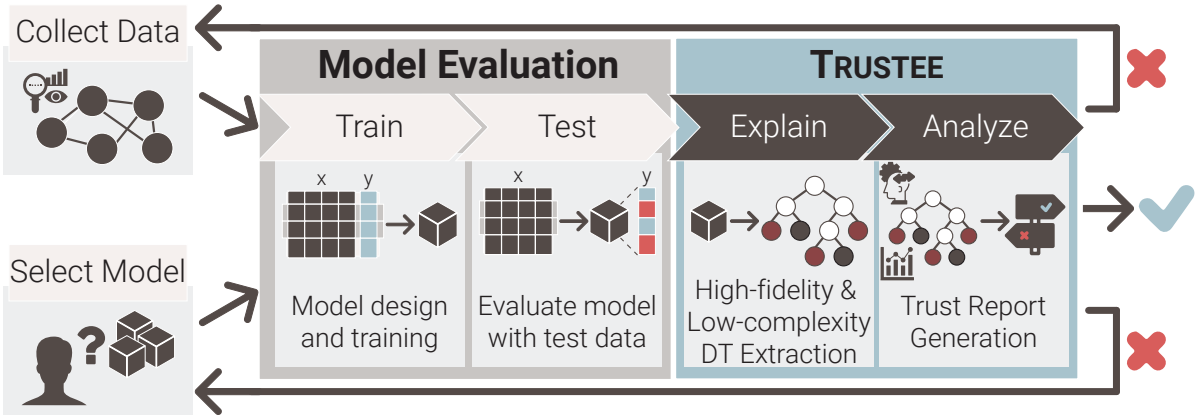


Figure 2.1: Trustee overview.

trained AI/ML models prevents these models from being credible or trustworthy; that is, generalize as expected in deployment scenarios. Thus, for establishing the specific type of trust in an ML model considered in this paper, it is critical to be able to identify these inductive biases, and this paper takes a first step towards achieving this ambitious goal.

To detect underspecification issues in learning models for network security problems, we develop Trustee (TRUSt-oriented decision TreE Extraction). This framework provides a means for carefully inspecting black-box learning models for the presence of inductive biases. Figure 2.1 shows how Trustee augments the traditional ML pipeline to examine the trustworthiness of a given ML model. Specifically developed with the application domain of network security in mind, Trustee takes a given black-box model and the dataset that has been used to train that model as input and outputs a "white-box" model in the form of a high-quality decision tree (DT) explanation.

Importantly, in synthesizing this DT, Trustee's focus is first and foremost on ensuring its practical use which, in turn, requires leveraging domain-specific observations to strike a balance between *model fidelity* (*i.e.,* accuracy of the DT with respect to the black-box model), *model complexity*, and *model stability*. Here, complexity refers to both the size of the DT and to aspects of the tree's branches. In particular, when viewing the tree's

branches as decision rules, we are concerned with their explicitness and intelligibility; that is, we require these rules to be readily recognizable by domain experts and be largely in agreement with the experts' domain knowledge. Model stability, on the other hand, is concerned with the correctness, coverage and stability of the decision rules; that is, we require them to correctly describe how the given black-box model makes a significant number of its decisions and also want them to be largely insensitive to the particular data samples that Trustee used in the process of selecting its final DT explanation. We achieve this insensitivity or stability by implementing a heuristic method that selects from among a number of different candidate DTs the one that has the highest mean agreement. Here, the agreement between two different DTs is a measure of how often the two DTs will make the same decision for the same input data [105, 234]. In practical terms, implementing this heuristic reduces the likelihood that Trustee outputs a misleading DT explanation.

Trustee also outputs a trust report associated with the DT explanation, which operators can consult to determine whether there is evidence that the given black-box model suffers from the problem of underspecification. If such evidence is found, the information provided in the trust report can be used to identify components of the traditional ML pipeline (*e.g.,* training data and model selection) that need to be modified in an effort to improve upon an ML model that Trustee has found to be untrustworthy.

While our work contributes to the rapidly growing ML literature on model explainability/interpretability and is inspired by ongoing developments in this area, our efforts and objectives differ from existing approaches in a number of significant ways. For one, given the inherent complexity of learning problems for networking, existing approaches for replacing black-box models with "white-box" models that are inherently explainable in the first place (*e.g.,* decisions trees) are in general impractical. Moreover, *local interpretability* methods [106, 199, 213] are not suitable for examining the various instances of the underspecification problem. At the same time, although our effort is motivated by prior studies

that focus on *global interpretability* [24, 25, 137], these works are either only applicable to a specific class of learning models (*e.g.,* reinforcement learning) or suffer from poor fidelity.

Through various case studies, we illustrate in Section 2.7 how operators can use Trustee's DTs and associated trust reports to detect the presence of inductive biases. More specifically, we use published ML models that are reproducible (*i.e.,* code base and datasets are publicly available) to show how network operators can use the information provided by Trustee to detect instances of shortcut learning strategies, obtain evidence of overfitting and/or whether the model relies on spurious correlations to make its decisions, or determine the model's inability to generalize to out-of-distribution data.

2.2 Background and Related Work

The application domain of AI/ML considered in this paper is the area of network security. In this section, we first discuss the unique challenges that this area poses for utilizing the latest advances in AI/ML. In particular, we focus on important recent AI/ML concepts such as "interpretable ML" and "explainable AI" and discuss their relevance for our work.

2.2.1 Challenges in ML for Network Security

Beyond the already-mentioned trust issue, there are a number of other reasons why the area of network security is a particularly challenging application domain for AI/ML. Networking-related datasets in general and cybersecurity-specific datasets in particular typically contain information about what is being communicated over a network (*e.g.,* packet-level traffic traces) or provide insight into how networks enable such information exchanges. As such, the datasets often raise serious end user-specific privacy concerns or reveal provider-specific details that many companies consider to be proprietary in nature

and are therefore unwilling to share. The result is a general paucity of publicly available datasets. Moreover, the datasets that are publicly available generally lack the complexity of real-world settings, either because they have been synthetically generated, have been obtained from small-scale testbed environments, or have been anonymized to the point where their general utility has been severely curtailed.

The scarcity of carefully labeled data poses an even bigger problem. Networking or cybersecurity datasets do not come in the form of images that humans can recognize but typically consist of semantically rich content, and unpacking that content and properly labeling it often requires substantial domain knowledge (*e.g.,* network architecture, protocols, and standards). This need for domain knowledge rules out labeling approaches that have been used successfully in other domains and include crowdsourcing (*e.g.,* for labeling images that are part of open-source databases such as ImageNet [68]) or outsourcing (*e.g.,* for labeling datasets that have been curated and open-sourced by commercial self-driving car companies for the benefit of researchers [44, 50]).

2.2.2 Interpretable ML and Explainable AI

As the scientific community continues to develop sophisticated AI/ML-based tools for high stakes decision-making throughout society, there has been a growing awareness about their actual or potential misuses and negative implications. As a result, calls for starting to study "trustworthy AI", "responsible AI", "ethical AI" and related topics have intensified in recent years and have identified model interpretability/explainability as a critically important but also highly elusive concept for facilitating these studies [145].

Interpretable ML: Ex-ante Interpretability. The application of modern AI/ML has resulted in a myriad of different learning models that are "black-box" in nature; that is, provide no insight in or understanding about why the black-box model makes

certain decisions (and not some other decisions) or what decision-making process gives rise to these decisions. This development has resulted in a recent explosion of work on "Explainable AI," where a second (post-hoc) model is created to explain the originally obtained black-box model [233]. This pursuit of explainable AI has been criticized in the recent AI/ML literature and called "problematic" (see, for example [206]), mainly because such post-hoc explanations are often not reliable and can be misleading [97, 137]. An alternative approach that has been advocated in [206] argues for using learning models such as linear models or DTs that are inherently (*i.e.,* ex-ante) interpretable.

Unfortunately, because of the rich semantic content of the data in the network security domain, uncovering the types of patterns in the data that matter has become increasingly the responsibility of trained "black-box" models rather than painstakingly-designed inherently explainable models. However, instead of considering this development as being "problematic," we view it as an unique opportunity to ultimately achieve the vision of interpretable ML, ensuring that AI/ML models used for high stakes decision-making are fully comprehensible by their end users and interested third parties.

Explainable AI: Post-hoc Interpretability. A commonly-made argument in favor of using black-box models such as deep neural networks or random forests is that they typically achieve higher accuracy compared to their interpretable counterparts (*e.g.,* DTs) and are therefore often more desirable when used in practice. Although this argument is not universally shared (*e.g.,* see [206]), it nevertheless has been a driving force behind the recent efforts on the topic of "explainable AI." Also referred to XAI [233], explainable AI describes efforts where the development of a trained black-box model is followed up with additional activities that are intended to help "explain" the originally obtained black-box model. These efforts can be divided into two disjoint categories, namely *local explainability* and *global explainability*.

Methods for providing local explanations aim at illuminating how a black-box model

makes individual decisions (or decisions in a local region near a particular data point) and include well-known techniques such as LIME [199], SHAP [153], and LEMNA [106]. Since these methods limit their attention to only a subset of individual decisions, they are prone to providing misleading explanations [145, 168, 259], depending on the subset of samples analyzed. Related methods such as Partial Dependence Plots (PDP) [90] and Accumulated Local Effect (ALE) plots [14] suffer from similar shortcomings. As such, these methods are of limited use when we seek explanations that we can trust in the sense that they accurately describe how a given black-box model makes decisions holistically.

In turn, methods that provide global explanations aim at describing how a given black-box model makes its decisions "as a whole" and not one data sample at a time. Extracted from the black-box model in a second step (*i.e.,* post-hoc), such explanations typically take the form of an inherently interpretable model such as a rule set or a DT [24, 138] and become the main vehicles for studying the decision-making process of the original black-box model and examining its properties. However, existing approaches for such post-hoc extractions of global explanations are known to produce at times too low of a fidelity to be useful in practice [24], target only a very specific set of black-box models [25], be difficult to reproduce [137, 138], and be possibly unreliable to the point of being misleading [97, 137]. To achieve the level of explainability required in high-stakes application domains such as network security, we seek to generate high-fidelity global explanations that are capable of accurately and faithfully describing a majority of the decisions made by any given black-box model.

2.3 Trustee Overview

Our focus in this paper is on post-hoc global model interpretability for the application domain of network security problems. The idea of using DTs for investigating global

model interpretability for a given black-box model is not novel. However, the set of requirements that we impose on the DT explanations is non-standard and makes this a challenging problem, which motivated us to develop Trustee.

For one, we require that our new DT extraction method be model-agnostic; that is, applicable to any given black-box model. Second, we also demand that the method produces high-fidelity DT explanations; that is, DTs whose expected predictive performance is similar to that of the black-box model. To quantify the fidelity of DTs, we rely on well-known metrics; for example, while for classification problems we measure fidelity using the F1-score between classifications from the black-box model and the DT, for regression problems, we measure fidelity in terms of the R-squared value between the predictions from the black-box and the DT. The third requirement we impose is that the extraction method also results in low-complexity DT explanations such that selected parts of the DTs are intelligible and comprehensible (*i.e.,* easy to understand by domain experts) and accurately describe how the black-box model makes most of its decisions. The fourth and last requirement concerns a stability property that we want our new DT extraction method to have. In particular, to reduce the chances that this output provides a misleading DT explanation, we require that most of the final DT's decisions should be insensitive to the minute details of how this final DT explanation has been determined.

For a DT that Trustee extracts from a given black-box model and satisfies this set of requirements, our next goal is to summarize the pertinent aspects of this synthesized tree in a trust report. This trust report is intended to help end users determine whether the given black-box model suffers from the problem of underspecification and cannot be trusted. To achieve this goal, we look for ways to exploit the extracted DT explanation for the purpose of enabling the end users to investigate the black-box model for likely indications of the presence of inductive biases. In particular, in this paper we consider the following three instances of inductive biases: *(i)* instances of shortcut learning, *(ii)*

signs of spurious correlations, and *(iii)* problems with out-of-distribution samples.

Note that the presence of any of these inductive biases proves that the given black-box model suffers from the underspecification problem and cannot be trusted. At the same time, the absence of these instances does not mean that the black-box model can be trusted. In fact, while proving for an arbitrary black-box model that the model does not suffer from the underspecification problem is hard and remains an unsolved problem, showing that the model does suffer from the underspecification problem only requires demonstrating the presence of a single instance of an inductive bias, and our design of Trustee is an initial effort that simplifies demonstrating that certain biases are present in a given model. After describing Trustee's design in detail in Section 2.4, we illustrate the end-to-end application of Trustee, including the use of the extracted DT explanation and resulting trust report with a number of illustrative examples in Section 2.7.

2.4 Extracting Decision Trees

The first step to realize the agenda detailed in Section 2.3 consists of generating high-fidelity and inherently interpretable (*i.e.,* "white-box") counterparts for any given black-box model, regardless of the learning method used by the black-box. To this end, we first discuss existing approaches to this problem and their limitations. We then present Trustee, an original and practical framework that end users can apply to extract high-fidelity DT explanations from an arbitrary black-box learning model.

2.4.1 Existing approaches

Global white-box explanations extracted from a black-box model can often describe in detail the reasoning behind the model's behavior, provided they achieve a good enough fidelity. Earlier works [24, 25, 61, 162] have proposed different approaches to extracting

DT explanations from black-box models, but these DTs typically do not satisfy all the above-listed requirements and are therefore ill-suited for end users who want to gauge their level of trust in a given black-box model. We list a number of relevant prior efforts and their pertinent features in Table 2.1. Note that some of these existing methods [25, 162] are not model-agnostic but have been designed for specific learning paradigms and models, such as Reinforcement Learning. As such, they typically rely on assumptions that are specific to the learning paradigm or model that their designs focus on. On the other hand, prior efforts that do propose model-agnostic approaches [24, 61] tend to produce DT explanations that don't satisfy the fidelity requirement that we demand for realizing our objective (see our technical report [124] for empirical evidence).

Table 2.1: Existing approaches to extract decision trees.

Method	Optimization Objective	Stopping Criterion	Model Agnostic	High Fidelity	Domain-specific Pruning
Trepan [61]	Fidelity	Max Nodes	✓	-	-
dtextract [24]	Accuracy	Max Nodes	✓	-	-
VIPER [25]	RL Reward	Max Iterations	-	-	-
Metis [162]	RL Reward	Max Iterations	-	-	-
Trustee	Fidelity	Max Iterations	✓	✓	✓

Another important aspect of many of these existing efforts is the stopping criterion they use to obtain their extracted DT explanations. For example, prior efforts such as [24, 61] require specifying the maximum size (*i.e.,* number of nodes) that the extracted DT can have and use this input parameter as stopping criterion. Such approaches are convenient for obtaining explanations that are guaranteed to be of a certain size, but this convenience typically comes at the cost of low fidelity, implying that important decision-making rules may be missing from the resulting DT. Other methods such as [25] and [162] extract DT explanations in an iterative fashion, require specifying the maximum number of iterations, and use this user-specified input parameter as stopping criterion. Even though these meth-

ods do not explicitly optimize for fidelity, they typically produce high-fidelity explanations, but at the cost of high complexity (*i.e.,* the large size of the resulting explanations makes interpreting cumbersome if not impractical). To overcome this problem, the authors of [162] rely on a commonly-used technique called Cost-Complexity Pruning (CCP) [90]. Similar to other pruning methods [84], CCP succeeds in striking a balance between the overall fidelity of the extracted DTs and their size. However, from an interpretability perspective, CCP tends to be oblivious to what role each decision-making rule plays as part of the resulting DT explanations. Because of this observed trade-off between model complexity and model interpretability, these methods are ill-suited for our purpose where we strive to shed light on the decision-making rules that are key to interpreting black-box models that arise in the context of solving network security-related problems.

2.4.2 Model-Agnostic Decision Tree Extraction

Given the absence of readily available model-agnostic methods for extracting high-fidelity, low-complexity, and stable DTs from black-box ML models, we present in the following Trustee. Algorithm 1 describes the steps that Trustee takes to achieve its objective.

At a high level, these steps are executed as part of an outer loop (lines 4-16) that is executed a total of S times. Each iteration of this outer loop involves an inner loop (Lines 5-12) that is performed N times. Here, this inner loop is designed to generate different high-fidelity DT explanations, one per iteration. It does so by applying a teacher-student dynamic derived from imitation learning [120] that uses π^* as an oracle in conjunction with a carefully curated dataset $\mathcal{D}$' to guide the training of a surrogate "white-box" model in the form of a DT that imitates the black-box's decisions. In contrast, the purpose of the outer loop is *(i)* to select from among the N high-fidelity DTs that have been

generated in the process of executing the inner loop the DT with the highest fidelity, *(ii)* to transform this resulting DT into a high-fidelity and low-complexity DT by means of a post-processing step that consists of applying a purposefully-developed pruning method (see Section 2.5 for details), and *(iii)* to consider all S high-fidelity and low-complexity DTs that have been generated in the process of executing the outer loop and output the one that is the most stable in the sense of having the highest mean agreement among these S DTs.

Algorithm 1 takes as input a given black-box model π^* that we desire to explain and the original dataset $\mathcal{D}_0$ that was used to train π^*. Other parameters that the algorithm requires as input are the number of iterations S for the outer loop, the number of iterations N for the inner loop, the number of samples M to select from $\mathcal{D}_0$ to use when training the surrogate DTs as part of each iteration of the inner loop, and a parameter k that is required by the tree pruning method used in Line 14 and is discussed in more detail in Section 2.5 below. The algorithm starts by initializing the training dataset $\mathcal{D}$ (Line 2) using the given black-box π^* to predict the expected outcomes from the given input data $\mathcal{D}_0$. It then initializes a set of DTs (Line 3) from which, at the end (Line 17), the most stable DT explanation will be selected and returned as output by Trustee (Line 18).

To execute the inner loop as part of an iteration of the outer loop, the steps that the algorithm performs during the j-th ($1 \leq j \leq N$) iteration of the inner loop consist of *(i)* selecting M training samples uniformly at random from the optimal prediction dataset $\mathcal{D}$ (Line 6) to initialize a training dataset $\mathcal{D}'$; *(ii)* splitting the dataset $\mathcal{D}'$ for training and testing (Line 7); *(iii)* training a DT student $\hat{\pi}_i$ on $\mathcal{D}'_{train}$ (Line 8) by using the well-known CART method [38]; *(iv)* testing the DT explanation using $\mathcal{D}'_{test}$, collecting the samples that the DT classifier wrongly classifies into the set $\mathcal{D}'_e$ (Line 9), and querying the black-box model for the expected results for $\mathcal{D}'_e$ (Line 7) to produce a correction dataset $\mathcal{D}_j$ (Line 10); and *(v)* augmenting the optimal dataset $\mathcal{D}'$ with this correction dataset (Line

11) to reinforce the correct decisions during the subsequent iterations of the inner loop.

The steps that the algorithm executes during the i-th ($1 \leq i \leq S$) iteration of the outer loop are *(i)* perform N iterations of the inner loop (Lines 6-11), *(ii)* select from among the N generated different student models the one DT explanation with the highest fidelity (Line 13), and *(iii)* apply a special pruning method to this highest-fidelity DT to obtain a high-fidelity and low-complexity DT explanation candidate. Finally, after S iterations of this outer loop, the algorithm selects from among the S obtained different high-fidelity and low-complexity DT explanation candidates the one that has the highest mean agreement (*i.e.,* is the most stable) and returns this "best of the best" DT as final output of Trustee.

In the following, we provide a more detailed description of the main design choices we made for Trustee and further evaluate some of these design choices as part of an ablation study in Section 2.8.

Multiple iterations and uniform sub-sampling. The CART algorithm that is traditionally used to train DT models relies on a greedy approach for finding the best splits in the given training dataset. This greedy approach ensures that for a given training dataset, the resulting DT will be largely insensitive to the order in which the input samples are processed. At the same time, this greedy approach is prone to produce over-fitted DTs [37]. While using this approach without further constraints (*e.g.,* stopping criterion) to train a DT results in perfect fidelity, being over-fitted makes the resulting DT ill-suited for providing an intelligible explanation for how the given black-box model makes its decisions. Instead, the resulting DT explanation is largely an artifact of the method used to generate the explanation. To overcome this problem, Trustee implements an iterative approach to train multiple student models on the expert model predictions. This iterative approach is implemented as the inner loop in Algorithm 1, where at each iteration, we select a fraction M of the input data by uniform sub-sampling from the original training

Algorithm 1 Model agnostic decision tree explanation extraction.

procedure TRUSTEE(
π^*: Black-box model,
$\mathcal{D}_0$: Initial training dataset,
M: Number of samples to train the decision tree,
N: Number of iterations of inner loop,
S: Number of iterations of outer loop,
k: Parameter for Top-k Pruning),
  Initialize dataset using black-box $\mathcal{D} \leftarrow \pi^*(\forall x \in \mathcal{D}_0)$
  Initialize stabilization set of DTs $\mathcal{R} \leftarrow \emptyset$
  for $i \leftarrow 1 \dots S$ **do**
    for $j \leftarrow 1 \dots N$ **do**
      Sample M training cases uniformly from $\mathcal{D}$
      Split sampled dataset for training and testing
        $\mathcal{D}'_{train}, \mathcal{D}'_{test} \leftarrow$ TRAINTESTSPLIT($\mathcal{D}'$)
      Train DT
        $\hat{\pi}_j \leftarrow$ TRAINDECISIONTREE($\mathcal{D}'_{train}$)
      Test and get samples DT misclassifies
        $\mathcal{D}'_e \leftarrow \{\forall (x, y) \in \mathcal{D}'_{test} \mid \hat{\pi}_j(x) \neq \pi^*(x)\}$
      Get correct outcome from black-box
        $\mathcal{D}_j \leftarrow \pi^*(\forall x \in \mathcal{D}'_e)$
      Augment dataset $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_j$
    end for
    Select tree with highest fidelity
      $\hat{\pi}_{max} \leftarrow \hat{\pi} \in \{\hat{\pi}_1, ..., \hat{\pi}_N\}$
    Prune selected tree $\hat{\pi}_i \leftarrow$ TOPKPRUNE($\hat{\pi}_{max}$, k)
    Add tree to the stabilization set $\mathcal{R} \leftarrow \mathcal{R} \cup \hat{\pi}_i$
  end for
  Select tree with highest mean agreement with others
    $\hat{\pi}_{agree} \leftarrow \hat{\pi} \in \mathcal{R}$
  return $\hat{\pi}_{agree}$
end procedure

dataset (Lines 5-11). This approach differs from existing efforts [25] in that by requiring the uniform sub-sampling step at each iteration, we ensure that each DT explanation will have only a limited view of the entire data, akin to a k-fold cross validation [207]. Incorporating this sub-sampling step allows us to stress-test how different features and/or feature values contribute to the decision-making of the black-box model and then select the ones that best fit our overall objective. In practical terms, uniform sub-sampling from the original training dataset assumes each sample has the same probability of being selected (*i.e.,* balanced dataset). While it is well known that using imbalanced datasets to train ML models leads to biases towards the majority classes, the existing ML literature provides several approaches that resolve this problem through proper pre-processing of the original training data [131].

Dataset augmentation. An important design choice for Trustee involves a dataset augmentation step (Line 9), where in each iteration of the inner loop, the algorithm uses the optimal predictions from the black-box model on the sampled dataset D' to augment the original training dataset D. The purpose of this step is to over-correct for data samples for which the student DT model makes wrong decisions. Leveraging results from the existing literature on imitation learning [25, 203], performing this step can not only increase the overall accuracy of the trained student model but also reduce the overall number of leaf nodes in the resulting tree.

Fidelity as objective function. When selecting from among the different student models that Trustee extracts from a given black-box model, it uses model fidelity as objective function and picks the student model with the highest fidelity (Line 11). This design choice implies that while the final DT explanation produced by Trustee is not necessarily the most *accurate* DT for the given classification problem, it is the DT that is the most *faithful* one in terms of explaining how the black-box model makes its decisions. Intuitively, it is by insisting on this high-fidelity aspect of the DTs that Trustee considers

that we are able to post-process the resulting DT explanation in ways that will help end users with varying degrees of domain knowledge to gauge their trust in the given black-box model. We provide evidence in support of this intuition in Section 5 where we describe the type of post-processing that we perform as part of Trustee so the final DT explanation it outputs can serve as an inherently practical means for faithfully explaining most of the given black-box model's decisions.

Model stability. Since the inner loop of Algorithm 1 (Lines 6-11) uses a different random subset of the entire dataset each time it trains a DT explanation, it is possible for Trustee to output a misleading explanation because of the particular subset of data that was used to train that final DT explanation. To minimize the chances for such scenarios to occur, we add an outer loop in Algorithm 1 (Lines 4-16). This addition results in the extraction of S different high-quality DT explanations from the given black-box model and allows us to measure the agreement among these S different DT explanations. The agreement of DTs is a well-known measure of how often a pair of DTs will make the same decisions for the same input data and is a metric that has been used in previous studies that concern assessing the stability of different DTs [105, 234]. Here, to select the final DT explanation that is returned as output of Trustee (Line 18), Line 17 in Algorithm 1 computes the pair-wise agreement among all S DT explanations and selects the one with highest mean agreement. While implementing this outer loop prevents Trustee from generating obviously misleading explanations and gives domain experts confidence that they can trust the explanations produced by Trustee's output, rigorously proving that a "white-box" model extracted from a given black-box model does not provide misleading explanations is an active area of current research [137].

2.5 Processing Decision Trees

When using Trustee to synthesize a high-fidelity DT explanation for a given black-box model, realizing the agenda outlined in Section 2.3 requires performing an additional step in Algorithm 1 (Line 14) each time N iterations of the inner loop have completed and the algorithm has selected the highest-fidelity DT from among the resulting N different high-fidelity DT candidates (Line 13). The purpose of this step is to transform this selected highest-fidelity DT into a DT explanation for the given black-box model that is inherently practical in the sense of having low complexity and at the same time high fidelity. Here, low complexity of a DT explanation refers to small trees but also, and more importantly, trees whose main branches (*i.e.,* decision rules ranked by number of input samples they classify) explicitly, intelligibly, and accurately describe how the black-box model makes most of its decision. Effectively, when generating this low-complexity and high-fidelity DT explanation as a result of this post-processing step, we tolerate some loss of fidelity in return for achieving low complexity. In the following, we examine different aspects of this fidelity-complexity trade-off and introduce a simple tree pruning method that we call *Top-k Pruning* and that comprises the required post-processing step. We design this method for the explicit purpose of ensuring that any final DT explanation that Trustee outputs can be readily processed and understood by domain experts.

2.5.1 Decision Tree Pruning: Trade-offs

One of the main disadvantages of CART models is that CART's greedy algorithm is known to be prone to overfitting, often producing high-fidelity DTs that can have thousands of nodes [84]. Clearly, such large trees are detrimental to our ultimate goal; that is, presenting end users with inherently practical explanations that they can readily inspect and understand with their available domain knowledge. In designing Trustee, we similarly

focused on first obtaining largely unconstrained DT explanations with the best possible fidelity (Line 13). However, our reasoning for doing so is that we explicitly require that any DT explanation that Trustee outputs will have undergone a post-processing phase for the purpose of making this final DT explanation intelligible and comprehensible for end users. Our intuition behind obtaining a high-fidelity DT explanation first and addressing its complexity later is that manipulating a high-fidelity explanation with an eye towards reducing its complexity is more likely to result in explanations that, while experiencing some decrease in their fidelity, still will have higher fidelity than their counterparts that had lower fidelity to start with.

A commonly-used approach to transforming large DTs into trees of smaller sizes is pruning, and the existing literature describes several pruning methods for DTs [84, 90], many of which are highly effective in obtaining DTs that have small complexity, at least as far as the sizes of the trees are concerned. Among the most widely-used approaches to pruning are *(i)* pre-pruning which limits either the total number of nodes or the overall depth of the tree and *(ii)* post-pruning, such as Cost-Complexity Pruning (CCP) [90]. On the one hand, by explicitly constraining either the number of tree nodes or the tree's depth, the pre-pruning approaches allow for direct control over the size of the resulting DT. However, this control over tree size typically comes at the cost of reduced fidelity, mainly because the obtained small trees run the risk of missing important decision branches as they prevent the consideration of any further decision branches once the stopping criterion is reached. On the other hand, using post-pruning such as CCP often results in a better trade-off between fidelity and size. However, this better trade-off comes at the cost of reduced interpretability, mainly because CCP relies on a parameter that is indirectly responsible for how many nodes of a tree get pruned. Lack of a more direct control makes it difficult to decide which tree branches to include in the pruned tree and which to exclude.

2.5.2 Top-k Pruning Method

Each branch in any of the highest-fidelity DTs that are selected in the process of executing an iteration of Algorithm 1's outer loop (*i.e.,* Line 13) represents a decision "rule"; that is, a combination of individual decisions on features that results in labeling the input data as belonging to a specific class (*e.g.,* malware vs. benign). Since each of these "rules" accounts for a certain percentage of all samples in the input dataset, the different rules also contribute differently to the overall model fidelity, and as the complexity of the DT grows (*i.e.,* larger number of branches), so does the DT's fidelity.

The idea behind our Top-k Pruning method is that to detect signs of the presence of inductive biases in a given black-box model, it often suffices to carefully scrutinize only the top-k branches of an extracted high-fidelity DT, ranked by the number of input samples a branch classifies, especially in cases where the branches intelligibly describe how the black-box makes most of its decisions. In particular, we argue that the "tail" end of the branches of an extracted high-fidelity DT (*i.e.,* branches that are not in the top-k for some large value of k) often reflects specific decisions of the black-box model that are overfitted to the training dataset and can, for all practical purposes, be ignored when trying to explain the most important decisions of the black-box model. However, since the trade-offs between model fidelity and model complexity are typically model dependent,Top-k Pruning requires a parameter k, and specifying a value for k gives end users complete control over how many branches they want to consider in their attempt to understand the trade-offs for a given model. Also note that even though selecting smaller k values can possibly result in poor fidelity, it does not mean that we cannot draw potentially important conclusions from the resulting explanation. For instance, one specific branch can sometimes cover most samples of a particular class, resulting in an apparent poor overall fidelity but still indicating a potential underspecification issue

related to that specific class. In short, the user-specified parameter k determines the complexity of the final DT explanation that Trustee presents to the end user. If, at any point, a user wants to inspect more branches of the tree, they can simply choose a larger k and rerun our algorithm. Note, however, that due to its probabilistic nature, re-running our algorithm will typically result in applying the Top-k Pruning method to a high-fidelity DT explanation that differs from the original one. We leave a careful investigation of this aspect of Trustee and its deeper implications for detecting instances of inductive biases in a given black-box model for future work.

2.5.3 Generating Trust Reports

We use the DT explanation that Trustee outputs as basis for populating a trust report that simplifies the task of end users of a given black-box model to gauge their trust in that model. In the following, we provide details on how we build this trust report so it helps end users spot signs that point to possible instances of inductive biases in the given black-box model. If upon further scrutiny of these signs the presence of such an inductive bias is confirmed, it would be proof for the end users that they cannot trust the given model. To this end, we leverage the fact that any DT explanation that Trustee outputs has been pruned with the help of our Top-k Pruning method and is therefore typically a small tree comprised of k branches. As part of the trust report, we present the details of the generated small DT to the end users so they can examine these details with an eye towards three common ways an underspecified ML model can be recognized. More precisely, we intend the trust report to be the main source of information that end users can consult when checking for inductive biases that manifest themselves as instances of shortcut learning strategies, through the presence of spurious correlations, or in an inability to generalize for realistic out-of-distribution data.

Importantly, by itself, the information contained in the trust report is in general insufficient to diagnose underspecification issues; instead, it points to potentially attention-worthy aspects of the model or the data that require further attention. As such, detecting and diagnosing underspecification issues is not a task that is currently automated but requires domain knowledge and great familiarity with the learning problem at hand. Consequently, the effort demands a (human) domain expert to actively inspect and check if the trust report-provided information points to possible problems in the data or in the model, or indicates that there are no problems with either the model or the data. In the following, we briefly describe how the generated trust report helps with checking for each of these three inductive biases.

Shortcut learning. Presenting a visual depiction of the small DT explanation that forms the output of Trustee and annotating it with pertinent information (*e.g.,* features used, splitting conditions or clauses present at the different nodes of the tree, number of input samples associated with each branch segment) allows for quick and intelligible perusing and inspection of the tree. In particular, observing that less than a handful of input features are required to accurately classify most of the input data (or a specific class of input samples) is often a strong indication of a shortcut that the black-box model learned and that can in general quickly be confirmed with readily available domain knowledge. Note however that a small number of features in the output of Trustee may also indicate that the learning problem for which the black-box was designed for in the first place is in fact simple and may not require any ML at all.

Spurious correlations. A more involved investigation of the information provided in the trust report concerns studying the impact of removing the identified most important feature(s) from the provided dataset. Upon removing such feature(s), we can then retrain the black-box model using this altered dataset, proceed to use Trustee to extract a new DT explanation, and repeat this process a number of different times. In general, the impact of

removing important features from the training dataset is that the accuracy of the black-box model decreases. However, especially in situations where the data include a large number of features, it is often the case that the black-box model is able to find alternative features so that removing the most important feature(s) leaves the overall model accuracy essentially unchanged. We take this as a strong indication of the presence of spurious correlations in the data that can subsequently be easily confirmed via an analysis of the original feature set.

Out-of-distribution samples. The annotated version of the output of Trustee that is shown as part of the trust report can also be used to uncover the individual features in each of the most important tree branches, allowing us to plot the distribution of the values that each of those features can take in the provided dataset. Inspecting the resulting distributions affords end users an opportunity to reason whether or not the observed distributions of feature values are consistent with those encountered in data collected from actual production settings. Such an inspection is especially informative when the provided datasets consist of network traffic measurements, where feature value distributions are typically dictated by the dominant protocols in use, where artifacts can often be easily identified (*e.g.,* due to simple testbed experimentations), and where generating meaningful out-of-distribution samples is in general feasible because the expected behavior across the full TCP/IP protocol stack is either known or well documented.

2.6 Using Trustee in Practice

The following step-by-step instructions are intended to help end users who want to use Trustee and associated trust reports to check if a given trained ML model is credible or not by inspecting it for possible underspecification issues.

Step 1 (Getting started): Select the ML model that needs to be analyzed and the dataset of the input samples that is used to examine the model's decisions and

decision-making process. The only requirement for the selected ML model is that it provides a `predict` interface that Trustee can use to query the model for its prediction for a given input sample. As for the input dataset, Trustee also accepts datasets that differ from the dataset used to train the ML model, but we recommend using the training dataset for a basic analysis of the selected ML model.

Step 2 (Selecting hyperparameters): Trustee requires selecting values for four hyperparameters: S and N (number of iterations of the outer loop and inner loop in Algorithm 1, respectively), M (sampling rate), and k (number of branches to keep as part of our Top-k Pruning method). Although highly model- and data-dependent, we found that in the context of the different use cases we analyzed, choosing $S = 10$, $N = 50$, $M = 30\%$ of the input dataset, and $k = 10$ is a good starting point and allows for subsequent modifications should the need arise. In general, we recommend selecting suitable values for M, S,and N by qualitatively comparing the learning curves [207, Section 18.3.3] and checking the DT fidelity between training and test data for different hyperparameter values, but more quantitative methods such as grid search [31] or Bayesian Optimization [224] could be used as well. However, the number of samples M is highly dependant on the size and type of the available data; setting a sampling rate too high increases the risk of over-fitting, setting the sampling rate too low increases the risk of under-fitting and is likely to require a higher number S of iterations of the outer loop of Algorithm 1. In turn, selecting the parameter k (*i.e.,* number of top-k branches) depends first and foremost on the amount of information a domain expert is willing to analyze when presented with Trustee's output.

Step 3 (Detecting underspecification issues): Identifying the presence and/or nature of underspecification issues in a model currently requires manual inspection by a domain expert. The main vehicle for performing this manual task is the DT explanation produced by Trustee in conjunction with the information provided in the corresponding

trust report. Relying on basic trust report-provided information can often point to potential biases, but it typically invites further scrutiny at the level of individual decision rules (*i.e.,* tree branches) where, for example, certain deficiencies in the training dataset (*e.g.,* missing samples of real-world patterns or behavior) can be identified.

Step 4 (Validating DT explanations): Validating a DT explanation that forms the output of Trustee typically requires tinkering with the ML model itself, with the feature engineering as part of the model's design, or with the provided dataset. Unfortunately, a general inability to easily collect new or different data severely limits the validation efforts that require modified data. In such cases, we found that tampering with the original data (*e.g.,* removing certain features or artificially modifying packet headers in a trace) can be a viable option but has to be done with care to ensure that the tampered dataset consists of realistic input samples that the ML model ought to be able to handle.

2.7 Results

In this section, we illustrate with different use cases how Trustee can be used in practice. Each use case concerns a recently published black-box model that has been developed for a particular network security-related problem and is accompanied by open-sourced artifacts (*e.g.,* code base, dataset) that are required for reproducing the reported findings and assessing whether the ML model is credible.

2.7.1 Summary

Table 2.2 summarizes the use cases we analyze. The first use case (§2.7.2) illustrates how an apparently high-performant neural network learns simple shortcuts to distinguish between two types of traffic (VPN vs. Non-VPN). It highlights the importance of having an in-depth understanding of the data used to train a model. The second use case (§2.7.3)

Table 2.2: Case Studies.

Analyzed in	Problem	Dataset(s)	Model(s)	Trustee Fidelity	Type of inferred inductive bias
Section 2.7.2	Detect VPN traffic	ISCX VPN-nonVPN [77]	1-D CNN [243]	1.00	Shortcut learning
Section 2.7.3	Detect Heartbleed traffic	CIC-IDS-2017 [214]	RF Classifier [214]	0.99	Out-of-distribution samples
Section 2.7.4	Detect Malicious traffic (IDS)	CIC-IDS-2017 [214], Campus dataset	nPrintML [118]	0.99	Spurious correlations
Tech Report [124]	Anomaly Detection	Mirai dataset [166]	Kitsune [166]	0.99	Out-of-distribution samples
Tech Report [124]	OS Fingerprinting	CIC-IDS-2017 [214]	nPrintML [118]	0.99	Potential out-of-distribution samples
Tech Report [124]	IoT Device Fingerprinting	UNSW-IoT [222]	Iisy [253]	0.99	Likely shortcut learning
Tech Report [124]	Adaptive Bit-rate	HSDPA Norway [201]	Pensieve [158]	0.99	Potential out-of-distribution samples

analyzes a black-box model trained using a synthetic dataset and shows that the developed model is vulnerable to o.o.d. samples. This use case cautions against an over-reliance on synthetic datasets because they often include measurement artifacts that commonly-considered black-box models exploit to achieve high accuracy. The third use case (§2.7.4) analyzes a recent approach that advocates using bit-level feature representations of the input data instead of carefully engineered and semantically meaningful features [118]. It shows that the indiscriminate use of the high-dimensional feature spaces that result from such representations can be problematic because it allows black-box models to identify and exploit spurious correlations between features to achieve high accuracy. The remaining use cases listed in Table 2.2 are analyzed in detail in our technical report [124]. All relevant research artifacts are publicly available at [125].

2.7.2 Detecting VPN vs. non-VPN Traffic

Problem setup. We consider the paper [243], which presents an AI/ML-based framework for encrypted traffic classification that integrates feature design, feature extraction, and feature selection. It uses one-dimensional convolutional neural networks (1D-CNN) to automatically learn the relationships between raw packets and the output labels. For classifying VPN vs. Non-VPN traffic, the authors train a 1D-CNN learning model with the PCAPs of the ISCX VPN-nonVPN dataset [77], treating the packets of

each session as a 2D image of size 28x28. As a result, the proposed model views the input traffic samples as discrete byte streams of fixed length (*i.e.,* 784 bytes) and treats each byte as a "feature." The paper [243] reports outstanding performance (*i.e.,* 100% (99.9%) precision and 99.9% (100%) recall for Non-VPN (VPN) traffic). All AI/ML research artifacts [243] and datasets [77] are available online, allowing full reproducibility of the described models and reported findings.

Explanation. We first reproduced the black-box model (*i.e.,* 1D-CNN) and the results presented in [243, Table VI] for classifying VPN vs. Non-VPN traffic. Next, we used Trustee to extract a DT from the black-box 1D-CNN model (Figure 2.2) and note that due to the small tree sizes, there was no need for Trustee to apply the Top-k Pruning method. To assess how well the extracted DT reproduces the black-box model, we used it to classify the test cases from [243] and compared the results with the classification from the black-box, measuring precision, recall, and F1. To our surprise, this simple and small white-box model accurately reproduced all black-box decisions, achieving a perfect F1-score.

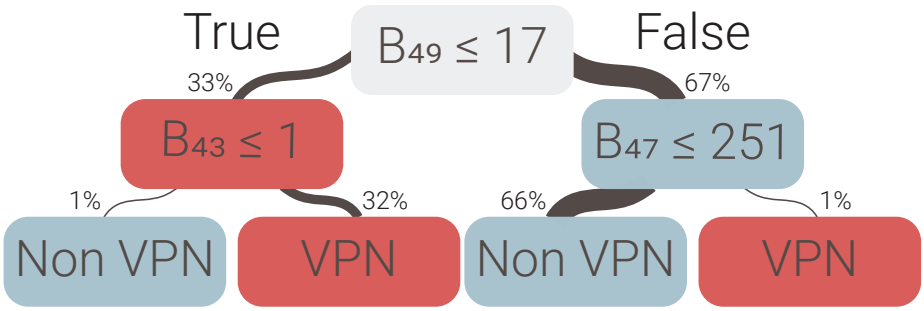


Figure 2.2: Decision tree for 1D-CNN model. The percentage of samples that follow each branch is presented above each node. Line widths are proportional to the percentage of samples.

Correctly interpreting this extracted DT requires understanding the structure of the input data. Because the DT makes a decision based only on three bytes in the initial

segment of each input sample (*i.e.,* bytes B_{49}, B_{43}, and B_{47}), we analyzed samples of VPN and Non-VPN test cases to uncover the "meaning" of those bytes. Figure 2.3 shows a schematic view of the first 80 bytes of actual input data used in [243]. We notice that each input sample consists of an initial set of bytes representing PCAP metadata, Ethernet header, and IP header. Importantly, none of these initial bytes say anything about actual VPN or Non-VPN traffic.

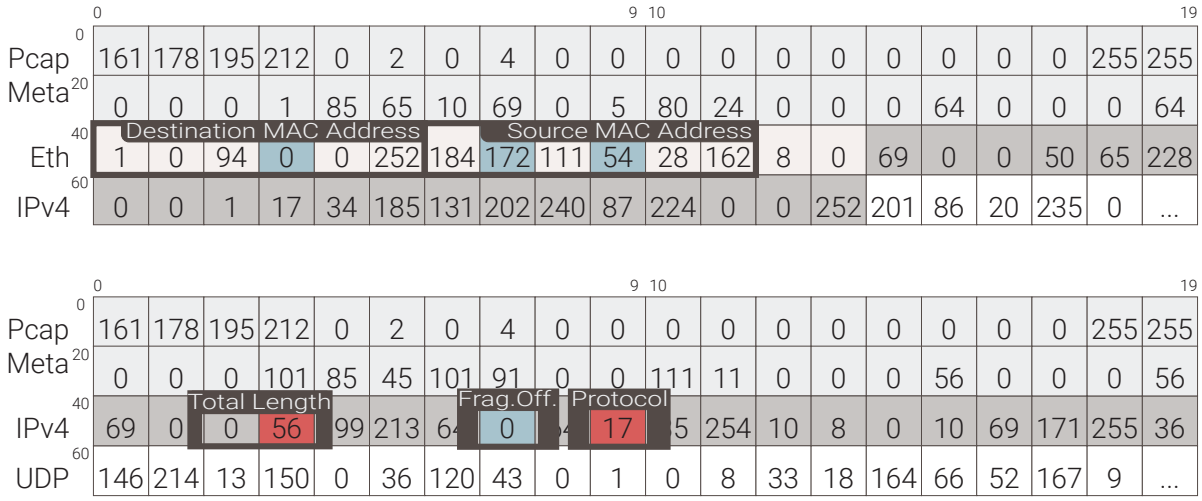


Figure 2.3: First 80 bytes from the training dataset.

Upon further scrutiny of the public dataset [77], we noticed that Non-VPN traffic samples always contain Ethernet headers while roughly 90% of the VPN traffic samples do not (Figure 2.3). Thus, if B_k denotes the byte in position k, then for $k \geq 40$, there is a misalignment in the features of the two types of traffic, resulting in completely different semantics for the byte k. In Figure 2.2, we see that the DT uses feature B_{49} as the splitting criterion at the root node. Due to the feature misalignment, B_{49} is the IPv4 protocol field in VPN samples or the fourth byte of the Ethernet source address in Non-VPN samples. Because the VPN traffic in the dataset uses either UDP ($B_{49} = 17$) or TCP ($B_{49} = 6$), the root node of the DT splits almost all the samples by comparing the IP protocol field

in the VPN traffic with a random byte of the Ethernet addresses of the machines used to generate the Non-VPN traffic trace, making feature B_{49} a classical "shortcut" to classify the traffic. However, the split is not perfect because, coincidentally, two machines used for generating Non-VPN traces had the fourth byte of their Ethernet source addresses less than or equal to 17 (`54:9f:35:`**`0d`**`:e9:c2` and `2c:44:fd:`**`02`**`:16:ef`).

The left branch of the DT classifies most samples as VPNs. However, to weed out a few remaining samples of Non-VPN traffic, the DT uses feature B_{43}. In this case, B_{43} corresponds to the Total Length IP field in most VPN samples or the fourth byte of the Ethernet destination address in Non-VPN samples. Once again, the black-box model takes a shortcut to distinguish between the two classes. A similar analysis applies to the right branch, which classifies most samples as Non-VPN and uses B_{47} (which appears to be a spurious correlation in this case) to weed out the few VPN samples.

Validation. Even though the DT extracted by Trustee is a high-fidelity proxy for the 1D-CNN black-box model, it is unreasonable to expect that a simple 3-node structure encompasses the model's entire decision-making process. We verify this intuition by generating a tampered validation dataset for the black-box model. In particular, we changed bytes 43, 47, and 49 in the VPN samples to mimic random Non-VPN samples. By following the logic of the decision tree branches, the black-box model would mis-classify all VPN samples. The first two rows of Table 2.3 give the average precision, recall, and F1-score for both classes (VPN vs. Non-VPN) for original and tampered datasets. The results show that tampering with only these three features out of 748 had no significant impact on the classification accuracy of the black-box model. However, by performing detective work similar to the one described above, we observed that the black-box model succeeds in finding alternative "shortcuts" that are as easy to identify and explain as the one we described earlier.

To further demonstrate that the black-box model described in [243] and claimed

Table 2.3: Accuracy of black-box classifier.

Validation dataset	Avg. Precision	Avg. Recall	Avg. F1
Untampered	0.959	0.956	0.955
Tampered-43-47-49	0.959	0.956	0.955
Tampered-32-to-63	0.889	0.861	0.856
Tampered-0-to-63	0.831	0.757	0.734
Tampered-0-to-127	0.753	0.555	0.398

to be highly successful in learning to classify encrypted VPN and Non-VPN traffic is not a credible predictor, we tampered with entire ranges of bytes instead of individual bytes. As Table 2.3 shows, tampering with byte ranges of 32-64, 0-64, and 0-128 makes it increasingly more difficult for the black-box model to identify alternative shortcut predictors, and not surprisingly, the model's performance (*i.e.,* accuracy) gets worse and quickly reaches the point where, without being able to resort to shortcut learning (*i.e.,* randomly altering the first 128 bytes, which is less than 18% of the features), the model's performance becomes comparable to that of flipping a fair coin.

2.7.3 Detecting Heartbleed Traffic

Problem Setup. We consider the paper [214], which presents the public dataset CIC-IDS-2017 with labeled attack traces and lists publications that rely on this dataset to propose ML-based intrusion detection systems. The dataset contains traces of benign background traffic and 13 different attacks, such as Heartbleed, DDoS, and PortScans. The dataset also includes 78 pre-computed flow features, such as flow duration and mean Inter Arrival Time (IAT). Several research efforts report excellent classification results (*e.g.,* average precision and recall above 99% for all classes) of learning models trained on the pre-computed features of this dataset [43, 75, 80, 214, 259].

Explanation. We again started by reproducing the reported classification results

using the pre-computed features from the dataset to train a multi-class Random Forest Classifier to identify the 13 attacks and benign traffic, with a 75%-25% train-test split of the data. We could reproduce the excellent results reported by several publications, but, in doing so, we noticed that the dataset in question is highly imbalanced, having as few as 3 Heartbleed samples and as many as 680,000 DDoS samples in the 25% test split. Hence, we used a Random Over Sampler [131, 141] to produce a balanced training dataset to re-train the Random Forest Classifier and then used Trustee to extract a DT explanation. Without applying our Top-k pruning method, the high-fidelity DT extracted by Trustee from the classifier contained 899 nodes, making it largely impossible to understand the decision-making process of the black-box model. However, when running Trustee with the Top-k Pruning method and setting $k = 3$, we obtain the small-sized and therefore inherently manageable DT shown in Figure 2.4.

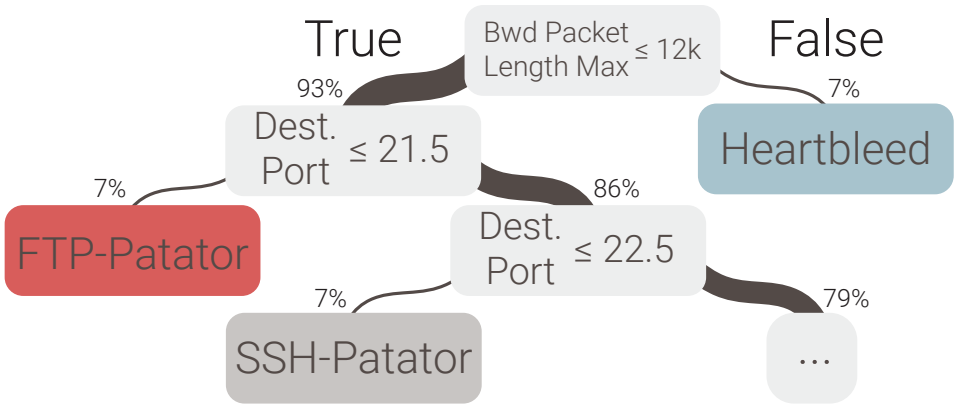


Figure 2.4: Decision tree for Random Forest Classifier.

Despite the likely shortcut the model takes by using TCP ports to classify SSH and FTP-Patator attacks, the root node of Figure 2.4 shows that the black-box model correctly classifies all samples of Heartbleed attacks based only on the maximum packet size of the victim server responses (*i.e.,* "Bwd Packet Length Max"). In Heartbleed, an attacker sends a TLS heartbeat message with a value in the size field that is bigger than the message.

A vulnerable server responds with a message with a size equal to the value specified in the size field and reviews information stored locally in its memory [79]. Prompted by this observation, we further inspect the DT to identify other features that appear as the most dominant features after we remove the "Bwd Packet Length Max" feature from the dataset. The results showed that the total backward inter-arrival time (*i.e.,* "Bwd IAT Total") also almost perfectly splits all Heartbleed samples. The distributions displayed in the *trust report* for both features (Figure 2.5) reveal a very telling pattern. To understand this behavior, we inspected the PCAP files and noticed that the TCP connections of the Heartbleed attacks were never closed between heartbeat messages, resulting in high values for the features "Bwd IAT Total" and "Bwd Packet Length Max".

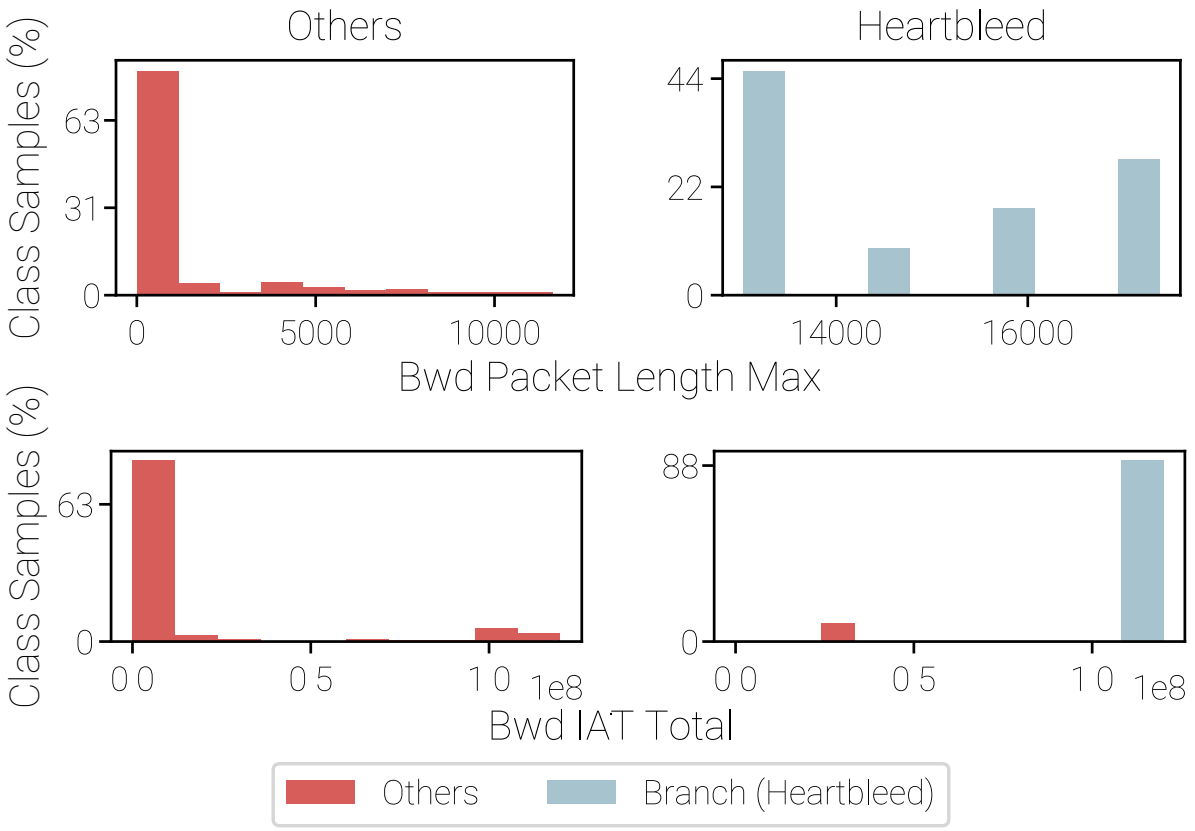


Figure 2.5: Data distribution of feature "Bwd Packet Length Max" (top) and "Bwd IAT Total" (bottom) comparing values in the Heartbleed class to all Others.

Validation. Considering that the dataset contained just one obvious pattern for the Heartbleed attack, it is not surprising that classifiers trained on this dataset have high accuracy when tested with i.i.d. samples. However, to demonstrate that a model is credible and generalizes as expected in deployment scenarios, we need to validate it with alternate but realistic test cases, *i.e.,* o.o.d. samples. We generated 1,000 new test cases of Heartbleed attacks using the same tool described in [214], but we closed the connection after the heartbeat request triggered a response with compromised data. This change resulted in Heartbleed flows with much smaller backward total IAT but with similar backward maximum packet length, as we use the same packet sizes as for the original trace. We then evaluated the Random Forest Classifier using the newly generated Heartbleed flows as test data. Table 2.4 shows that with just a simple change in the attack pattern, the classifier could not correctly classify a single one of the 1,000 new Heartbleed attacks, resulting in an F1-score of 0. This experiment demonstrates that the considered black-box learning model overfits on the i.i.d. cases, is not a credible predictor of realistic o.o.d. cases, and does not learn anything that readily available domain knowledge tells us about Heartbleed attacks.

Table 2.4: Black-box classifier's accuracy.

Class	Precision	Recall	F1
Heartbleed (i.i.d.)	1.000	1.000	1.000
Heartbleed (o.o.d.)	0.000	0.000	0.000

2.7.4 Inferring Malicious Traffic for IDS

Problem setup. We consider the paper [118], which proposes nPrint and the stable bit-level representation of network packets for automatically training learning models using AutoML [83]. The idea is to use a sequence of ordered features with values -1,

0, or 1 where each feature represents a bit of a set of pre-established protocol headers. The value -1 represents bits that are not present in a packet, while the values 1 and 0 are the actual values of present bits. The paper shows excellent results for an AutoML IDS model (called nPrintML) with 4,480 features trained using raw PCAP files from the CIC-IDS-2017 dataset [214].

Explanation. We successfully reproduced the reported results using the same configurations as those used in [118], obtaining a model with a 0.999 F1-score. To investigate this high-performance model, we used Trustee (with $k = 4$ for our Top-k Pruning method) to extract a high-fidelity (0.999) DT and show the top-4 branches in Figure 2.6. We can see that the top nodes rely on bits of the IP TTL field of the packets to separate the Benign class from the others. To understand the reason behind this observation, we inspect the description of the setup used to generate the CIC-IDS-2017 dataset. While all attacks were generated using hosts outside of the network in which the dataset was collected, the benign traffic was from hosts inside the network, creating a strong correlation between the packets' TTL value and traffic type. Also, most attacks were generated by a host running Kali Linux, which sets the initial value for TTL to 64 (*i.e.,* 00**1**00000). Similarly, the DDoS attack traffic was generated using a host running Windows 8.1, which sets the initial TTL value to 128 (*i.e.,* 0**1**000000). This setup where the traffic was generated makes it easy for the model to separate all DDoS attacks using only the second and third most significant bits of the TTL field.

We used the extracted DT to further investigate the model's behavior. We iteratively removed (assigned -1 to) the bits of the TTL field and other prominent features from the nPrint representation and retrained the model on the same dataset until the single *tcp_opt* field remained, representing bits of options of the TCP header. Given only these bits, the black-box nPrintML model still separates the attacks in the CIC-IDS-2017 dataset almost perfectly, reaching a F1-score of 0.990. In these cases, the DT explanations

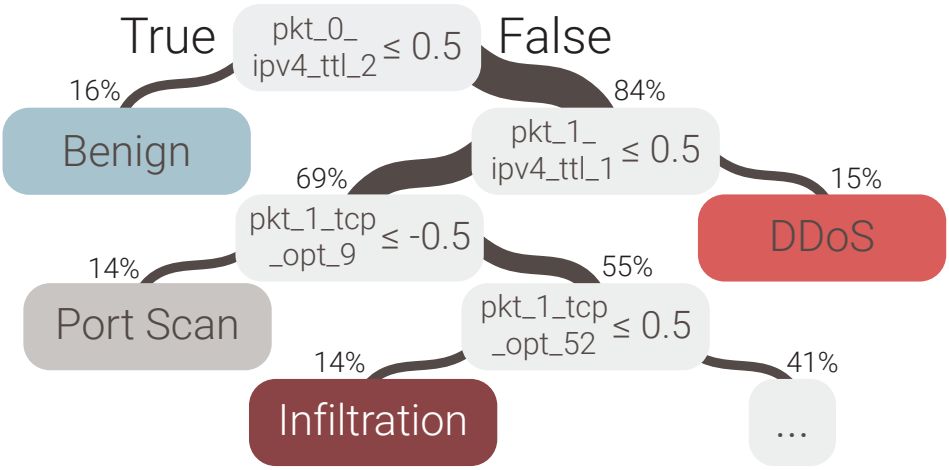


Figure 2.6: Decision tree for nPrintML IDS model.

produced by Trustee showed that the model still used single bits of packets to divide the traffic perfectly. These experiments demonstrate that the model succeeds in exploiting spurious correlations in the dataset, finding shortcuts due to the vast feature space where each bit is a feature. This issue is also known as the "curse of dimensionality" [207] and concerns cases where a model faces a high-dimensional feature space (*e.g.,* 4,480 features per sample in the case of nPrintML IDS) and not a diverse and dense enough data distribution to avoid occurrences of spurious correlations, which in turn a model can exploit to learn various shortcuts.

Validation. To examine the ability of the nPrintML IDS model to generalize to other deployment environments, we deployed the Suricata Intrusion Detection System [89] in the UCSB campus network and mirrored all the traffic before the firewall to produce a real-world dataset of network attacks. We captured about 12 hours of user traffic and the associated Suricata IDS alerts (see technical report [124] for details). We found 1,344 flows of DoS attempts. Also, we randomly sampled 1,366 port scan flows (out of 9 million) and 1,337 flows that didn't trigger any alert, which we labeled as benign traffic. Finally, we used nPrint to create a test dataset from that traffic to validate the trained model

of [118]. Table 2.5 shows the classification results of the model for the trace of our campus network.

Table 2.5: Accuracy for black-box model trained in [118] and tested with traffic captured in our campus network.

Class	Precision	Recall	F1
Benign	0.653	0.806	0.722
DoS	0.000	0.000	0.000
Port Scan	0.120	0.143	0.130
Average	0.256	0.315	0.282

We notice that the model classified most of the traffic as *benign*, a few samples as port-scan attacks, and none as DoS attacks. While we did not expect the model to generalize to real-world settings, we were intrigued that it correctly classified some port scans. Inspecting the decision presented in Figure 2.6, we can see that the ancestor of the Port Scan node splits most port scan attacks by checking $pkt\_1\_ipv4\_opt\_9 \leq -0.5$. Since the nPrintML model builds its feature vector using the first five packets of a flow (896 features for each packet and 4,480 in total), when a flow has fewer than five packets, it fills the remaining features with -1 values. Hence, to identify port scan attacks, the nPrintML model simply recognizes the absence of the second packet of the flow. To confirm this hypothesis, we carefully investigated the dataset published by the authors of nPrint [118] and noticed that most of the port scans in their dataset have only 1 SYN packet from the attacker to the target (differently from the original PCAPs in [214]). Thus the simple rule that the second packet of a flow is missing would be enough to find all port scans. However, in the case of our campus network traffic, most port-scan attacks also contain a second packet, which prevented the black-box model from classifying this type of traffic. This second packet is a TCP RST packet that attackers send to prevent the target from triggering the TCP SYN Cookie protection used to deal with TCP SYN

flooding attacks.

2.7.5 Additional Use Cases

As summarized in Table 2.2, we applied Trustee to analyze four additional use cases. Here, we briefly describe these additional use cases and report on the main findings that result from analyzing them with the help of Trustee. A detailed account of each of these additional use cases can be found in our technical report [124]. The first additional use case concerns Kitsune, an unsupervised ML classifier in the form of a complex ensemble of neural networks that was proposed in [166] to perform traffic anomaly detection (*e.g.,* Mirai attack). Using Trustee, we show that Kitsune is vulnerable to o.o.d. samples, thus corroborating previously-reported criticism of this model [18] and substantiating it with additional concrete evidence. For our second additional use case, we applied Trustee to scrutinize the OS fingerprinting application of nPrintML described in [118]. While Trustee's DT explanation confirms most of the results and claims obtained by nPrintML (*e.g.,* the model relies on TTL and window size header values to distinguish between Windows, Linux and MacOS OSes), it also provides evidence that the trained model is vulnerable to o.o.d. samples (*e.g.,* due to the limited set of operating systems in the dataset used). As third additional use case, we used Trustee to analyze the Random Forest Classifier that was proposed in [253] and trained with the UNSW-IoT dataset [222] to distinguish between different classes of IoT devices (*i.e.,* video, audio, etc.). In this case, Trustee's DT explanation identified clear instances of shortcut learning, mainly as a result of extracted features such as source and destination TCP and UDP ports that were used to train the model. The fourth and last additional use case deals with a network performance-related problem and concerns an application of Trustee to analyse Pensieve, a RL-based learning model for adaptive bit-rate video streaming [158]. Trustee's

DT explanation not only corroborates most of the previously-reported Pensieve-specific results [69,162], but it also achieves higher fidelity compared to reward-based optimization approaches such as Metis [162] (*i.e.,* F1 score = 0.90) and suggests that Pensieve may not be generalizable because the proposed model is vulnerable to o.o.d. samples.

2.8 Ablation Study

In this section, we evaluate key design choices we made for Trustee and that we motivated in Section 2.4.2.

Data augmentation and optimizing for fidelity. We first assess the impact of data augmentation (Line 11 in Algorithm 1) on the size and fidelity of the DT explanations generated by Trustee and at the same time consider the impact of using accuracy (*i.e.,* how well the DT classifies the data) rather than fidelity (*i.e.,* how well the DT mimics black-box classifications) as the optimization goal for Trustee. To this end, for the first three use cases described in Section 2.7, we use Trustee to extract DT explanations for four different settings (*i.e.,* with and without data augmentation, using either accuracy or fidelity), with all four settings using the same set of hyperparameter values. The results are shown in Figure 2.7 where the top plot depicts the (normalized) DT size and the bottom plot shows fidelity. We observe that in cases of small-sized extracted DTs (*e.g.,* maximum tree size for VPN vs. NonVPN and nPrintML IDS is 7 nodes and 47 nodes, respectively), data augmentation is not necessary. However, for extracted DTs that are more complex (*e.g.,* maximum tree size for Heartbleed is 1,491 nodes), the data augmentation step results in a significant reduction in DT size (roughly 20%, and especially for imbalanced datasets) and also improves the DT's fidelity (although only slightly, about 2-3%). In terms of optimizing for fidelity vs. accuracy, we observe no significant differences, mainly because all the analyzed use cases have excellent accuracy

to start with. However, we expect that for models that have lower accuracy, optimizing for fidelity may help end users identify reasons for why the model accuracy is low.

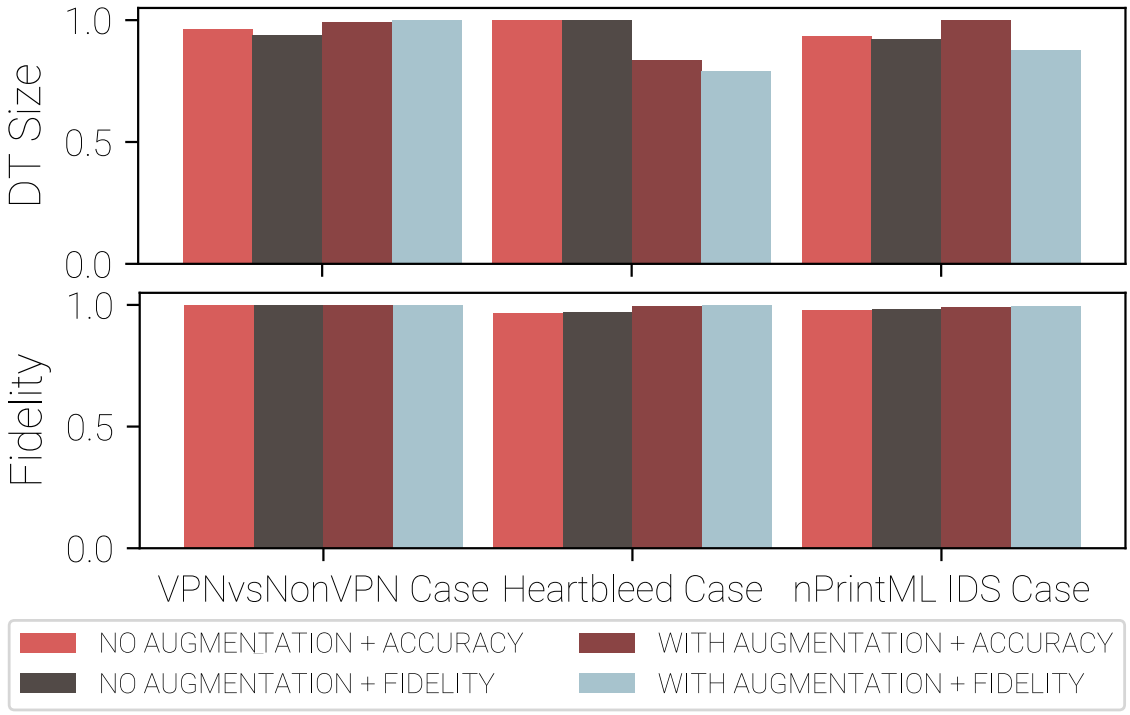


Figure 2.7: Ablation study results for data augmentation and optimization for fidelity/accuracy.

Pruning methods. We next evaluate and compare the trade-offs between fidelity and complexity of the DT explanations that Trustee generates when using a tree pruning method other than our proposed Top-k Pruning method (Line 14 in Algorithm 1). In particular, we consider the three pruning methods mentioned in Section 2.5.1: Max Leaves (pre-pruning), Max Depth (pre-pruning) and CCP (post-pruning). Figure 2.8 (top) depicts the results for the Heartbleed use case and shows the number of branches (x-axis) and fidelity (y-axis) that are achievable by each of these three methods as well as by our Top-k Pruning method. We observe that except for the Max Depth method, all other methods show similar performance, with the two post-pruning methods (*i.e.,* CCP and Top-k Pruning) outperforming the competitive pre-pruning method Max Leaves,

especially for high-fidelity DTs (*e.g.,* fidelity of 0.9 and above). In view of such minimal differences in their overall performance, choosing between CCP and Top-k Pruning boils down to practical considerations. In particular, while the CCP method relies on an implicit parameter α to determine how much post-pruning is necessary, our Top-k Pruning method gives end users direct control by means of the parameter k that explicitly reflects the amount of effort an end user is willing or capable to spend inspecting and analyzing Trustee's output.

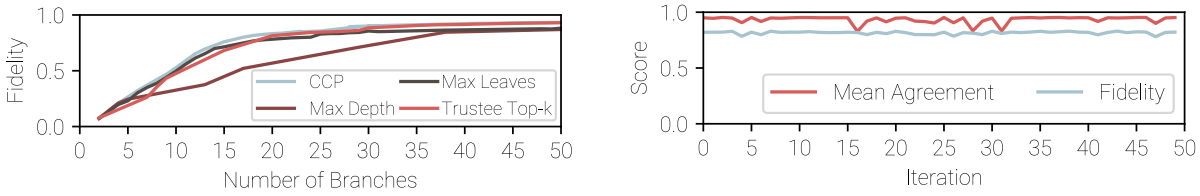


Figure 2.8: Ablation study results: pruning methods for Heartbleed use case (top), and model stability for Heartbleed use case with Top-10 Pruning (bottom)

Model stability. The last design choice we evaluate concerns the inclusion of an outer loop as part of Algorithm 1 (Lines 4-16). Given that Trustee only analyzes a subset of the input data to generate its output in the form of a DT explanation, it is fully expected that running Trustee under identical conditions (*i.e.,* same set of hyperparameter values) multiple times will result in different DT explanations. However, for an end user to trust the output of Trustee, it should be the case that the different DT explanations are stable in the sense that they make in general identical decisions when presented with the same input samples. We quantify this stability aspect of the output of Trustee by using the notion of agreement between DTs that measures how often the DTs will make the same decision for the same input data. To examine this aspect of Trustee, we consider the Heartbleed use case and ran Trustee (with S=1, thus effectively disabling the outer loop; number of samples $M = 593,123$ (*i.e.,* 30% of D_0); $N = 50$; and $k = 10$) a total of 50 different times. The results are presented in Figure 2.8 (bottom) and show an overall

high mean agreement and fidelity for each of the resulting 50 different DT explanations. However, in a few cases (*e.g.,* iterations 16, 28, 31), the mean agreement of the obtained DT explanations is as low as about 80%. This observation motivated us to include the outer loop in Algorithm 1 that ensures that Trustee outputs a DT explanation that has been selected so as to avoid obviously "bad" (*i.e.,* low mean agreement) and possibly misleading DT explanations for the given black-box model.

2.9 Conclusions and Discussions

In this paper, we present Trustee, a new framework that enables end users of ML-based solutions to gauge their trust in the black-box models that underlie these solutions. To demonstrate how Trustee works in practice, we consider several use cases of published ML-based solutions from the existing literature, examine whether end users can trust them, and discuss our findings and lessons learned.

First, we emphasize that our Trustee-based analyses of the considered use cases rely critically on the work of researchers who have made their ML-related artifacts publicly available. In recent years, the scientific community and the network research community, in particular, have argued strongly for more reproducibility [20, 179], and we second this effort. However, for the time being, network security researchers interested in using ML have to accept the lack of open-source datasets and a general reluctance for widespread data sharing due to privacy concerns as *faits accomplis*.

Second, given that the vast majority of published ML models that have been developed for a range of different network security problems are not fully reproducible, our reported findings based on a handful of use cases that are fully reproducible are in no way representative of the existing literature on applications of ML in the field of network security. However, the problematic nature of our findings for the few analyzed use cases

should serve as a cautionary tale as far as the popular use of standard ML pipelines in the field is concerned. In this sense, our work contributes to existing efforts that argue for looking at developments in this area with a critical eye (*e.g.,* see [15, 18, 225] and references therein) and identifies specific pitfalls that prevent end users from trusting proposed ML-based solutions and deploying them in production networks.

Last but not least, we purposefully designed Trustee to aid end users' efforts to check whether a given black-box model suffers from the problem of underspecification and can therefore not be trusted. While underspecification is a known and common problem in modern ML pipelines [64], this paper takes a first step towards detecting the presence and identifying the type of underspecification in a given black-box model. However, in the context of Trustee, these detection and identification tasks are currently not automated and depend critically on the help of domain experts who can use Trustee as-is to assert if a given black-box model makes decisions in accordance with existing domain knowledge or is even capable of teaching the domain experts new decision-making strategies. To realize the goal of automating these tasks, much work remains. In particular, we need to involve network operators and security experts in carefully designed user studies for quantitatively assessing their level of trust in a given black-box ML model that drives a proposed ML-based solution for a specific network security problem.

Chapter 3

Demystifying Network Foundation Models

3.1 Introduction

Machine learning solutions are widely applied in the networking domain, with numerous applications ranging from traffic classification [6, 144, 263, 264] and anomaly detection [129, 229] to quality of service optimization [7, 158, 258] and network security [3, 166]. However, many of them fail to generalize to production environments [18, 126, 148, 267] due to inherent biases in data collection methodology, limited coverage of operational scenarios, distribution shifts, and other factors, creating fundamental generalizability challenges.

Network foundation models as a potential solution. As a potential answer to these challenges, network foundation models (NFMs) have been gaining traction in the networking community [109, 144, 194, 241, 242, 264]. Similar to foundation models in other application domains, these models incorporate additional self-supervised pretraining phases that utilize unlabeled data to learn critical spatial, temporal, and causal relationships. NFMs represent a paradigm shift in network analysis, moving from the development of

task-specific learning models to the use of general-purpose pre-trained representations. Significant aspects of this shift include (i) generalizability: NFMs learn representations that transfer across multiple network analysis tasks; (ii) scale: NFMs process raw packet data and require no domain expert-based feature engineering; and (iii) impact: NFMs are expected to be increasingly deployed in production network systems for security, optimization, and monitoring. The adoption of NFMs promises to alleviate the difficulties caused by the growing complexity of modern network infrastructures and is viewed as an important step towards realizing the vision of self-driving networks.

Diverse architectures complicate evaluation. While aiming toward the similar goal of utilizing unlabeled data, design architectures and pretraining tasks vary significantly between existing NFMs, including masked token prediction purely on raw network payload [144] or packet header bytes [161, 194, 241]; flow statistics calculation [109]; patched image reconstruction [242, 264]; and modifying foundation models developed for other domains and applying them to the networking domain [128, 240]. Resulting largely from the variability in network data preprocessing, such diversity obscures the influence of the pretraining phase on model performance, making it difficult to understand what knowledge the model gains during pretraining. Historically, the community has relied on the performance of such models on fine-tuning tasks [193], which utilize small (compared to the pretraining phase) labeled datasets as quality indicators and comparison metrics between models. However, this approach poses challenges for understanding foundation models' design choices, pretraining tasks, chosen pretraining datasets, and overall knowledge gained during the critical pretraining phase.

Beyond downstream tasks: intrinsic evaluation. In this paper, we address these limitations and complement the existing downstream task-oriented research by exploring and assessing the representational quality of NFMs' embeddings without depending on fine-tuning problems. Specifically, we develop three complementary analysis techniques

to answer specific questions: (1) Embedding Geometry Analysis (§3.3.1) quantifies *how effectively models distribute representations across the embedding space* through anisotropy analysis, measuring the entanglement of learned representations and their influence on model performance. (2) Metric Alignment Assessment (§3.3.2) evaluates *feature correlation to identify whether models capture domain-expert metrics* like flow duration or TCP window dynamics. (3) Causal Sensitivity Testing (§3.3.3) employs perturbation analysis to evaluate *how embeddings respond to controlled protocol and context modifications, revealing higher-order context understanding*, including traffic shaping policies and congestion control mechanisms.

Key findings. Our empirical results demonstrate (§3.4.3) that embeddings from publicly available pretrained models exhibit significant anisotropy (mean cosine similarity $= 0.86 \pm 0.09$) and that addressing this issue leads to model performance improvement (up to 0.35 increase in F_1 scores). We notice (§3.4.4) significant alignment of models' embeddings with packet lengths, time-based features, packet flags, and even payload information (§3.4.5) despite its frequent encryption or absence in production environments, and show a direct correlation between anisotropy and high-level context in the models.

Our work shifts the focus from downstream task performance to intrinsic representational properties and sets the stage for further explorations into developing next-generation NFMs that can be expected to facilitate the creation of performant, generalizable, and robust ML-based solutions for disparate learning problems – a critical step toward realizing the ambitious goal of fully self-driving computer networks. Our fully reproducible code is available at `https://github.com/maybe-hello-world/demystifying-networks`.

3.2 Preliminaries

In this section, we provide information about existing evaluation efforts and preliminary information on what hidden context refers to in the networking domain.

3.2.1 Existing evaluation efforts

To our best knowledge, several papers attempted to taxonomize the existing works in the area of foundation models for networking. [36] contains a survey of existing works in GenAI for networking, including both text-based and traffic-based approaches, and raises various questions about interpretability and efficiency, but does not provide any comparative analysis of the models and their performance. [247] provides a comprehensive analysis of both feature-engineered and foundation model-based approaches for network traffic classification, including design choices, feature selection, traffic granularity, and benchmarks and downstream tasks used. The work also implements several data occlusion strategies to evaluate the influence of different features on model performance and includes model development guidelines, though these results are derived from a single dataset for two models only. [193] provides a downstream task-oriented framework for evaluating classic and pretrained models for network traffic classification, relying on 7 public datasets covering 20 existing tasks and 7 different models for evaluation (with only two foundation models). Despite including an extensive set of downstream results, this work does not investigate model design choices, input data, or embedding structure and quality.

3.2.2 Hidden context in networking

One key learning challenge in the networking domain is that network traffic is influenced by various *hidden context* C, which is not explicitly expressed in the network traffic traces but significantly influences it. This context consists of: various **network conditions**,

which include partially observable characteristics of the network (throughput, latency), traffic shaping policies, network congestion situation, and influence of other network traffic passing through the same interfaces; **application behavior**, which includes participants and their communication rules, such as communication protocols and congestion control algorithms, adaptive bitrate algorithms, application specific communication pattern (prevalence of download traffic over upload), inherent burstiness of the traffic; and others.

Both aspects of networking hidden context can often depend on one another, introducing *cross-dependencies* that have to be dealt with during analysis. Examples include how network conditions influence application behavior (e.g., ABR algorithm choices of bitrate [7, 8, 34, 158, 254]), and vice versa (e.g., TCP congestion control algorithms fairness [4, 154, 255]), or how even more complicated multi-way dependencies may arise (e.g., usage of performance-enhancing proxies [147] in geostationary satellites which terminate connections and introduce their own behavior on multiple levels).

3.2.3 Necessity of intrinsic evaluation

Since traditional evaluation efforts rely exclusively on downstream task performance, they suffer from critical shortcomings such as (i) performance scores provide no insight into what the models actually learn about network conditions or application behaviors, (ii) high performance for one task does not guarantee generalizability to new network domains, (iii) models can achieve good performance through dataset-specific shortcuts rather than genuine network understanding, and (iv) practitioners have no guidance for choosing models or identifying failure modes. Addressing these shortcomings calls for creating an intrinsic evaluation framework that can reveal what the models actually learn about network traffic, independent of specific tasks.

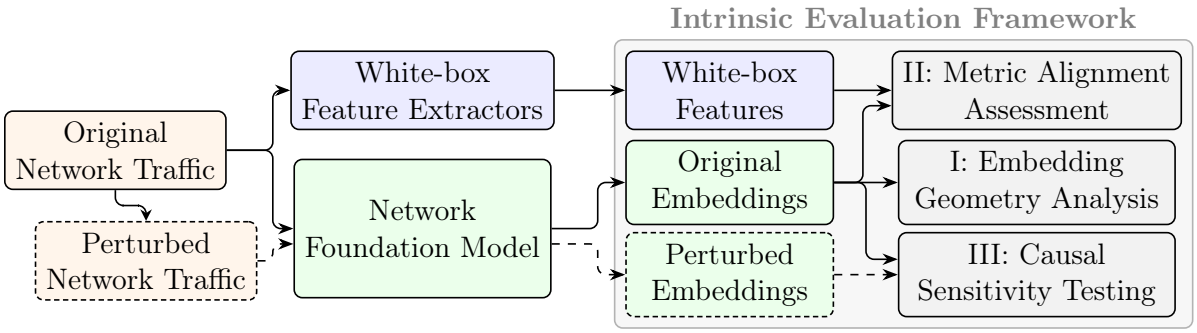


Figure 3.1: Visual overview of the proposed framework. Color indicates different stages of the analysis and dashed lines and boxes denote the perturbed network traffic path.

3.3 Intrinsic Evaluation Framework

NFMs pretrained on raw packet traces inherently encode an internal *latent space* reflecting various network and application characteristics captured from the utilized training data. To gain insights into these characteristics, we explore *embeddings* generated from diverse network data by *frozen* NFMs.

Using the calculated embeddings, we aim to: quantify how effectively models distribute representations across their available dimensions (§3.3.1), demonstrating if they capture both shared protocol patterns and distinctive flow characteristics; measure the correspondence between learned embeddings and established network metrics (§3.3.2), revealing whether NFMs inherently calculate the same statistical features that domain experts have engineered over decades; and evaluate how embeddings respond to controlled perturbations at both protocol and contextual levels (§3.3.3), verifying that models learn meaningful causal dependencies. For each technique, we provide a claim (*what NFMs should be able to do*), the rationale behind this claim, and our methodology to evaluate NFMs for the given claim. A visual overview of the Intrinsic Evaluation Framework is presented in Figure 3.1.

Applicability. In theory, our intrinsic evaluation framework can be used to evaluate any

model that learns latent space representations, but it is most valuable for models that involve task-agnostic pre-training. While traditional approaches (e.g., neural networks trained using supervised learning techniques) can be evaluated using our metric alignment component, they often lack the explicitly extracted latent representations that make geometric analysis and causal sensitivity testing meaningful.

Importantly, because of the data representation challenges posed by network traffic, how a model's input data is being preprocessed is intrinsically entangled with what model architecture is selected. Since both aspects reflect the explicit decisions that the original model's developers made, we are unable to evaluate them individually.

3.3.1 Embedding Geometry Analysis: quantifying representation space utilization

Claim. NFMs that effectively capture both shared protocol semantics and distinctive per-flow packet dynamics produce embeddings that efficiently utilize the full representation space. These models generate flow representations that distribute anisotropically throughout the latent space rather than collapsing into a concentrated cluster.

Rationale. Effective network modeling requires capturing both global protocol patterns and flow-specific variations. A well-distributed embedding geometry with measured anisotropy indicates [92] that the model distinguishes between flows along multiple meaningful dimensions. Such embeddings encode critical network characteristics including spatial (handshake vs. data exchange vs. teardown) or temporal patterns (short vs. long inter-arrival times) and causal reactions (packet loss vs. delays). Conversely, embeddings that cluster tightly with high cosine similarity suggest the model compresses diverse flows into nearly identical representations (similar to [142, 227]), overlooking subtle per-flow variations essential for robust network reasoning and downstream task performance.

Methodology. To quantify embedding geometry, we employ the established concept of anisotropy from contextualized language models [45, 71, 91], expressed by cosine similarity between different traffic flows. For each network traffic flow x_j, we extract its hidden representation h_j from the final encoder layer of an NFM (as defined by the authors of the model). Across a set of n such embeddings $\{h_j\}_{j=1}^{n}$, we estimate the anisotropy score $\mathcal{A} = \mathbb{E}_{i \neq j}[\cos(h_i, h_j)] \approx \frac{1}{|S|} \sum_{(i,j) \in S} \cos(h_i, h_j)$ (where $cos(u, v)$ is the *cos* similarity), by sampling random flow pairs S similar to [85].

Further, we analyze the contributions to anisotropy of the largest dimensions by using Mean Cosine Contribution (MCC) from [122] (defined for a dimension k as $MCC(k) = \frac{1}{|S|} \sum_{(i,j) \in S} CC_k(h_i, h_j)$, where $CC_k(u, v) = \frac{u_k v_k}{||u||\ ||v||}$) to ensure no single axis dominates. Lower values of $\mathcal{A}$ combined with a uniform MCC distribution indicate that embeddings uniformly utilize the available dimensions, possibly facilitating distinguishing between variations in network behavior.

3.3.2 Metric Alignment Assessment: measuring correspondence with domain-expert features

Claim. NFMs implicitly compute well-known network performance and behavioral metrics within their latent space. Flow-level embeddings encode critical network statistics such as flow duration, packet size distributions, and TCP dynamics without explicit supervision.

Rationale. Over decades, network domain experts have engineered various statistical white-box features and successfully used them for developing traffic-based machine learning solutuons [3, 140, 211]. By measuring the structural alignment between hidden representations of NFMs and these network metrics, we can verify whether models have learned traditionally meaningful generalizable network semantics rather than superficial

correlations with particular downstream labels.

Methodology. We select a set of established network metrics calculated by CICFlowMeter [140], which have been extensively validated for traffic classification in prior works [3]. Let $\{x_j\}_{j=1}^{n}$ be our set of flows and let $m_i(x_j)$ denote the value of the ith CICFlowMeter metric for flow x_j. We extract the embedding $h_j \in \mathbb{R}^d$ from the NFM's final encoder layer before any task-specific heads (ET-BERT, netFound) or decoder layers (YaTC, NetMamba) and assemble $H = [h_1, \ldots, h_n]$.

For each metric m_i, we compute the similarity index $\rho(m_i, h_j)$ between the metric and the model's embedding representation across all traffic flows in the dataset using Centered Kernel Alignment (CKA) [133]. Unlike cosine similarity, CKA allows for calculating the similarity index between representations of different dimensionalities and is invariant to orthogonal transformations and isotropic scaling, allowing for effective capture of both linear and non-linear relationships between representations. CKA values near 1 demonstrate that semantic probes successfully extract network-level metrics from embeddings, validating the model's implicit computation of these key features.

3.3.3 Causal Sensitivity Testing: interventional analysis of protocol and context dependencies

Claim. NFMs that effectively encode both protocol semantics and network context exhibit two key properties: (1) protocol-relevant perturbations produce quantifiable, focused changes in embedding similarity scores, and (2) high-level network contexts (e.g., congestion control algorithms, queue management policies) create distinct, linearly separable embedding subspaces. These properties are proof that the model learns meaningful causal dependencies between inputs and network conditions.

Rationale. A comprehensive test of causal understanding requires dual validation:

we aim to verify both bottom-up causality (how low-level protocol features influence embeddings) and top-down contextual awareness (how high-level network conditions shape representation space). We explicitly separate these efforts into two complementary methodologies – feature perturbation analysis and context discrimination – to provide quantitative evidence of whether an NFM has learned a coherent causal model of network behavior spanning multiple levels of abstraction.

Methodology: sensitivity to protocol-relevant perturbations. For a selected dataset comprising flows $\{x_1, x_2, ..., x_n\}$, we compute the baseline similarity score $S(\{h_j\}_{j=1}^n)$, where h_j denotes the embedding of the j^{th} flow and $S(\cdot)$ quantifies the average pairwise cosine similarity between embeddings. We define a perturbation strategy $\delta(f)$ operating on a subset of features $f \subseteq F$, where $F = \{f_1, f_2, ..., f_n\}$ constitutes the complete set of traffic header features. For our analysis, we use a token replacement strategy that implements any token replacement from a valid range of tokens from the model's dictionary [204]. This strategy applies uniformly across all flows in the dataset.

Given a feature perturbation, we derive the perturbed flows $\{x_j^{\delta(f)}\}_{j=1}^n$ and extract their corresponding embeddings $\{h_j^{\delta(f)}\}_{j=1}^n$, and then measure the similarity score between perturbed embeddings $h_j^{\delta(f)}$ and the original h_j using cosine similarity to measure changes in the hidden representations.

A significant decrease in the similarity score after perturbation indicates that the model's representation exhibits sensitivity to the perturbed features. Large changes for protocol-relevant features demonstrate that these features causally influence the model's internal representations. Such findings reveal that the model implicitly learns to encode protocol-relevant features, aligning with domain knowledge about network protocol behavior.

Methodology: extraction of exogenous network context. First, we use a network emulator to generate network traffic with distinct high-level contexts ($c_j \in$

$\{1, \ldots, C\}$). Each context has multiple possible values, such as application type (video streaming vs. conferencing), congestion control algorithm (CUBIC vs. BBR), queue management policies (pFIFO vs. CoDEL), and cross traffic characteristics at the bottleneck link (more bursty and less intense or vice versa). For any given NFM, we extract embeddings (h_j) from the last encoder layer that represent the encoded state of j^{th} flow. We compute these embeddings across different combinations of high-level contexts.

To quantify the NFM's context sensitivity, we employ two complementary analyses. First, we calculate pairwise cosine similarities between embeddings generated under different context combinations. We also establish a reference point by computing the average similarity score from embeddings of publicly available unlabeled datasets (the "average"). We then measure the relative change in similarity when moving from a base context (with default parameter values) to each alternative context combination. By comparing these changes against the deviation of the base context from the baseline average, we can contextualize the magnitude of embedding shifts. A significantly higher relative change in similarity compared to the baseline deviation suggests that the model effectively captures the distinctive characteristics of different network contexts.

As a second quantitative measure, we train a simple logistic regression classifier built on top of the frozen embeddings to distinguish between different contexts, and calculate its F_1 score. This approach assesses whether the information necessary to separate contexts is linearly accessible from the embedding space. A high F_1 score confirms that the NFM's embeddings internalize unobserved network conditions and can effectively discriminate between different operational scenarios, validating that the model learns meaningful representations of network behavior under varying conditions.

3.4 Benchmarking

In this section, we evaluate state-of-the-art NFMs using our proposed evaluation framework.

3.4.1 Network Foundation Models

Our benchmarking applies the intrinsic evaluation framework to diverse architectural approaches in NFMs. We selected four state-of-the-art representative NFMs that span different design philosophies, input representations, and pretraining strategies to evaluate how these fundamental choices impact representational quality.

For each model, we used publicly available pretrained checkpoints provided by the authors without modifications, maintaining each model's original data processing pipeline to ensure fair comparison. Our selection includes **YaTC** [264], **ET-BERT** [144], **netFound** [109], and **NetMamba** [242].

These models represent key design dimensions in NFM architecture: input modality (headers vs. payload vs. both), sequence length (5 vs. 60 packets), architectural foundation (Transformer vs. Mamba), and pretraining objectives (single vs. multi-task). Through our framework, we assess how these design choices influence embedding geometry, metric alignment, and causal sensitivity.

3.4.2 Datasets

Our benchmarking requires diverse network traffic datasets to ensure comprehensive evaluation across varying network conditions, traffic types, and operational settings. We selected five datasets spanning both controlled environments (**Android Crossmarket** [197], **CIC-IDS2017** [214], **CIC-APT-IIoT24** [96]) and real-world deployments (**CAIDA** [11], **MAWI** [57]). For preprocessing, we standardized all datasets by extracting

uniform flow records using the data processing scripts provided by the authors of the considered models.

3.4.3 Embedding Geometry Analysis

Table 3.1: Mean cosine similarity (cos) between embeddings and Mean Cosine Contribution (MCC) of the top dimension of the embeddings towards the average cosine similarity.

	YaTC		**ET-BERT**		**netFound**		**NetMamba**	
Dataset	*cos*	*MCC*	*cos*	*MCC*	*cos*	*MCC*	*cos*	*MCC*
Crossmarket	0.85	0.25	0.88	0.02	0.69	0.01	0.93	0.02
CIC-APT-IIoT24	0.87	0.27	0.88	0.02	0.82	0.01	0.98	0.02
CIC-IDS2017	0.85	0.22	0.74	0.01	0.69	0.01	0.92	0.02
CAIDA	0.87	0.21	0.71	0.01	0.86	0.01	0.99	0.03
MAWI	0.88	0.19	0.78	0.01	0.94	0.02	0.99	0.02

Table 3.1 presents results from applying our embedding geometry analysis.

Models are not consistent in the anisotropy scores. Our analysis reveals significant variability in how different NFMs utilize their representation space. netFound demonstrates more variability in space utilization (with anisotropy score varying between 0.69 and 0.94 for different datasets) compared to autoencoder-based models YaTC and NetMamba (which have more consistent scores of 0.85-0.88 and 0.92-0.99, respectively). ET-BERT occupies an intermediate position (0.71-0.88).

MCC analysis reveals distinct failure modes. While anisotropy scores identify models with concentrated representations, our Mean Cosine Contribution analysis further distinguishes between qualitatively different representational problems. NetMamba's uniform MCC values (the highest being around 0.03) indicate semantically collapsed representations without any single dimension contributing significantly to the collapse, while YaTC's uneven distribution (MCC 0.19-0.27) reveals problematic dimensional dominance where a single dimension captures disproportionate variance.

Geometric properties predict environmental responses. Embedding geometry analysis systematically exposes how different architectures respond to dataset variations. The observation that most models produce higher cosine similarity (more collapsed representations) on real-world datsets — with ET-BERT uniquely showing the opposite pattern — provides predictive insight into how these models might generalize to deployment environments with different traffic distributions.

Table 3.2: ΔF_1 of NetMamba after fine-tuning a single linear layer for 30 epochs on decorrelated embeddings (over five training runs).

	Crossmarket	**CIC-IDS2017**	**CIC-APT-IIoT24**
NetMamba	$+0.35 \pm 0.02$	$+0.11 \pm 0.27$	$+0.03 \pm 0.02$

Anisotropy directly impacts performance. While anisotropy is a measure that quantifies embedding quality, it does not directly capture how this geometry affects downstream performance [71]. Therefore, we complement our anisotropy analysis by considering an additional metric: isotropification gain. This metric quantifies the potential performance improvement when correcting for suboptimal embedding distributions. We apply a deterministic decorrelation transformation [119] to the frozen raw representations and retrain only a linear classifier for the downstream task. We define the isotropification gain as $\Delta F_1 = F_1^{\text{decorrelated}} - F_1^{\text{raw}}$.

To validate that embedding geometry directly affects model utility, we identified NetMamba as an ideal candidate for isotropification based on its high anisotropy. Table 3.2 shows the resulting F_1 score improvements ($+0.03$ to $+0.35$) after applying decorrelation transformations. Note that this improvement is calculated for a simplified classification head (linear classifier), and results for more complicated models can vary. However, these results suggest that anisotropy is not merely a descriptive metric but can predict potential performance gains that derive from applying representation enhancement techniques to NFMs.

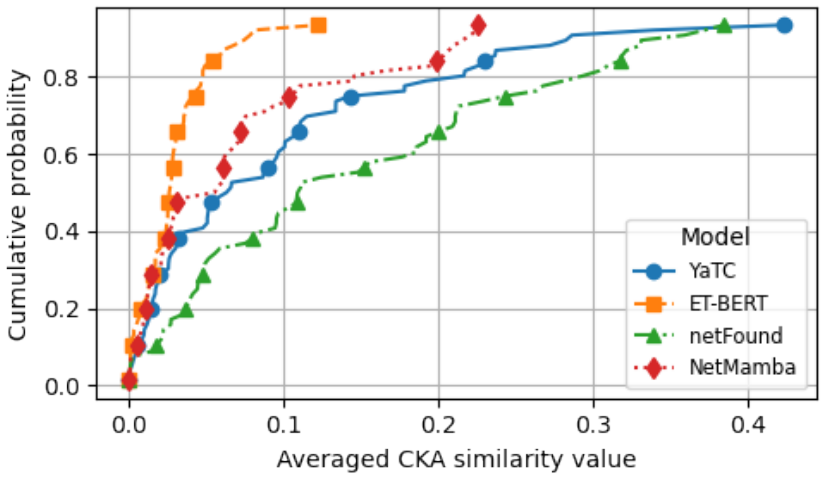


Figure 3.2: CDF of CKA similarity among different model embeddings and CICFlowMeter features averaged across all five datasets.

3.4.4 Metric Alignment Assessment

Table 3.3: Averaged CKA similarity index between CICFlowMeter white-box features and model embeddings.

	Crossmarket	CIC-APT-IIoT24	CIC-IDS2017	CAIDA	MAWI	Average
YaTC	0.098	0.148	0.092	0.014	**0.070**	0.093
ET-BERT	0.012	0.014	0.064	0.033	0.026	0.029
netFound	**0.156**	**0.219**	**0.167**	**0.052**	**0.070**	**0.143**
NetMamba	0.047	0.141	0.042	0.030	0.051	0.066
Average	0.078	0.131	0.091	0.032	0.055	0.077

Table 3.3 presents the averaged Centered Kernel Alignment (CKA) similarity index between model embeddings and CICFlowMeter features across all datasets, quantifying the degree to which NFMs implicitly encode established network metrics without explicit supervision.

Architectural design determines metric alignment. Our analysis reveals substantial variation in how different architectures encode domain-expert features. netFound consistently demonstrates the highest alignment (average CKA: 0.143), suggesting its multi-input approach and packet burst representation effectively capture statistical prop-

erties identified by domain experts. In contrast, ET-BERT's payload-only approach shows minimal alignment (average CKA: 0.029), indicating traditional network metrics cannot be reliably extracted from payload representations alone. Since YaTC and NetMamba share similar architectural designs, they demonstrate similar CKA performance across datasets.

Feature diversity enhances metric coverage. Figure 3.2 shows the CDF of the CKA similarity index, revealing that models with diverse input features exhibit a more uniform alignment distribution compared to models with constrained input features (i.e., more slowly increasing CDF, resulting in more features having a high CKA similarity value). Models processing packet headers, payload, and flow metadata (netFound) capture a broader spectrum of expert-designed metrics compared to specialized architectures (ET-BERT, YaTC, NetMamba), indicating that architectural diversity directly impacts the comprehensiveness of implicit feature computation.

Environmental context affects metric relevance. All models demonstrate significantly lower CKA similarity on real-world datasets (CAIDA: 0.032, MAWI: 0.055) compared to datasets from controlled environments (CIC-APT-IIoT24: 0.131, CIC-IDS2017: 0.091). This consistent pattern suggests that existing NFMs struggle to encode established network metrics in production environments.

Table 3.4: Average CKA similarity index between model embeddings and CICFlowMeter features. Top 5 features with the highest similarity index are presented for each model.

Model	Feature name	Mean CKA	CKA Std	Model	Feature name	Mean CKA	CKA Std
YaTC	Fwd Seg Size Min	0.423	0.185	netFound	FWD Init Win Bytes	0.385	0.352
	FWD Init Win Bytes	0.339	0.241		Packet Length Max	0.372	0.302
	Packet Length Min	0.286	0.085		Fwd Seg Size Min	0.359	0.338
	SYN Flag Count	0.282	0.230		Fwd IAT Total	0.331	0.173
	Bwd Packet Length Min	0.272	0.209		Flow Duration	0.331	0.170
ET-BERT	SYN Flag Count	0.122	0.080	NetMamba	Fwd Segment Size Avg	0.225	0.244
	FWD Init Win Bytes	0.081	0.068		Fwd Packet Length Mean	0.225	0.244
	Packet Length Std	0.078	0.094		Average Packet Size	0.221	0.258
	FIN Flag Count	0.074	0.074		Packet Length Min	0.215	0.273
	Packet Length Max	0.068	0.097		Packet Length Mean	0.210	0.253

Temporal-spatial features dominate alignment patterns. Table 3.4 shows that all models demonstrate the highest alignment with features related to packet length, segment size, and flow burstiness. For instance, netFound shows particularly strong alignment with features *forward initial window bytes* (CKA: 0.385) and *packet length max* (CKA: 0.372), while YaTC prioritizes the feature *forward segment size min* (CKA: 0.423). Even ET-BERT, despite overall low alignment, shows relative preference for the feature *SYN flag count* (CKA: 0.122). These observations suggest that all models discover the importance of learning initial connection behavior and data transfer volumes.

3.4.5 Causal Sensitivity Testing: protocol-relevant perturbations

Table 3.5: Cosine similarity between the baseline and embeddings after perturbing IP and TCP features.

	YaTC		**ET-BERT**		**netFound**		**NetMamba**	
Baseline *cos*	0.87		0.71		0.86		0.99	
	% tok	*cos*	% tok	*cos*	% tok	*cos*	% tok	*cos*
SEQ/ACK	5%	0.61	0%	-	22%	0.99	5%	0.98
IP Total Length	0.6%	0.88	0%	-	5%	0.99	0.6%	0.99
IP TTL	0.6%	0.88	0%	-	5%	0.98	0.6%	0.99
TCP Flags	0.9%	0.86	0%	-	5%	0.99	0.9%	0.99
TCP Window Size	2.5%	0.67	0%	-	5%	0.99	2.5%	0.99
Payload	75%	0.18	100%	0.48	33%	0.99	75%	0.62

Table 3.5 presents results of our protocol-relevant perturbations testing on the CAIDA dataset, measuring how modifications to specific protocol features affect embedding stability. We selected the CAIDA dataset because it contains diverse, unfiltered, real-world Internet backbone traffic and allows therefore for a more realistic evaluation of model robustness than using controlled laboratory datasets.

Perturbation methodology accounts for model differences. Since the considered NFMs employ substantially different tokenization techniques, we applied pertur-

bations to semantically meaningful packet features (e.g., TCP Window Size, IP TTL) rather than model-specific tokens. Consequently, the same packet feature modification affects different percentages of tokens across models as shown in the "% tok" columns of Table 3.5. This approach ensures fair comparison by focusing on model-independent protocol semantics rather than model-specific implementations.

Models generally maintain stability under header perturbations. For most model-feature pairs (netFound, NetMamba, some features of YaTC), the cosine similarity between embeddings before and after header perturbation exceeds the average internal similarity of the baseline dataset. These findings indicate that single feature modifications do not significantly influence model representations from an embedding geometry perspective, indirectly showing architectural stability for header processing.

Selective feature sensitivity reveals architectural priorities. YaTC demonstrates selective sensitivity to modifying transport layer features, particularly SEQ/ACK (0.61) and TCP Window Size (0.67), even though these modifications affect only small portions of the input (5% and 2.5% of tokens, respectively). This targeted sensitivity suggests architectural attention to specific protocol control signals, a pattern that is not evident in the other models, all of which maintain near-baseline similarity across all header modifications.

Architectural differences in protocol processing. ET-BERT shows no sensitivity to header modifications. This result is not surprising because the model's architecture deliberately ignores headers. netFound demonstrates remarkable stability (0.98-0.99) across all header modifications despite processing these fields, suggesting its representations prioritize higher-level patterns. These distinct responses directly reflect fundamental differences in how models process protocol information.

Payload dependency across NFMs. Unlike header features, payload perturbations introduce significant representation shifts across almost all models (YaTC: 0.18, ET-BERT:

0.48, NetMamba: 0.62). These results reveal a critical dependency on payload features for decision-making, even though this data is often encrypted or omitted in production environments. These findings highlight a potential model robustness concern that would go unnoticed in traditional downstream evaluations.

Perturbation magnitude does not predict representation impact. The percentage of tokens modified does not consistently correlate with representation changes. Small modifications to critical features (SEQ/ACK at 5% for YaTC) can produce larger representational shifts than more extensive modifications to other features (IP TTL at 0.6%), revealing an implicit hierarchy of feature importance within each model's internal representations.

3.4.6 Causal Sensitivity Testing: exogenous network context

We use the network emulator NetReplica [66] to generated five synthetic datasets, each consisting of ≈ 100 network flows and obtained using specific choices of congestion control algorithms, AQM policies, and cross-traffic patterns.

Table 3.6: Average *cos* similarity across all datasets, *cos* similarity change (w.r.t. difference between stability baseline and average value), and linear probing F_1 score for synthetic datasets.

	YaTC		**ET-BERT**		**netFound**		**NetMamba**	
Metric	*cos*	-	*cos*	-	*cos*	-	*cos*	-
Average	0.8633	-	0.7977	-	0.8017	-	0.9639	-
Stability baseline	0.8939	-	0.9698	-	0.9700	-	0.9940	-
	Δcos	F_1	Δcos	F_1	Δcos	F_1	Δcos	F_1
Congestion Control	-13.07%	0.48	0.32%	0.41	-1.84%	**0.60**	0.50%	0.29
AQM	0.99%	0.51	-1.44%	0.68	-10.97%	**0.93**	-280.01%	0.70
Crosstraffic	-5.71%	0.24	0.60%	0.43	-6.45%	**0.78**	0.51%	0.33
All	6.72%	0.47	-1.07%	0.46	-31.32%	**0.97**	-42.34%	0.62

Table 3.6 presents average *cos* similarity values for a diverse set of public datasets used

in this work ("Average" row) and for the stability baseline dataset (i.e., a fixed dataset with given parameters for all choices, selected as a baseline). Assuming that the average and stability baselines represent lower and higher bounds respectively for the possible similarity values for each model, we list for each different high-level context change (see rows of the table labeled "Congestion Control", "AQM", "Crosstraffic", and "All" for all changes) the Δcos value which is calculated as the difference between the average *cos* similarity of the test dataset and the stability baseline. The reported F_1 scores denote the performance of a linear probe trained to separate between the stability baseline and the test dataset.

All models are able to group the baseline traffic. Table 3.6 ('Average' and 'Stability baseline' lines) shows that all models demonstrate much higher average *cos* similarity (but staying below the upper bound provided by the stability baseline) compared to the average similarity in diverse datasets. This property suggest that the models are able to reliably group the similar traffic and is important, for example, for anomaly detection tasks, where previously unseen anomalous traffic should be separated from the normal traffic.

High-level context insignificantly influences the *cos* similarity of embeddings. Given that average *cos* similarity of diverse datasets and stability baseline are different for different models, we normalized Δcos in Table 3.6 by the difference between the average *cos* similarity of the stability baseline and average *cos* similarity across diverse datasets, effectively measuring changes between the "lowest" and the "highest" measured *cos* values. We notice that for all models, the Δcos of the synthetic datasets is relatively small, implying that the resulting *cos* is very close to the stability baseline. Except for NetMamba-AQM, NetMamba-All, and netFound-All, for all other model-dataset pairs, the absolute value of Δcos is lower than 15%, an indication that the models do not explicitly disentangle the high-level context from the baseline. In some cases, the Δcos is

even positive (i.e., embeddings are grouped even closer), demonstrating that the model considers the traffic with the introduced changes to be more similar to the stability baseline than the stability baseline itself and does therefore not notice the high-level context change.

Δcos correlates with linear probing F_1 score. For most of the model-dataset pairs, the lower average *cos* score correlates with a higher linear probing F_1 score, demonstrating (similar to Table 3.2) that disentangling produced embeddings might lead to better fine-tuning performance even for more complicated fine-tuning tasks.

Models are better at distinguishing different AQMs than different Congestion Control algorithms, crosstraffic patterns, or even all changes together. For almost all models, the linear probing F_1 score is significantly higher for the AQM row in Table 3.6 compared to the Congestion Control and Crosstraffic rows. This observation suggests that the influence of AQM policies is easier captured from the raw network traffic, and sometimes even easier than when all changes are introduced together. Across all four rows, netFound has the highest F_1 scores, an indication that it is capable of capturing the high-level context changes.

3.5 Conclusion

In this paper, we introduce an initial approach to a principled evaluation of NFMs through intrinsic representation analysis and conduct a series of experiments on four SOTA NFMs over five diverse datasets for embedding geometry analysis, metric alignment assessment, and causal sensitivity testing. Our work reveals a number of critical insights, such as (1) all analyzed NFMs exhibit significant anisotropy that directly impacts downstream performance; (2) each model's architectural design fundamentally determines which network metrics can be reliably extracted from its representations; and (3) all

models demonstrate concerning sensitivity to payload information despite its frequent encryption (or omission) in production environments. In particular, our proposed framework provides the network community with systematic tools for comparing different NFM architectures objectively, identifying model limitations before deployment, guiding architectural improvements based on intrinsic properties, and understanding failure modes that downstream metrics miss. At the same time, our findings highlight fundamental limitations in current NFM architectures and motivate future research on relationships between architectural choices, representation properties, and models' performance.

Limitations. The current work poses a number of limitations that affect the impact of our study. Our current scope of exploration and metrics used are far from exhaustive and are an initial attempt at evaluating NFMs' performance and latent knowledge without focus on downstream performance. Our application of the cosine similarity metric can produce incorrect insights [226] if applied to other models (e.g., which used cosine-based loss functions during pretraining). Finally, our dataset selection is based on publicly available data that can be flawed [148] or biased in terms of the network traffic patterns contained in the data.

Part II

Infrastructure for Networking Research

This part presents the systems contributions of the dissertation and argues that credibility in machine learning for networking depends not only on better analysis, but also on better data. The chapters in this part address one of the central causes of brittle and non-generalizable models identified earlier in the dissertation: the reliance on narrow, synthetic, or otherwise unrepresentative datasets. Together, they build the infrastructure needed to collect realistic, diverse, and reusable network measurements, turning data collection from a one-off effort into a practical and programmable capability for networking research.

Chapter 4 introduces PINOT, a campus-scale programmable infrastructure deployed on the UCSB production network that combines passive traffic monitoring with active measurements from distributed devices to support the collection of realistic labeled data. Chapter 5 then generalizes this effort through netUnicorn, a thin-waist, intent-based platform that separates experiment specification from infrastructure-specific mechanisms and enables iterative, portable data collection across heterogeneous environments. In the overall dissertation, these systems form the bridge between diagnosing why current network ML models fail and enabling the development of more generalizable models by making higher-quality data collection feasible at scale.

Chapter 4

PINOT: Programmable Infrastructure for Networking

As modern network communication moves closer to being fully encrypted and hence less exposed to passive monitoring, traditional network measurements that rely on unencrypted fields in captured traffic provide less and less visibility into today's network traffic. At the same time, approaches that use techniques from machine learning (ML) to extract subtle temporal and spatial patterns from encrypted packet-level traces have shown great promise in offsetting the lack of visibility due to encryption [2, 9, 10, 40, 42, 54, 132, 143, 146, 156, 158, 165, 183, 218, 232].

Despite their promise, ML-based approaches often have a credibility problem that arises from the quality of underlying training data. Given the challenges of curating high-quality training data at scale, researchers typically end up collecting their own (or reusing existing third-party or synthetic) data, often from small-scale testbeds. Such data is generally of low quality as it is not representative of the target environment, collected over too short of a time period, or measured at too coarse of a granularity. The learning models trained using such data tend to be vulnerable to different failure modes that make

them not credible [64]. This observation begs a fundamental question, **how can we develop credible ML artifacts for managing encrypted network traffic?**

This paper describes our ongoing efforts to enable researchers and practitioners to develop more credible ML artifacts by lowering the effort that is required for collecting more high-quality data for a wide range of learning problems from realistic and representative network environments.

4.1 PINOT

To answer the stated question, we argue for developing a representative open infrastructure where researchers can freely explore different data collection strategies (e.g., collecting the same type of data as some existing data from new networks) or flexibly perform new data collection experiments (i.e., reproducing existing experiments) to develop more credible ML-based solutions. Previous efforts [107] consider university campus networks as representative real-world infrastructures, mainly because of their scale, user base, and realistic nature of observed traffic.

At UCSB, we designed and implemented PINOT[1], a programmable data-collection infrastructure for collecting labeled data from the underlying production network. Being able to collect and curate such data sets aids the development of more credible ML models for different networking-related learning problems. For example, we have used this platform to curate labeled data sets to train learning models for inferring QoE metrics for various video streaming (e.g., YouTube, Twitch, etc.) and video conferencing applications (e.g., Google Meet); detecting traffic of interest for cybersecurity appliances (e.g., IDS); and performing device or application fingerprinting. We also used a combination of active and passive measurements to first find campus locations where devices experience large

[1]Supported by the NSF's Campus Cyberinfrastructure (CC*) grant, OAC-2126327

downstream RTT values and then identify a device, access point, or downlink as the main culprit for the observed large RTT. Being highly configurable, the platform allows us, for example, to collect data over extended periods of time, enabling the study of weekly, monthly, and seasonal trends or changes in traffic patterns.

4.2 Passive Data Collection

To enable passive real-world traffic monitoring, PINOT supports data collection from the border gateway of the UCSB campus network. This data collection infrastructure relies primarily on PISA-based switches to ensure the passive collection of packet-level traffic traces in a privacy-preserving manner at scale. More concretely, for each incoming packet, the switch generates a cloned copy of the packet with randomized privacy-sensitive fields (e.g., IP addresses of end users on campus) and pruned payload fields, and then load-balances the output packet stream to a cluster of collection servers. Each of these servers captures and stores anonymized packet headers for further analysis. We augment these packet traces by joining them with logs from various production appliances (e.g., Aruba's AirMon, and Palo Alto's MTA) to attach labels for learning problems.

4.3 Active Data Collection

To support active measurements, we deployed a fleet of network measurement devices across the campus and connected them to the public network infrastructure.

We used ARM64-based single-board computers Raspberry Pi 4, running a modified version of the Ubuntu Server that includes performance and security optimizations. Each device is powered either via Power-over-Ethernet or an outlet adapter to flexibly choose deployment locations across the campus. The choice of devices and OS enables us to run

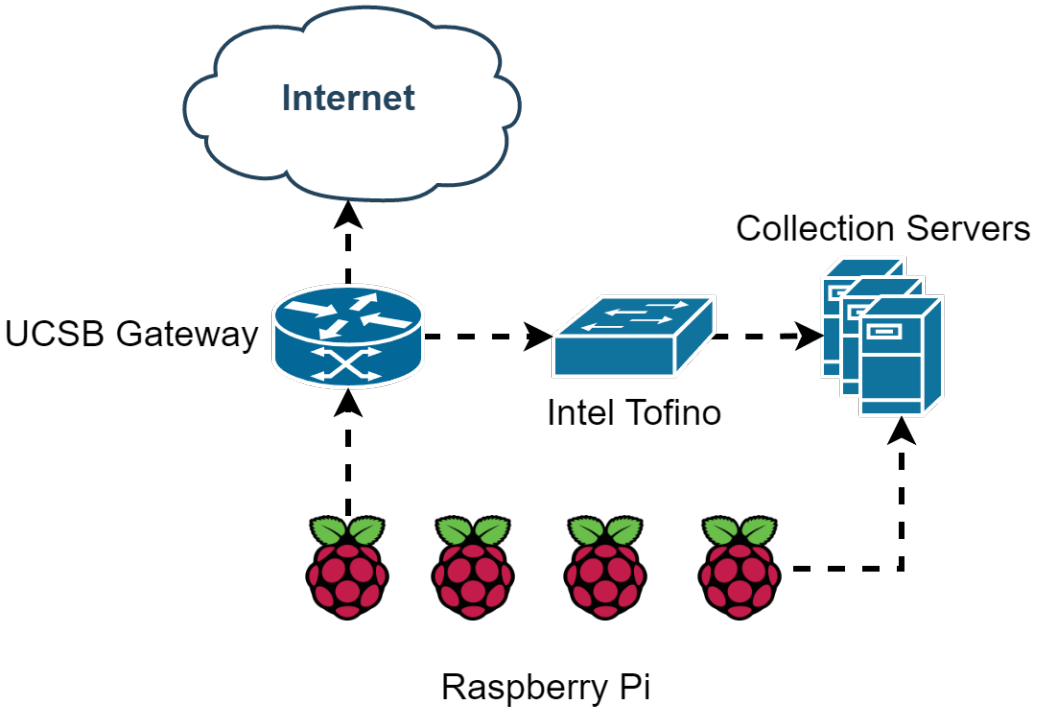


Figure 4.1: Simplified view of the PINOT infrastructure.

arbitrary network experiments, from simple ones such as pings to more complex scenarios such as synchronized simulations of network attacks.

To control these devices, we deployed a SaltStack configuration management system [208] with agents installed on our nodes. This system allows us to control nodes remotely, automatically deploy and start network experiments, and collect produced artifacts remotely. SaltStack's agent-based approach is also particularly well suited for NAT-based network environments so that devices can be deployed without publicly available addresses. We also use our `netunicorn` platform [29] to implement network experiments and seamlessly switch between physical infrastructures and virtual alternatives (e.g., Containernet [189] or clouds).

We deployed these devices across UCSB's campus, where the university operates a campus-wide Wi-Fi infrastructure for all students, available via access points in each

building and facility of the university. During the deployment, we connect our nodes to both wired access ports and the public Wi-Fi network, sharing the network connection with all university students. This allows us to mimic a typical network user in the campus network and obtain instances of representative real-world measurements from different physical locations and under a range of different network conditions.

During the first phase of our deployment, we identified the most populated and traffic-heavy areas within the campus (e.g., student dormitories, libraries, university centers, and dining halls). In total, we deployed over 50 devices in these locations that are now online and allow us to perform various measurement experiments and tests. We also provided each device with a QR code that links to a device portal where we provide different statistics and results of background measurements to other users at this location.

4.4 Discussion & Future Research

To further democratize ML-based networking research [107], we made the PINOT platform publicly available. We offer access to the platform based on principles of fair and responsible usage. Compared to existing alternatives [19, 174], we support arbitrary experiments and allow to implement them with `netunicorn` platform.

Users are able to leverage PINOT's active and passive data collection capabilities separately or in combination. While passively collected data has limitations (i.e., lack of payload and sensitive headers), it could be used for enriching labeled datasets with unlabeled user traffic or comparing captured traffic from multiple vantage points to investigate different latency issues or network anomalies.

At the same time, our use of an existing campus network infrastructure constrains the type of data collection scenarios that PINOT can support. For example, we cannot support specific data collection scenarios such as data-center-oriented traffic collections or

efforts that presume a particular type of network connectivity.

We encourage and make it possible for all users to participate in the platform development, either by receiving a physical node or by hosting a virtual node using our provided Docker container. Virtual nodes entail only minimal configuration efforts, require no special permissions, and only assume basic Internet access.

We maintain a project website [191] where we share information about the project, including details of our current installation at UCSB and up-to-date statistics and metadata webpages for each deployed device. For network experimenters and developers, we also share the OpenAPI endpoint [190] for our raw network data and detailed information on node location for more controlled and automated node selection for measurements.

Our ongoing and future research efforts focus on increasing the coverage within our campus network. We also plan to improve the flexibility of our network setup to provide more capabilities for potential network experiments by adding options to redirect the required traffic from our nodes through a single controlled routing point to implement different traffic-shaping solutions or emulate a range of network conditions and scenarios. You should build it to this problem.

Chapter 5

In Search of netUnicorn: A Data-Collection Platform to Develop Generalizable ML Models for Network Security Problems

5.1 Introduction

Machine learning-based methods have outperformed existing rule-based approaches for addressing different network security problems, such as detecting DDoS attacks [166], malwares [19, 23], network intrusions [78], etc. However, their excellent performance typically relies on the assumption that the training and testing data are independent and identically distributed. Unfortunately, due to the highly diverse and adversarial nature of real-world network environments, this assumption does not hold for most network security problems. For instance, an intrusion detection model trained and tested with data from a specific environment cannot be expected to be effective when deployed in a different

environment, where attack and even benign behaviors may differ significantly due to the nature of the environment. This inability of existing ML models to perform as expected in different deployment settings is known as *generalizability problem* [64], poses serious issues with respect to maintaining the models' effectiveness after deployment, and is a major reason why security practitioners are reluctant to deploy them in their production networks in the first place.

Recent studies (e.g., [18]) have shown that the quality of the training data plays a crucial role in determining the generalizability of ML models. In particular, in popular application domains of ML such as computer vision and natural language processing [236, 260], researchers have proposed several data augmentation and data collection techniques that are intended to improve the generalizability of trained models by enhancing the diversity and quality of training data [108]. For example, in the context of image processing, these techniques include adding random noise, blurring, and linear interpolation. Other research efforts leverage open-sourced datasets collected by various third parties to improve the generalizability of text and image classifiers.

Unfortunately, these and similar existing efforts are not directly applicable to network security problems. For one, since the semantic constraints inherent in real-world network data are drastically different from those in text or image data, simply applying existing augmentation techniques that have been designed for text or image data is likely to result in unrealistic and semantically incoherent network data. Moreover, utilizing open-sourced data for the network security domain poses significant challenges, including the encrypted nature of increasing portions of the overall traffic and the fact that without detailed knowledge of the underlying network configuration, it is, in general, impossible to label additional data correctly. Finally, due to the high diversity in network environments and a myriad of different networking conditions, randomly using existing data or collecting additional data without understanding the inherent limitations of the available training

data may even reduce data quality. As a result, there is an urgent need for novel data curation techniques that are specifically designed for the networking domain and aid the development of generalizable ML models for network security problems.

To address this need, we propose a new closed-loop ML pipeline (workflow) that focuses on training generalizable ML models for networking problems. Our proposed pipeline is a major departure from the widely-used standard ML pipeline [64] in two major ways. First, instead of obscuring the role that the training data plays in developing and evaluating ML models, the new pipeline elucidates the role of the training data. Second, instead of being indifferent to the black-box nature of the trained ML model, our proposed pipeline deliberately focuses on developing explainable ML models. To realize our new ML pipeline, we designed it using a closed-loop approach that leverages a novel data collection platform (called netUnicorn[1]) in conjunction with state-of-the-art explainable AI (XAI) tools so as to be able to iteratively collect new training data for the purpose of enhancing the ability of the trained models to generalize. Here, during each iteration, the insights obtained from applying the employed explainability tools to the current version of the trained model are used to synthesize new policies for exactly what kind of new data to collect in the next iteration so as to combat generalizability issues affecting the current model.

In designing and implementing netUnicorn, the novel data collection platform that our proposed ML pipeline relies on, we leveraged state-of-the-art programmable data-plane targets, programmable network infrastructures, and different virtualization tools to enable flexible data collection at scale from disparate network environments and for different learning problems without network operators having to worry about the details of implementing their desired data collection efforts. This platform can be envisioned as representing the “thin waist" of the classic hourglass model [26], where the different learning

[1] `https://netunicorn.cs.ucsb.edu`

problems comprise the top layer and the different network environments constitute the bottom layer. To realize this “thin waist" analog, netUnicorn supports a new programming abstraction that (i) decouples the data-collection intents or policies (i.e., answering what data to collect and from where) from the mechanisms (i.e., answering how to collect the desired data on a given platform); and (ii) disaggregates the high-level intents into self-contained and reusable subtasks.

In effect, our newly proposed ML pipeline advances the current state-of-the-art in ML model development by (1) augmenting the standard ML pipeline with an explainability step that impacts how we evaluate ML models' suitability for deployments, (2) leveraging existing explainable AI (XAI) tools to identify issues with the utilized training data that may affect a trained model's generalizability, and (3) utilizing insights from (2) to direct the netUnicorn-driven iterative data collection, thus gradually improving the models' generalizability using these new datasets. Unlike the standard “open-loop” ML pipelines, which often rely on synthetic data to bolster model generalizability or lack the means to gather data from varying network settings or distinct learning scenarios, our innovative closed-loop ML pipeline can consistently collect the “right” training data from diverse network conditions for any specified learning problem. This paper shows that the proposed ML pipeline's ability to iteratively collect the “right" training data from disparate network environments and for any given learning problem paves the way for the development of generalizable ML models for networking problems.

Contributions. This paper makes the following contributions:

- **An alternative ML pipeline.** We propose a novel closed-loop ML pipeline that leverages a new data-collection platform in conjunction with state-of-the-art explainability (XAI) tools to enable iterative and informed data collection to gradually improve the quality of the data used for model training and thus boost

the trained models' generalizability (Section 5.2).

- **A new data-collection platform.** We justify (Section 5.3) and present the design and implementation (Section 5.4) of netUnicorn, the new data-collection platform that is key to performing iterative and informed data collection for any given learning problem and from any network environment as part of our newly proposed closed-loop ML pipeline in practice. We made several design choices in netUnicorn to tackle the research challenges of realizing the "thin waist" abstraction.

- **An extensive evaluation.** We demonstrate the capabilities of netUnicorn and the effectiveness of our newly proposed ML pipeline by (i) considering various learning models for network security problems that have been studied in the existing literature and (ii) evaluating them with respect to their ability to generalize (Section 5.5 and Section 5.6).

- **Artifacts.** We make the full source code of the system, as well as the datasets used in this paper, publicly available. Specifically, we have released three repositories: full source code of netUnicorn [175], a repository of all discussed tasks and data-collection pipelines [176], and other supplemental materials [177] (see [29], Appendix I).

We consider the proposed ML pipeline and data-collection platform a promising foundation for creating generalizable ML-based network security solutions more likely to see practical deployment. Still, significant work lies ahead. To ensure model generalizability, we must thoroughly assess the network infrastructure for data collection and the nature of traffic in production settings.

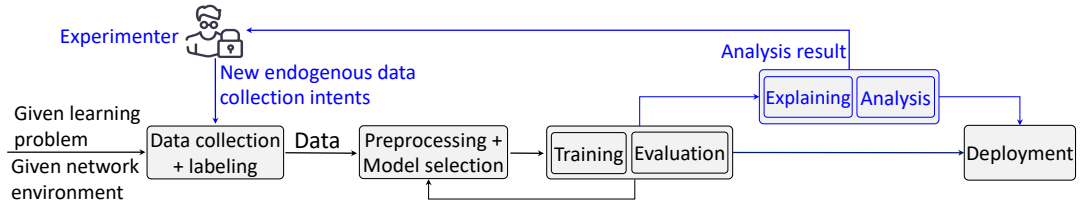


Figure 5.1: Overview of the existing (standard) and the newly-proposed (closed-loop) ML pipelines. The components marked in blue are our proposed augmentations to the standard ML pipeline.

5.2 Background and Problem Scope

5.2.1 Existing ML Pipeline for Network Security

Key components. The standard ML pipeline (see Figure 5.1) defines a workflow for developing ML artifacts and is widely used in many application domains, including network security. To solve a learning problem (e.g., detecting DDoS attack traffic), the first step is to collect (or choose) labeled data, select a model design or architecture (e.g., random forest classifier), extract related features, and then perform model training using the training dataset. An independent and identically distributed (iid) evaluation procedure is then used to assess the resulting model by measuring its expected predictive performance on test data drawn from the training distribution. The final step involves selecting the highest-performing model from a group of similarly trained models based on one or more performance metrics (e.g., F1-score). The selected model is then considered the ML-based solution for the task at hand and is recommended for deployment and being used or tested in production settings.

Data collection mechanisms. As in other application areas of ML, the collection of appropriate training data is of paramount importance for developing effective ML-based network security solutions. In network security, the standard ML pipeline integrates two basic data collection mechanisms: *real-world network data* collection and *emulation-based*

network data collection.

In the case of real-world network data collection, data such as traffic-specific aspects are extracted directly (and usually passively) from a real-world target network environment. While this method can provide datasets that reflect pertinent attributes of the target environment, issues such as encrypted network traffic and user privacy considerations pose significant challenges to understanding the context and correctly labeling the data. Despite an increasing tendency towards traffic encryption [51], this approach still captures real-world networking conditions but often restricts the quality and diversity of the resulting datasets.

Regarding emulation-based network data collection, the approach involves using an existing or building one's own emulated environment of the target network and generating (usually actively) various types of attack and benign traffic in this environment to collect data. Since the data collector has full control over the environment, it is, in general, easy to obtain ground truth labels for the collected data. While created in an emulated environment, the resulting traffic is usually produced by existing real-world tools. Many widely used network datasets, including the still-used DARPA1998 dataset [1] and the more recent CIC-IDS intrusion detection datasets [58] have been collected using this mechanism.

5.2.2 Model Generalizability Issues

Although existing emulation-based mechanisms have the benefit of providing datasets with correct labels, the training data is often riddled with problems that prevent trained models from generalizing, thus making them ill-suited for real-world deployment.

There are three main reasons why these problems can arise. First, network data is inherently complex and heterogeneous, making it challenging to produce datasets that do

not contain inductive biases. Second, emulated environments typically differ from the target environment – without full knowledge of the target environment's configurations, it is difficult to accurately mimic it. The result is datasets that do not fully represent all the target environment's attributes. Third, shifting attack (or even benign) behavior is the norm, resulting in training datasets that become less representative of newly created testing data after the model is deployed.

These observations motivate considering the following concrete issues concerning the generalizability of ML-based network security solutions but note that there is no clear delineation between notions such as *credible*, *trustworthy* or *robust* ML models and that the existing literature tends to blur the line between these (and other) notions and what we refer to as model generalizability.

Shortcut learning. As discussed in [18], ML-based security solutions often suffer from shortcuts. Here, shortcuts refer to encoded/inductive biases in a trained model that stem from false or non-causal associations in the training dataset [93]. These biases can lead to a model not performing as desired in deployment scenarios, mainly because the test datasets from these scenarios are unlikely to contain the same false associations. Shortcuts are often attributable to data-collection issues, including how the data was collected (intent) or from where it was collected (environment). Recent studies have shown that shortcut learning is a common problem for ML models trained with datasets collected from emulated networking environments. For example, [126] found that the reported high F1-score for the VPN vs. non-VPN classification problem in [77] was due to a specific artifact of how this dataset was curated.

Out-of-distribution issues. Due to unavoidable differences between a real-world target environment and its emulated counterpart or subtle changes in attack and/or benign behaviors, out-of-distribution (ood) data is another critical factor that limits model generalizability. The standard ML pipeline's evaluation procedure results in models that

may appear to be well-performing, but their excellent performance can often be attributed to the models' innate ability for "rote learning", where the models cannot transfer learned knowledge to new situations. To assess such models' ability to learn beyond iid data, purposefully curated ood datasets can be used.

For network security problems, ood datasets of interest can represent different real-world network conditions (e.g., different user populations, protocols, applications, network technologies, architectures, or topologies) or different network situations (also referred to as distribution shift [195] or concept drift [151]). For determining whether or not a trained model generalizes to different scenarios, it is important to select ood datasets that accurately reflect the different conditions that can prevail in those scenarios.

5.2.3 Existing Approaches

We can divide the existing approaches to improving a model's generalizability into two broad categories: (1) Efforts for improving model selection, training, and testing algorithms; and (2) Efforts for improving the training datasets. The first category focuses mainly on the later steps in the standard ML pipeline (see Figure 5.1) that deal with the model's structure, the algorithm used for training, and the evaluation process. The second category is concerned with improving the quality of datasets used during model training and focuses on the early steps in the standard ML pipeline.

Improving model selection, training, and evaluation. The focal point of most existing efforts is either the model's structure (e.g., domain adaption [86, 212] and multi-task learning [205, 262]), or the training algorithm (e.g., few-shot learning [101, 202]), or the evaluation process (e.g., ood detection [130, 257]). However, they neglect the training dataset, mainly because it is in general assumed to be fixed and already given. While these efforts provide insights into improving model generalizability, studying the

problem without the ability to actively and flexibly change the training dataset is difficult, especially when the given training dataset turns out to exhibit inductive biases, be noisy or of low quality, or simply be non-informative for the problem at hand [108]. See Section 5.8 for a more detailed discussion about existing model-based efforts and how they differ from our proposed approach described below.

Improving the training dataset. Data augmentation is a passive method for synthesizing new or modifying existing training datasets and is widely used in the ML community to improve models' generalizability. Technically, data augmentation methods leverage different operations (e.g., adding random noise [236], using linear interpolations [260] or more complex techniques) to synthesize new training samples for different types of data such as images [219, 236], text [260], or tabular data [52, 134]. However, using such passive data-generation methods for the network security domain is inappropriate or counterproductive because they often result in unrealistic or even semantically meaningless datasets [95]. For example, since network protocols usually adhere to agreed-upon standards, they constrain various network data in ways that such data-generation methods cannot ensure without specifically incorporating domain knowledge. Furthermore, various network environments can induce significant differences in observed communication patterns, even when using the same tools or considering the same scenarios [81], by influencing data characteristics (e.g., packet interarrival times, packet sizes, or header information) and introducing unique network conditions or patterns.

5.2.4 Limitations of Existing Approaches

From a network security domain perspective, these existing approaches miss out on two aspects that are intimately related to improving a model's ability to generalize: (1) Leveraging insights from model explainability tools, and (2) ensuring the realism of

collected training datasets.

Using explainable ML techniques. To better scrutinize an ML model's weaknesses and understand model errors, we argue that an additional explainability step that relies on recent advances in explainable ML should be added to the standard ML pipeline to improve the ML workflow for network security problems [106, 126, 188, 217]. The idea behind adding such a step is that it enables taking the output of the standard ML pipeline, extracting and examining a carefully-constructed white-box model in the form of a decision tree, and then scrutinizing it for signs of blind spots in the output of the standard ML pipeline. If such blind spots are found, the decision tree and an associated summary report can be consulted to trace their root causes to aspects of the training dataset and/or model specification that led the output to encode inductive biases.

Ensuring realism in collected training datasets. To beneficially study model generalizability from the training dataset perspective, we posit that for the network security domain, the collection of training datasets should be done *endogenously* or *in vivo*; that is, performed or taking place within the network environment of interest. Given that network-related datasets are typically the result of intricate interactions between different protocols and their various embedded closed control loops, accurately reflecting these complexities associated with particular deployment settings or traffic conditions requires collecting the datasets from within the network.

5.2.5 Our Approach in a Nutshell

We take a first step towards a more systematic treatment of the model generalizability problem and propose an approach that (1) uses a new closed-loop ML pipeline and (2) calls for running this pipeline in its entirety multiple times, each time with a possibly different model specification but always with *a different training dataset* compared to

the original one. Here, we use a newly-proposed closed-loop ML pipeline (Figure 5.1) that differs from the standard pipeline by including an explanation step. Also, each new training dataset used as part of a new run of the closed-loop ML pipeline is assumed to be endogenously collected and not exogenously manipulated.

The collection of each new training dataset is informed by a root cause analysis of identified inductive bias(es) in the trained model. This analysis leverages existing explainability tools that researchers have at their disposal as part of the closed-loop pipeline's explainability step. In effect, such an informed data-collection effort promises to enhance the quality of the given training datasets by gradually reducing the presence of inductive biases that are identified by our approach, thus resulting in trained models that are more likely to generalize. Note, however, that our proposed approach *does not guarantee* model generalizability. Instead, by eliminating identified inductive biases in the form of shortcuts and ood data, our approach enhances a model's generalizability capabilities. Also, note that our focus in this paper is not on designing novel model explainability methods but rather on applying available techniques from the existing literature. In fact, while we are agnostic about which explainability tools to use for this step, we recommend the application of global explainability tools such as Trustee [126] over local explainability techniques (e.g., [106, 153, 199, 238, 246]), mainly because the former are in general more powerful and informative with respect to faithfully detecting and identifying root causes of inductive biases compared to the latter. However, as shown in Section 5.5 below, either of these two types of methods can shed light on the nature of a trained model's inductive biases.

Our proposed approach differs from existing approaches in several ways. First, it reduces the burden on the user or domain expert to select the "right" training dataset apriori. Second, it calls for the collection of training datasets that are endogenously generated and where explainability tools guide the decision-making about what "better"

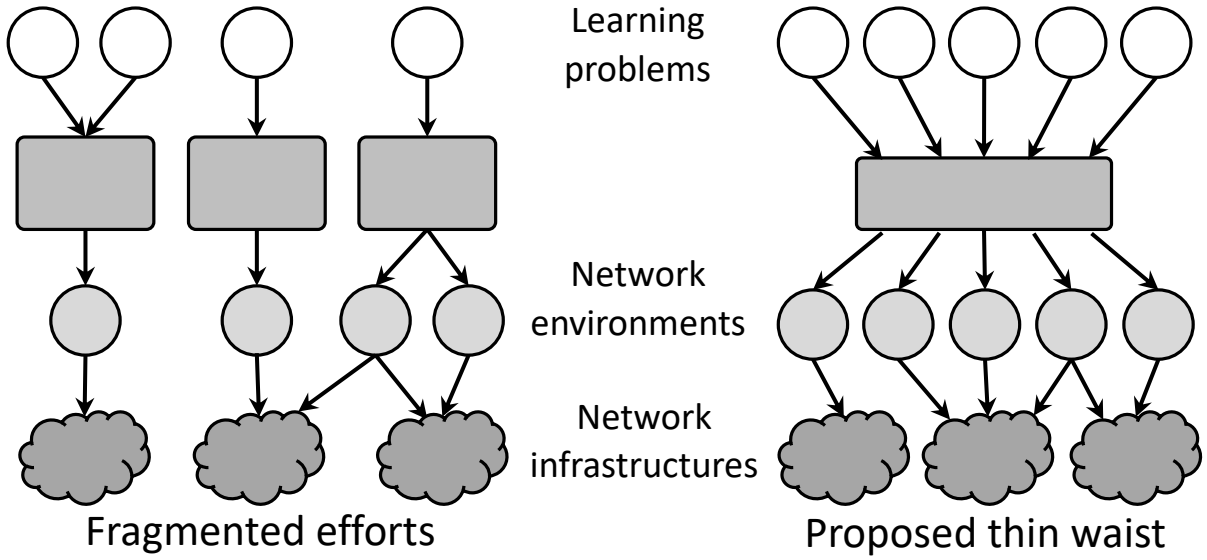


Figure 5.2: netUnicorn vs. existing data collection efforts.

data to collect. Third, it proposes using multiple training datasets, collected iteratively (in a fail-fast manner), to combat the underspecification of the trained models and thus enhance model generalizability. In particular, it recognizes that an "ideal" training dataset may not be readily available in the beginning and argues strongly against attaining it through exogenous means.

5.3 On "in vivo" Data-Collection

In this section, we discuss some of the main issues with existing data-collection efforts and describe our proposed approach to overcome their shortcomings.

5.3.1 Existing Approaches

Data collection operations. We refer to collecting data for a learning problem from a specific network environment (or domain) as a data-collection *experiment*. We divide such a data-collection experiment into three distinct operations. (1) *Specification:*

expressing the intents that specify what data to collect or generate for the experiment. (2) *Deployment:* bootstrapping the experiment by translating the high-level intents into target-specific commands and configurations across the physical or virtual data-collection infrastructure and implementing them. (3) *Execution:* orchestrating the experiment to collect the specified data while handling different runtime events (e.g., node failure, connectivity issues, etc.). Here, the first operation is concerned with "what to collect," and the latter operations deal with "how to collect" this data.

The "fragmentation" issue. Existing data-collection efforts are inherently *fragmented*, i.e., they only work for a specific learning problem and network environment, emulated using one or more network infrastructures (Figure 5.2). Extending them to collect data for a new learning problem or from a new network environment is challenging. For example, consider the data-collection effort for the video fingerprinting problem [209], where the goal is to fingerprint different videos for video streaming applications (e.g., YouTube) using a stream of encrypted network packets as input. Here, the data-collection intent is to start a video streaming session and collect the related packet traces from multiple end hosts that comprise a specific target environment. The deployment operation entails developing scripts that automate setting up the computing environment (e.g., installing the required selenium package) at the different end hosts. The execution operation requires developing a runtime system to start/stop the experiments and handle runtime events such as node failure, connectivity issues, etc.

Lack of modularity. In addition to being one-off in nature, existing approaches to collecting data for a given learning problem are also monolithic. That is, being highly problem-specific, there is, in general, no clear separation between experiment specification and mechanisms. An experimenter must write scripts that realize the data-collection intents (e.g., start/stop video streaming sessions, collect pcaps, etc.), deploy these scripts to one or more network infrastructures, and execute them to collect the required data.

Given this monolithic structure, existing data collection approaches [209] cannot easily be extended so that they can be used for a different learning problem, such as inferring QoE [40, 103, 111] or for a different network environment, such as congested environments (e.g., hotspots in a campus network) or high-latency networks (e.g., networks that use GEO satellites as access link).

Disparity between virtual and physical infrastructures. While a number of different network emulators and simulators are currently available to researchers [139, 173, 180, 255], it is, in general, difficult or impossible to write experiments that can be seamlessly transferred from a virtual to a physical infrastructure and back. This capability is particularly appealing in view of the fact that virtual infrastructures provide the ability to quickly iterate on data collection and test various network conditions, including conditions that are complex in nature and, in general, difficult to achieve in physical infrastructures. Due to the lack of this capability, experimenters often end up writing experiments for only one of these infrastructures, creating different (typically simplified) experiment versions for physical test beds, or completely rewriting the experiments to account for real-world conditions and problems (e.g., node and link failures, network synchronization)

Missed opportunity. Together, these observations highlight a missed opportunity for researchers who now have access to different network infrastructures. The list includes NSF-supported research infrastructures, such as EdgeNet [82], ChiEdge [49], Fabric [21], PAWR [186], etc., as well as on-demand infrastructure offered by different cloud services providers, such as AWS [46], Azure [164], Digital Ocean [47], GCP [48], etc. This rich set of network infrastructures can aid in emulating diverse and representative network environments for data collection.

5.3.2 An "Hourglass" Design to the Rescue

The observed fragmented, one-off, and monolithic nature of how training datasets for network security-related ML problems are currently collected motivates a new and more principled approach that aims at lowering the threshold for researchers wanting to collect high-quality network data. Here, we say a training dataset is of high quality if the model trained using this dataset is not obviously prone to inductive biases and, therefore, likely to generalize.

Our hourglass model. Our proposed approach takes inspiration from the classic "hourglass" model [26], a layered systems architecture that, in our case, consists of designing and implementing a "thin waist" that enables collecting data for different learning problems (hourglass' top layer) from a diverse set of possible network environments (hourglass' bottom layer). In effect, we want to design the thin waist of our hourglass model in such a way that it accomplishes three goals: (1) allows us to collect a specified training dataset for a given learning problem from network environments emulated using one or more supported network infrastructures, (2) ensures that we can collect a specified training set for each of the considered learning problems for a given network environment, and (3) facilitates experiment reproducibility and shareability.

Requirements for a "thin waist". Realizing this hourglass model's thin waste requires developing a flexible and modular data-collection platform that supports two main functionalities: (1) *decoupling* data-collection intents (i.e., expressing **what** to collect and from **where**) from mechanisms (i.e., **how** to realize these intents); and (2) *disaggregating* intents into independent and reusable tasks.

The required first functionality allows the experimenter to focus on the experiment's intent without worrying about how to implement it. As a result, expressing a data-collection experiment does not require re-doing tasks related to deployment and execution

in different network environments. For instance, to ensure that the learning model for video fingerprinting is not overfitted to a specific network environment, collecting data from different environments, such as congested campus networks or cable- and satellite-based home networks, is important. Not requiring the experimenter to specify the implementation details simplifies this process.

Providing support for the second functionality allows the experimenter to reuse common data-collection intents and mechanisms for different learning problems. For instance, while the goal for QoE inference and video fingerprinting may differ, both require starting and stopping video streaming sessions on an end host.

Ensuring these two required functionalities makes it easier for an experimenter to iteratively improve the data collection intent, addressing apparent or suspected inductive biases that a model may have encoded and may affect the model's ability to generalize.

5.4 Realizing the "Thin Waist" Idea

To achieve the desired "thin waist" of the proposed hourglass model, we develop a new data-collection platform, netUnicorn. We identify two distinct stakeholders for this platform: (1) *experimenters* who express data-collection intents, and (2) *developers* who develop different modules to realize these intents. In Section 5.4.1, we describe the programming abstractions that netUnicorn considers to satisfy the "thin" waist requirements, and in Section 5.4.2, we show how netUnicorn realizes these abstractions while ensuring fidelity, scalability, and extensibility.

5.4.1 Programming Abstractions

To satisfy the second requirement (*disaggregation*), netUnicorn allows experimenters to disaggregate their intents into distinct pipelines and tasks. Specifically, netUnicorn

offers experimenters `Task` and `Pipeline` abstractions. Experimenters can structure data collection experiments by utilizing multiple independent pipelines. Each pipeline can be divided into several processing stages, where each stage conducts self-contained and reusable tasks. In each stage, the experimenter can specify one or more tasks that netUnicorn will execute concurrently. Tasks in the next stage will only be executed once all tasks in the previous stage have been completed.

To satisfy the first requirement, netUnicorn offers a unified interface for all tasks. To this end, it relies on abstractions that concern specifics of the computing environment (e.g., containers, shell access, etc.) and executing target (e.g., ARM-based Raspberry Pis, AMD64-based computers, OpenWRT routers, etc.) and allows for flexible and universal task implementation.

To further decouple intents from mechanisms, netUnicorn's API exposes the `Nodes` object to the experimenters. This object abstracts the underlying physical or virtual infrastructure as a pool of data-collection nodes. Here, each node can have different static and dynamic attributes, such as type (e.g., Linux host, PISA switch), location (e.g., room, building), resources (e.g., memory, storage, CPU), etc. An experimenter can use the `filter` operator to select a subset of nodes based on their attributes for data collection. Each node can support one or more compute environments, where each environment can be a shell (command-line interpreter), a Linux container (e.g., Docker [72]), a virtual machine, etc. netUnicorn allows users to map pipelines to these nodes using the `Experiment` object and `map` operator. Then, experimenters can deploy and execute their experiments using the `Client` object. For details about the key components of netUnicorn's API, see [29], Table 7.

Illustrative example. To illustrate with an example how an experimenter can use netUnicorn's API to express the data-collection experiment for a learning problem, we consider the bruteforce attack detection problem. For this problem, we need to realize

three pipelines, where the different pipelines perform the key tasks of running an HTTPS server, sending attacks to the server, and sending benign traffic to the server, respectively. The first pipeline also needs to collect packet traces from the HTTPS server.

Listing 5.1 shows how we express this experiment using netUnicorn. Lines 1-6 show how we select a host to represent a target server, start the HTTPS server, perform PCAP capture, and notify all other hosts that the server is ready. Lines 8-16 show how we can take hosts from different environments that will wait for the target server to be ready and then launch a bruteforce attack on this node. Lines 18-26 show how we select hosts that represent benign users of the HTTPS server. Finally, lines 28-32 show how we combine pipelines and hosts into a single experiment, deploy it to all participating infrastructure nodes, and start execution.

Note that in Listing 5.1 we omitted task definitions and instantiation, package imports, client authorization, and other details to simplify the exposition of the system.

5.4.2 System Design

The netUnicorn compiles high-level intents, expressed using the proposed programming abstraction, into target-specific programs. It then deploys and executes these programs on different data-collection nodes to complete an experiment. netUnicorn is designed to realize the high-level intents with *fidelity*, minimize the inherent computing and communication overheads (*scalability*), and simplify supporting new data-collection tasks and infrastructures for developers (*extensibility*).

Ensuring high fidelity. netUnicorn is responsible for compiling a high-level experiment into a sequence of target-specific programs. We divide these programs into two broad categories for each task: deployment and execution. The deployment definitions help configure the computing environment to enable the successful execution of a task. For

```
# Target server
h1 = Nodes.filter('location', 'azure').take(1)
p1 = Pipeline()
     .then(start_http_server)
     .then(start_pcap)
     .then(set_readiness_flag)

# Malicious hosts
h2 = [
  Nodes.filter('location', 'campus').take(40),
  Nodes.filter('location', 'aws').take(40),
  Nodes.filter('location', 'digitalocean').take(40),
]
p2 = Pipeline()
     .then(wait_for_readiness_flag)
     .then(patator_attack)

# Benign hosts
h3 = [
  Nodes.filter('location', 'campus').take(40),
  Nodes.filter('location', 'aws').take(40),
  Nodes.filter('location', 'digitalocean').take(40),
]
p3 = Pipeline()
     .then(wait_for_readiness_flag)
     .then(benign_traffic)

e = Experiment()
    .map(p1, h1)
    .map(p2, h2)
    .map(p3, h3)
Client().deploy(e).execute(e)
```

Listing 5.1: Data collection experiment example for the HTTPS bruteforce attack detection problem. We have omitted task instantiations and imports to simplify the exposition.

example, executing the `YouTubeWatcher` task requires installing a Chromium browser and related extensions. Since successful execution of each specified task is critical for satisfying the fidelity requirement, netUnicorn must ensure that the computing environment at the nodes is set up for a task before execution.

Addressing the scalability issues. To execute a given pipeline, a system can control deployment and execution either at the task- or pipeline-level granularity. The first option entails the deployment and execution of the *task* and then reporting results back to the

system before executing the next *task*. It ensures fidelity at the task granularity and allows the execution of pipelines even with tasks with contradicting requirements (e.g., different library versions). However, since such an approach requires communication with core system services, it slows the completion time and incurs additional computing and network communication overheads.

Our system implements the second option, running all the setup programs before marking a *pipeline* ready for execution and then offloading the task flow control to a node-based executor that reports results only at the end of the pipeline. It allows for optimization of environment preparation (e.g., configure a single docker image for distribution) and time overhead between tasks, and also reduces network communication while offering only "best-effort" fidelity for pipelines.

Enabling extensibility. Enabling extensibility calls for simplifying how a developer can add a new task, update an existing task for a new target, or add a new physical or virtual infrastructure. Note that the netUnicorn's extensibility requirement targets developers and not experimenters.

Simplify adding and updating tasks. An experimenter specifies a task to be executed in a pipeline. The netUnicorn chooses a specific implementation of this task. This may require customizing the computing environment, which can vary depending on the target (e.g., container vs shell of OpenWRT router). For example, a Chromium browser and specific software must be installed to start a video streaming session on a remote host without a display. The commands to do so may differ for different targets. The system provides a base class that includes all necessary methods for a task.

Developers can extend this base class by providing their custom subclasses with the target-specific `run` method to specify how to execute the task for different types of targets. This allows for easy extensibility because creating a new task subclass is all that is needed to adapt the task to a new computing environment.

Simplify adding new infrastructures. To deploy new data-collection pipelines, send commands, and send/receive different events and data to/from multiple nodes in the underlying infrastructure, netUnicorn requires an underlying deployment system.

One option is to bind netUnicorn to one of the existing deployment (orchestration) systems, such as Kubernetes [135], SaltStack [208], Ansible [13], or others for all infrastructures. However, requiring a physical infrastructure to support a specific deployment system is disruptive in practice. Network operators managing a physical infrastructure are often not amenable to changing their deployment system as it would affect other supported services.

Another option is to support multiple deployment systems. However, we need to ensure that supporting a new deployment system does not require a major refactoring of netUnicorn's existing modules. To this end, netUnicorn introduces a separate connectivity module that abstracts away all the connectivity issues from the netUnicorn's other modules (e.g., runtime), offering seamless connectivity to infrastructures using multiple deployment systems. Each time developers want to add a new infrastructure that uses an unsupported deployment system, they only need to update the connectivity manager — simplifying extensibility.

5.4.3 Prototype Implementation

Our implementation of netUnicorn is shown in Figure 5.3. Our implementation embraces a service-oriented architecture [200] and has three key components: *client(s)*, *core*, and *executor(s)*. Experimenters use local instances of netUnicorn's *client* to express their data-collection experiments. Then, netUnicorn's *core* is responsible for all the operations related to the compilation, deployment, and execution of an experiment. For each experiment, netUnicorn's core deploys a target-specific *executor* on all related data-

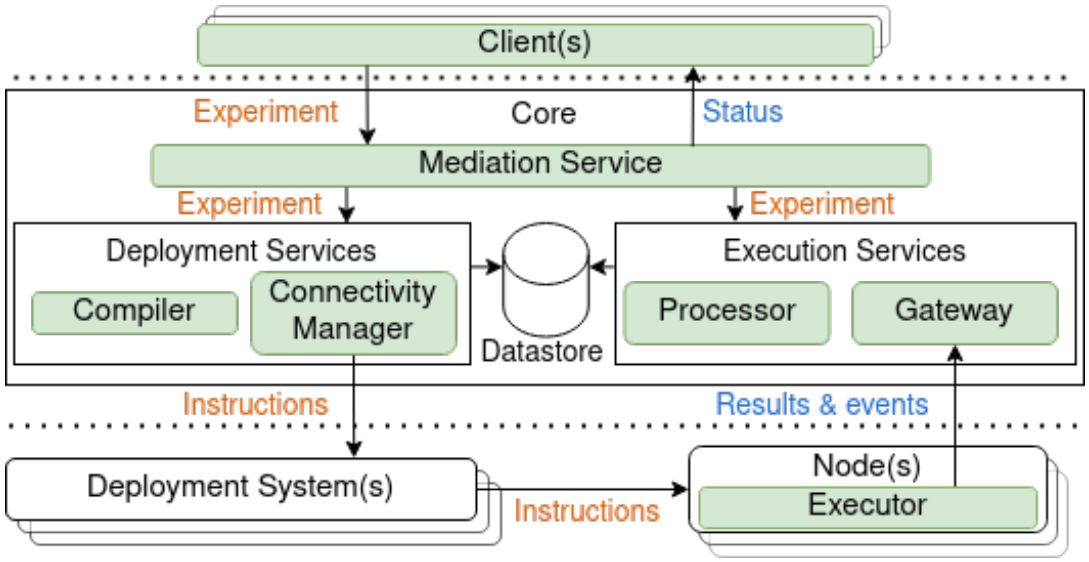


Figure 5.3: Architecture of the proposed system. Green-shaded boxes show all the implemented services.

collection nodes for running and reporting the status of all the programs provided by netUnicorn's core.

The netUnicorn's core offer three main service groups: mediation, deployment, and execution services. Upon receiving an experiment specification from the client, the `mediation service` requests the `compiler` to extract the set of setup configurations for each distinct (pipeline, node-type) pair, which it uploads to the local PostgreSQL database. After compilation, the `mediation service` requests the connectivity manager to ship this configuration to the appropriate data-collection nodes and verify the computing environment. In the case of docker-based infrastructures, this step is performed locally, and the configured docker image is uploaded to a local docker repository. The `connectivity-manager` uses an infrastructure-specific deployment system (e.g., SaltStack [208]) to communicate with the data-collection nodes.

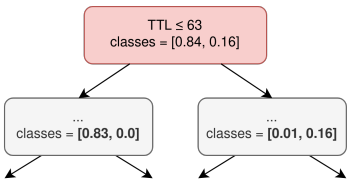
After deploying all the required instructions, the `mediation service` requests the connectivity manager to instantiate a target-specific `executor` for all data-collection

nodes. The `executor` uses the instructions shipped in the previous stage to execute a data-collection pipeline. It reports the status and results to netUnicorn's `gateway` and then adds them to the related table in the SQL database via the `processor`. The `mediation service` retrieves the status information from the database to provide status updates to the experimenter(s). Finally, at the end of an experiment, the `mediation service` sends cleanup scripts (via `connectivity-manager`) to each node—ensuring the reusability of the data-collection infrastructure across different experiments.

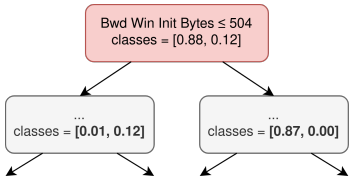
5.5 Evaluation: Closed-loop ML Pipeline

Table 5.1: Number of LLoC changes, data points, and F1 scores across different environments and iterations.

	Iteration #0 (initial setup)		Iteration 1		Iteration 2	
LLoCs	**80**		**+10**		**+20**	
	`UCSB-0` (train)	`multi-cloud` (test)	`UCSB-1` (train)	`multi-cloud` (test)	`UCSB-2` (train)	`multi-cloud` (test)
MLP	1.0	0.56	0.97 (-0.03)	0.62 (+0.06)	0.88 (-0.09)	**0.94 (+0.38)**
GB	1.0	0.61	1.0 (+0.00)	0.61 (+0.00)	0.92 (-0.08)	**0.92 (+0.31)**
RF	1.0	0.58	1.0 (+0.00)	0.69 (+0.11)	0.97 (-0.03)	**0.93 (+0.35)**
TN	1.0	0.66	0.97 (-0.03)	0.78 (+0.12)	0.92 (-0.05)	**0.95 (+0.29)**



(a) Iteration #0: top branch is a shortcut.



(b) Iteration #1: top branch is a shortcut.

(c) Iteration #2: no obvious shortcut.

Figure 5.4: Decision trees generated using Trustee [126] across the three iterations. We highlight the nodes that are indicators for shortcuts in the trained model.

In this section, we demonstrate how our proposed closed-loop ML pipeline helps to improve model generalizability. Specifically, we seek to answer the following questions: ❶ Does the proposed pipeline help in identifying and removing shortcuts? ❷ How do

models trained using the proposed pipeline perform compare to models trained with existing exogenous data augmentation methods? ❸ Does the proposed pipeline help with combating ood issues?

5.5.1 Experimental Setup

To illustrate our approach and answer these questions, we consider the bruteforce example mentioned in Section 5.4.1 and first describe the different choices we made with respect to the ML pipeline and the iterative data-collection methodology.

Network environments. We consider three distinct network environments for data collection: a `UCSB` network, a hybrid `UCSB-cloud` setting, and a `multi-cloud` environment.

The `UCSB` network environment is emulated using a programmable data-collection infrastructure PINOT [28]. This infrastructure is deployed at a campus network and consists of multiple (40+) single-board computers (such as Raspberry Pis) connected to the Internet via wired and/or wireless access links. These computers are strategically located in different areas across the campus, including the library, dormitories, and cafeteria. In this setup, all three types of nodes (i.e., target server, benign hosts, and malicious hosts) are selected from end hosts on the campus network. The `UCSB-cloud` environment is a hybrid network that combines programmable end hosts at the campus network with one of three cloud service providers: AWS, Azure, or Digital Ocean.[2] In this setup, we deploy the target server in the cloud while running the benign and malicious hosts on the campus network. Lastly, the `multi-cloud` environment is emulated using all three cloud service providers with multiple regions. We deploy the target server on Azure and the benign and malicious hosts on all three cloud service providers.

Data collection experiment. The data-collection experiment involves three pipelines, namely target, benign, and malicious. Each of these pipelines is assigned to different

[2]Unless specified otherwise, we host the target server on Azure for this environment.

sets of nodes depending on the considered network environment. The target pipeline is responsible for deploying a public HTTPS endpoint with a real-world API that requires authentication for access. Additionally, this pipeline utilizes *tcpdump* to capture all incoming and outgoing network traffic. The benign pipeline emulates valid usage of the API with correct credentials, while the malicious pipeline attempts to obtain the service's data by brute-forcing the API using the Patator [184] tool and a predefined list of commonly used credentials [210].

Data pre-processing and feature engineering. We used CICFlowMeter [59] to transform raw packets into a feature vector of 84 dimensions for each unique connection (flow). These features represent flow-level summary statistics (e.g., average packet length, inter-arrival time, etc.) and are widely used in the network security community [62, 77, 214, 267].

Learning models. We train four different learning models. Two of them are traditional ML models, i.e., Gradient Boosting (GB) [170], Random Forest (RF) [39]. The other two are deep learning-based methods: Multi-layer Perceptron (MLP) [101], and attention-based TabNet model (TN) [16]. These models are commonly used for handling tabular data such as CICFlowMeter features [104, 220].

Explainability tools. To examine a model trained with a given training dataset for the possible presence of inductive biases such as shortcuts or ood issues, our newly proposed ML pipeline requires an explainability step that consists of applying existing model explainability techniques, be they global or local in nature, but what technique to use is left to the discretion of the user.

We illustrate this step by first applying a global explainability method. In particular, our method-of-choice is the recently developed tool Trustee [126], but other global model explainability techniques could be used as well, including PDP plots [90], ALE plots [14], and others [169, 178]. Our reasoning for using the Trustee tool is that for any trained

black-box model, it extracts a high-fidelity and low-complexity decision tree that provides a detailed explanation of the trained model's decision-making process. Together with a summary report that the tool provides, this decision tree is an ideal means for scrutinizing the given trained model for possible problems such as shortcuts or ood issues.

To compare, we also apply local explainability tools to perform the explainability step. More specifically, we consider the two well-known techniques, LIME [199] and SHAP [153]. These methods are designed to explain a model's decision for individual input samples and thus require analyzing the explanations of multiple inputs to make conclusions about the presence or absence of model blind spots such as shortcuts or ood issues. While users are free to replace LIME or SHAP with more recently developed tools such as xNIDS [246] or their own preferred methods, they have to be mindful of the efforts each method requires to draw sound conclusions about certain non-local properties of a given trained model (e.g., shortcut learning).

5.5.2 Identifying and Removing Shortcuts

To answer ❶, we consider a setup where a researcher curates training datasets from the `UCSB` environment and aims at developing a model that generalizes to the `multi-cloud` environment (i.e., unseen domain).

Initial setup (iteration #0). We denote the training data generated from this experiment as `UCSB-0`. Table 5.1 shows that while all three models have a perfect training performance, they all have low testing performance (errors are mainly false positives). We first used our global explanation method-of-choice, Trustee, to extract the decision tree of the trained models. As shown in Figure 5.4, the top node is labeled with the separation rule ($TTL \leq 63$) and the balance between the benign and malicious samples in the data ("classes"). Subsequent nodes only show the class balance after the split.

From Figure 5.4a, we conclude that all four models use almost exclusively the TTL (time-to-live) feature to discriminate between benign and malicious flows, which is an obvious shortcut. Note that the top parts of Trustee-extracted decision trees were identical for all four models. When applying the local explanation tools LIME and SHAP to explain 100 randomly selected input samples, we found that these explanations identified TTL as the most important feature in all 100 samples. While consistent with our Trustee-derived conclusion, these LIME- or SHAP-based observations are necessary but not sufficient to conclusively decide whether or not the trained models learned a TTL-based shortcut strategy and further efforts would be required to make that decision.

To understand the root cause of this shortcut, we checked the `UCSB` infrastructure and noticed that almost all nodes used for benign traffic generation have the exact same TTL value due to a flat structure of the `UCSB` network. This observation also explains why most errors are false positives, i.e., the model treats a flow as malicious if it has a different TTL from the benign flows in the training set. Existing domain knowledge suggests that this behavior is unlikely to materialize in more realistic settings such as the `multi-cloud` environment. Consequently, we observe that models trained using the `UCSB-0` dataset perform poorly on the unseen domain; i.e., they generalize poorly.

Removing shortcuts (iteration #1). To fix this issue, we modified the data-collection experiment to use a more diverse mix of nodes for generating benign and malicious traffic and collected a new dataset, `UCSB-1`. However, this change only marginally improved the testing performance for all three models (Table 5.1). Inspection of the corresponding decision trees shows that all the models use the "Bwd Init Win Bytes" feature for discrimination, which appears to be yet another shortcut. Again, we observed that all trees generated by Trustee from different black-box models have identical top nodes. Similar, our local explanation results obtained by LIME and SHAP also point to this feature as being the most important one across the analyzed samples.

More precisely, this feature quantifies the TCP window size for the first packet in the backward direction, i.e., from the attacked server to the client. It acts as a flow control and reacts to whether the receiver (i.e., HTTP endpoint) is overloaded with incoming data. Although it could be one indicator of whether the endpoint is being brute-force attacked, it should only be weakly correlated with whether a flow is malicious or benign. Given this reasoning and the poor generalizability of the models, we consider the use of this feature to be a shortcut.

Removing shortcuts (iteration #2). To remove this newly identified shortcut, we refined the data-collection experiment. First, we created a new task that changes the workflow for the Patator tool. This new version uses a separate TCP connection for each bruteforce attempt and has the effect of slowing down the bruteforce process. Second, we increased the number of flows for benign traffic and the diversity of benign tasks. Using these changes, we collected a new dataset, `UCSB-2`.

Table 5.1 shows that the change in data-collection policy significantly improved the testing performance for all models. We no longer observe any obvious shortcuts in the corresponding decision tree. Moreover, domain knowledge suggests that the top three features (i.e., "Fwd Segment Size Average", "Packet Length Variance", and "Fwd Packet Length Std") are meaningful and their use can be expected to accurately differentiate benign traffic from repetitive brute force requests. Applying the local explanation methods LIME and SHAP also did not provide any indications of obvious additional shortcuts. Note that although the models appear to be shortcut-free, we cannot guarantee that the models trained with these diligently curated datasets do not suffer from other possible encoded inductive biases. Further improvements of these curated datasets might be possible but will require more careful scrutiny of the obtained decision trees and possibly more iterations.

Table 5.2: F1 score of models trained using our approach (i.e., leveraging netUnicorn) vs. models trained with datasets collected from the `UCSB` network by exogenous methods (i.e., without using netUnicorn).

	Iteration #0				Iteration #1				**Iteration #2**			
	MLP	GB	RF	TN	MLP	GB	RF	TN	MLP	GB	RF	TN
Naive Aug.	0.51	0.57	0.56	0.53	0.73	0.67	0.71	0.82	-	-	-	-
Noise Aug.	0.66	0.68	0.67	0.66	0.72	0.83	0.76	0.82	-	-	-	-
Feature Drop	0.74	0.55	0.72	**0.87**	**0.91**	0.58	0.63	**0.89**	-	-	-	-
SYMPROD	0.66	0.71	0.67	0.41	0.69	0.66	0.75	0.67	**0.94**	**0.93**	**0.95**	**0.96**
Our approach									**0.94**	**0.92**	**0.95**	**0.95**

5.5.3 Comparison with Exogeneous Methods

To answer ❷, we compare the performance of the model trained using `UCSB-2` (i.e., the dataset curated after two rounds of iterations) with that of models trained with datasets modified by means of existing exogenous methods. Specifically, we consider the following methods:

1. **Naive augmentation.** We use a naive data collection strategy that does not apply the extra explanation step that our newly proposed ML pipeline includes to identify training data-related issues. The strategy simply collects more data using the initial data-collection policy. It is an ablation study demonstrating the benefits of including the explanation step in our new pipeline. Here, for each successive iteration, we double the size of the training dataset.

2. **Noise augmentation.** This popular data augmentation technique consists of adding suitable chosen random uniform noise [155] to the identified skewed features in each iteration. Here, for iteration #0, we use integer-valued uniformly-distributed random samples from the interval $[-1; +1]$ for TTL noise augmentation, and for iteration #1, we similarly use integer-valued uniformly-distributed samples from the interval $[-5; +5]$ for noise augmentation of the feature “Bwd Init Win Bytes".

3. **Feature drop.** This method simply drops a specified skewed feature from the

dataset in each iteration. In our case, we drop the identified skewed feature for all training samples in each training dataset.

4. **SYMPROD.** SMOTE [52] is a popular augmentation method for tabular data that applies interpolation techniques to synthesize data points to balance the data across different classes. Here we utilize a recently considered version of this method called SYMPROD [136] and augment each training set by adding the number of rows necessary for restoring class balance ($proportion = 1$).

We apply these methods to the three training datasets curated from the campus network in the previous experiment. For `UCSB-0` and `UCSB-1`, we use the two identified skewed features for adding noise or dropping features altogether.

Note that since we did not identify any skewed features in the last iteration, we did not apply any noise augmentation and feature drop techniques in this iteration and did not collect more data for the naive data augmentation method.

As shown in Table 5.2, the models trained using these exogenous methods perform poorly in all iterations when compared to our approach. This highlights the main benefit we gain from applying our proposed closed-loop ML pipeline for iterative data collection and model training. In particular, it demonstrates that the explanation step in our proposed pipeline adds value. While doing nothing (i.e., naive data augmentation) is clearly not a worthwhile strategy, applying either noise augmentation or SYMPROD can potentially compromise the semantic integrity of the training data, making them ill-suited for addressing model generalizability issues for network security problems.

5.5.4 Combating ood-specific Issues

To answer ❸, we consider two different scenarios: *attack adaptation* and *environment adaptation*.

Table 5.3: The testing F1 score of the models before and after retraining with malicious traffic generated by Hydra.

	MLP	GB	RF	TN	Avg
Before retraining	0.87	0.81	0.86	0.83	0.84
After retraining	**0.93**	**0.96**	**0.91**	**0.91**	**0.93**

Table 5.4: The F1 score of models trained using only `UCSB` data or data from `UCSB` and `UCSB`-**cloud** infrastructures.

	`UCSB`		`UCSB`-**cloud**	
	Training	Test	Training	Test
MLP	0.88	0.94	**0.95 (+0.07)**	**0.95 (+0.01)**
GB	0.92	0.92	**0.96 (+0.04)**	**0.95 (+0.03)**
RF	0.97	0.93	**0.96 (-0.01)**	**0.97 (+0.04)**
TN	0.83	0.95	**0.84 (+0.01)**	**0.96 (+0.01)**

Attack adaptation. We consider a setup where an attacker changes the tool used for the bruteforce attack, i.e., uses Hydra [121] instead of Patator. To this end, we use netUnicorn to generate a new testing dataset from the `UCSB` infrastructure with Hydra as the bruteforce attack. Table 5.3 shows that the model's testing performance drops significantly (to 0.85 on average). We observe that this drop is because of the model's reduced ability to identify malicious flows, which indicates that changing the attack generation tool introduces oods, although they belong to the same attack type.

To address this problem, we modified the data generation experiment to collect attack traffic from both Hydra and Patator in equal proportions. This change in the data-collection experiment only required **6** LLoC. We retrain the models on this dataset and observe significant improvements in the model's performance on the same test dataset after retraining (see Table 5.3).

Note that we only test one type of oods where the evolved attack still has the same goal and functionality. However, an attack can also evolve into another attack with a different goal, resulting in ood samples with new labels. Here, we leverage ensemble

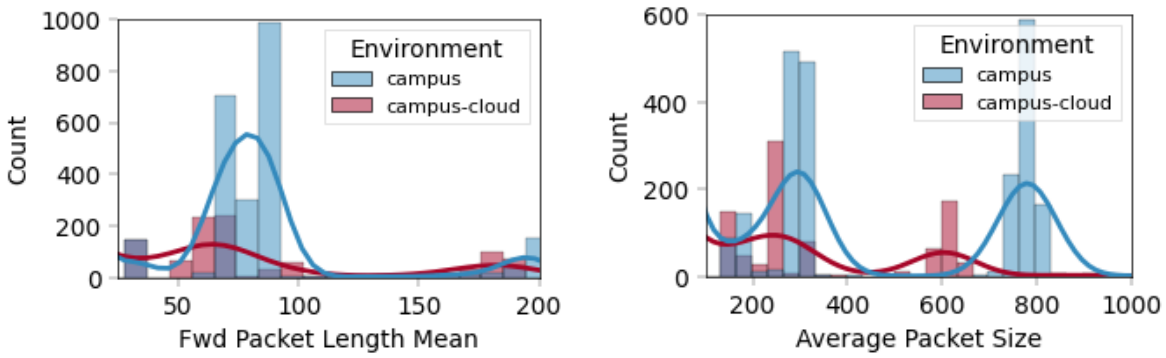


Figure 5.5: Distributions of several features across two different environments: `UCSB` and `UCSB-cloud`

models and human analysis to identify the ood case. While it may be possible to identify ood issues using more automated methods that are motivated by findings obtained from applying global explainability tools, we plan to revisit this problem in our future work.

Environment adaptation. We consider testing the model we developed in the `UCSB` environment in the unseen `multi-cloud` environment as a different instance of an ood issue that is due to possible feature distribution differences. To address this issue, we use the `UCSB-cloud` environment for data collection. As expected, we observe differences in the distributions for some of the features across the two environments (see Figure 5.5). Table 5.4 shows the performance of the models trained using only the data from the `UCSB` environment compared to the ones that use data from both the `UCSB` and `UCSB-cloud` environments. Notably, as `UCSB-cloud` is more similar to the `multi-cloud` environment than the `UCSB` environment, the models trained with the `UCSB-cloud` data show improvements in their performance under the test settings.

5.6 Evaluation: netUnicorn

We answer if netUnicorn lowers the threshold for data collection for: ❹ different learning problems for a given network environment? ❺ a given learning problem from different environments, emulated using one or more network infrastructures? and ❻ iteratively calibrating the data collection intents for a given learning problem and environment? We also demonstrate ❼ how well does netUnicorn scale for larger data-collection infrastructures, especially the ones equipped with relatively low-end devices, such as RPis?

5.6.1 Experimental Setup

Learning problems. Besides the *HTTP bruteforce attack detection* problem, we explore two more learning problems for this experiment, namely *video fingerprinting* and *advanced persistent threats detection* (APTs). In the case of the first additional example, the learning problem is to fingerprint videos for web-based streaming services, such as YouTube, that adopt variable bitrates [209]. Previous work [209] did not evaluate the proposed learning model under realistic network conditions. Thus, to collect meaningful data for this problem, we use a network of end hosts in the `UCSB` infrastructure to collect a training dataset for five different YouTube videos.[3] Specifically, our data-collection intent is specified by the following sequence of tasks: start packet capture, watch a YouTube video in headless mode for 30 seconds, and stop packet capture. We repeat this sequence ten times for each video in a shuffled order and combine it into a single pipeline, where at the end, we upload the collected data to our server.

Regarding the second additional example, the learning problem, in this case, is to identify the hosts that some APTs have compromised. To generate data for this learning problem, we write an experiment that mimics the behavior of a compromised host.

[3]Each video is identified with a unique URL.

Specifically, our data-collection intent is as follows: find active hosts using `Ping`, check if port 443 is opened for active hosts (identified in the previous stage) with `PortScan`, and then for each host with open 443 port launch four different attacks in parallel: `CVE20140160` (Heartbleed), `CVE202141773` (Apache 2.4.49 Path), `CVE202144228` (Log4J), and `Patator` (HTTP admin endpoint bruteforce using the Patator tool). The ML pipeline creates a "semi-realistic" training dataset by combining actively generated attack traffic with passively collected packet traces from a border router of a production network, such as the `UCSB` network.[4] We then use this dataset for model training. Note, here we assume that we know the attacker's playbook; that is, the goal, in this case, is not to demonstrate a realistic attack playbook but to demonstrate that netUnicorn simplifies generating attack traffic for a given APT attack playbook.

Network environments. netUnicorn enables emulating network environments for data collection using one or more physical/virtual infrastructures. Previously, we used a SaltStack-based infrastructure at `UCSB` and multiple clouds to emulate various network environments: `UCSB`, `UCSB-cloud`, and `multi-cloud`. In this experiment, we implement a connector to another infrastructure, Azure Container Instances (ACI) to expand cloud-based environments with serverless Docker containers. During the experiments, containers were dynamically created in multiple regions and used for pipeline execution. Overall, netUnicorn currently supports six different deployment system connectors (see [29], Table 8, for more details).

Baseline. To the best of our knowledge, none of the existing platforms/systems offer the desired extensibility, scalability, and fidelity for data collection (see Section 5.8 for more details). To illustrate how netUnicorn simplifies data collection efforts, we consider base-

[4]Note, in theory, we could use netUnicorn to actively collect the benign traffic for this learning problem in addition to the attack traffic. However, generating representative benign traffic for a large and complex enterprise network will require a more complex data-collection infrastructure than the one we use for evaluation. Section 5.7 discusses this issue in greater detail.

lines that directly configure three different deployment/orchestration systems. Specifically, we consider the following deployment systems as baselines: **Kubernetes**, **SaltStack**, and **Azure Container Instances (ACI)**. For each data-collection experiment, we explicitly compose different tasks to realize different data-collection pipelines, create pipeline-specific docker images, and use existing tools (e.g., *kubectl*) to map and deploy these pipelines to different nodes.

5.6.2 Simplifying Data Collection Effort

We now demonstrate how netUnicorn simplifies data collection for:

Different learning problems for a given network environment (❹). Table 5.5 reports the effort in expressing the data-collection experiments for the three learning problems for the `UCSB` network. We observe that netUnicorn only requires 17-35 LLoCs to express the data-collection intent. The `UCSB` network infrastructure uses SaltStack as the deployment system, and we observe that it takes 113-237 LLoC (around 5-13 × more effort) to express and realize the same data-collection intents without netUnicorn.

The key enabler here is the set of self-contained tasks that realize different data-collection activities. For each learning problem, Table 5.5 quantifies the overhead of specifying new tasks unique to the problem at hand. Even taking the overheads of expressing these tasks into consideration, collecting the same data from `UCSB` network without netUnicorn requires around 2-3 × more effort.

Overall, we implemented around twenty different tasks to bootstrap netUnicorn (see the extended version of the paper [29] for more details). The total development effort for the bootstrapping was around **900 LLoCs** . Though this bootstrapping effort is not insignificant, we posit that this effort amortizes over time as this repository of reusable and self-contained tasks will facilitate expressing increasingly disparate data-collection

experiments.

Table 5.5: LLoCs to implement different problems using netUnicorn and other deployment systems. Here, the three learning problems are (1) Bruteforce detection, (2) video fingerprinting, and (3) APT detection.

Learning Problems	**netUnicorn** Experiment (Tasks)	**Other Deployment Systems** Kubernetes	SaltStack	ACI
1	**21** (18)	74	113	61
2	**35** (115)	161	237	179
3	**17** (120)	151	232	176
LLoC Ratio for Experiments + Tasks		$1-2\times$	$2-3\times$	$1-2\times$
LLoC Ratio for Experiments		$3-9\times$	$5-13\times$	$3-10\times$

Given learning problem from multiple network environments (❺). As we discussed before, netUnicorn is inherently extensible, i.e., it can use different sets of network infrastructures to emulate disparate network environments for data collection. With netUnicorn, changing an existing data-collection experiment to collect data from a new set of network infrastructure(s) requires changing only a few LLoCs (2-3 for the examples in Table 5.5). In contrast, collecting the data for the HTTP Bruteforce detection problem from a cloud infrastructure (ACI) and a Kubernetes cluster requires writing additional 61 and 74 LLoCs, respectively. This effort is even more intense for video fingerprinting and APT detection problems.

The key enabler for simplifying data collection across one or more network infrastructures is netUnicorn's extensible connectivity manager that can interface with multiple deployment systems via a system of connectors. In [29], we provide a table where we enumerate all the implemented connectors and corresponding logical lines of code (LLoC) for each implementation. Note that this bootstrapping is a one-time effort, and these connectors can be reused across multiple physical infrastructures that are managed using either of the supported deployment systems (e.g., SaltStack, Kubernetes, etc.).

Iterative data collection (❻). To iteratively modify data collection intents, the system

should allow flexibility in both pipeline modifications and environment changes. We implemented the experiment, described in Section 5.5, using netUnicorn, for all three environments (`UCSB`, `UCSB-cloud`, and `multicloud`). We report the combined LLoCs for experiment definitions and tasks implementations in Table 5.1. As we reused previously implemented connectors, we do not report their LLoC in the table.

The table shows that the overhead for iterative updates is minimal. While this overhead may also be minimal for more conventional (platform- and problem-specific) solutions, netUnicorn's abstractions allow for seamless integration of many other platforms, thus providing a means to increase the diversity of the collected datasets further and, in turn, a model's generalizability capabilities.

5.6.3 Scaling Data Collection

To quantify the computing and memory overheads of netUnicorn's core and executors (❼), we measure the wall time or elapsed time as a proxy for CPU cycles and use a Python-based memory profiler [160], respectively. Our results show that the executor running on a low-end node such as a Raspberry Pi incurs a computing overhead of approximately **1 second per stage** and **0.13 seconds per task** while consuming less than **21 MB of memory**. Meanwhile, netUnicorn's core incurs a computing overhead of around five seconds for deployment and 20 seconds for execution in a 20-node infrastructure while consuming less than **417 MB of memory**. These experiments are described in more detail in [29], Appendix F.

5.7 Discussion

More learning problems. While not implemented in this paper, we envision that the netUnicorn platform can be used for a wide range of different network security problems,

such as network censorship [12, 32, 113], website fingerprinting [56, 223], Tor traffic analysis [73], and others. Many of these problems involve an active measurement component for data collection, labeling, or communication and would benefit from netUnicorn-provided capabilities such as (i) running experiments that require the simultaneous use of different infrastructures and (ii) facilitating the reproducibility and shareability of experiments. To demonstrate this benefit, we used netUnicorn to implement a multi-vantage point validation of the Let's Encrypt ACME challenge [33]. We provide additional evidence for the practicability and versatility of netUnicorn and its use as part of our newly-proposed ML pipeline by describing in the extended version of this paper [29] (Appendices A and B) the application of our approach to the Let's Encrypt example and two additional real-world security problems, namely Heartbleed detection and OS fingerprinting.

Usability and Realism. First, a critical step in our proposed method is that we require domain experts to articulate data collection intents. As demonstrated in Section 5.5, it is often possible to generate appropriate intents with the help of explainable ML models. Our platform design further simplifies the process of translating intents into action, ensuring the usability of our proposed method. Second, our data collection follows an emulation-based mechanism that enables accurate labeling. With our proposed iterative approach, we can eliminate biases from the collected data. Additionally, our platform significantly lowers the threshold for gathering data from multiple environments, enhancing the diversity of the data collected. As demonstrated in Section 5.5, the data we collected is realistic and representative and can improve the generalizability of trained models in various environments.

Limitations of the proposed approach.

Active data collection. Our approach uses endogenously generated (labeled) network data from actual network environments. We note that it may also be possible to improve a model's generalizability by means of carefully selected and exogenously generated (passive)

data from a production network, but such an approach is beyond the scope of this paper.

Feature pre-processing. Curating training datasets entails both data collection and pre-processing. Since data pre-processing remains the same for different versions of the collected data that result from our iterative approach, it poses no problems for the desired "thin waist" of netUnicorn's design. In this paper, we utilized the CICFlowmeter for pre-processing, which worked well for all considered learning problems. While we readily acknowledge that there is more to data pre-processing than CICFlowmeter, we leave the exploration of alternative pre-processing (as well as model selection and optimization) techniques for future work.

Decomposing pipelines. We assume that it is possible to decompose a data-collection pipeline into self-contained tasks. However, such a decomposition may be cumbersome for complex learning problems like Puffer [254] that require closer service integration.

Decoupling pipelines from infrastructures. We assume that it is possible to decouple the data-collection intents from actual infrastructure-specific mechanisms. However, realizing this may be difficult, especially for experiments where the data-collection tasks are heavily intertwined with a specific attribute of the data-collection node. For example, some IoT security experiments [235] require running the data-collection pipeline on specific devices with integrated firmware and pre-defined implementations of closed-source services, which cannot be easily supported by netUnicorn.

Programming overheads. Our approach requires experimenters to express new data-collection tasks that are not yet presented in netUnicorn's library. Though this effort will amortize over time, it will only materialize if we succeed in building and incentivizing a broad user community for the proposed platform. Here, we take a first step and make a case for a holistic communal effort to address the data quality and model generalizability issues that have impeded the use of ML-based network security solutions in practice to date.

Limitations of the prototype implementation.

Data-collection nodes. Our current prototype only supports Linux- or Windows-based nodes, optionally with Docker support to enable full platform capabilities (such as Docker container environments). This restriction is reasonable because of the widespread support for Docker-based containers in current data-collection infrastructures [49,82] and a growing trend to manage Docker-based infrastructures [22,135]. In future work, we plan to extend support to other computing environments, such as OpenWRT routers and PISA switches, which do not natively support Python or Docker. Currently, such extensions are possible using the sidecar model [221], which allows the configuration of nodes without Python support through Python-based APIs, such as P4-runtime [182].

Potential subjectivity and biases. Applying our proposed closed-loop ML pipeline involves the use of domain experts who themselves can be a source of possible biases or can make subjective decisions. One immediate solution to address this problem is to rely on multiple experts for cross-validation of explanations and decisions regarding data collection. For a more long-term solution, we envision the development of quantitative methods (e.g., metrics for evaluating explanation fidelity [106]) that will facilitate the detection of possible shortcuts or other types of inductive biases.

As far as other bias-related issues are concerned, we are already using a validation set for parameter selection to reduce parameter bias, and our method naturally helps avoid data snooping because it supports collecting data for different tasks and from different network environments at different times and allows for periodically examining and (if necessary) updating trained models.

Manual effort. A concerning side effect of using domain experts as part of our closed-loop ML pipeline is the manual effort it entails. While this makes the current version of our new pipeline inherently semi-automatic, future development of quantitative methods for detecting and possibly eliminating different types of inductive biases promises to reduce

the manual effort required and make the pipeline more automatic. The development of such methods could potentially also benefit from advances in how AI can be utilized for examining model explanations and making model modification suggestions, but such issues are beyond the scope of this paper.

5.8 Related Work

Alternative approaches for our designs. In principle, it is possible to use existing tools and frameworks to realize the "thin waist" we implemented for data collection, but doing so while achieving netUnicorn's level of abstraction, extensibility, fidelity, and scalability poses significant challenges (see [29], Appendix H and Appendix G, for details). For example, one possibility is to disaggregate pipelines into tasks with existing *workflow-management platforms*, such as Airflow [5] or others [63, 152, 167]. However, there is often no explicit support to map these pipelines to specific data-collection nodes and instantiate multiple copies of tasks – limiting data-collection experiments' flexibility. Existing *CI/CD systems* (e.g., Jenkins [127] and others [98, 99] allow explicit mapping of pipelines to nodes but typically require specific infrastructure access and configuration, limiting the desired extensibility and fidelity. Besides, they do not optimize inter-task execution time, limiting their ability to scale the data collection scenarios. Finally, one can also use different *configuration* (e.g., SaltStack [208]) or *orchestration platforms* (e.g., Kubernetes [135]), and others [13, 53, 192, 239]. However, these systems lack the desired extensibility and flexibility because, being tailor-made for orchestration, they only work for specific types of infrastructures and do not provide explicit support for the proposed pipelines and stages abstraction, limiting tasks and experiments' reusability.

Passive data augmentation. In computer vision, researchers synthesize novel training data by adding random Gaussian noise to training images [219, 236] or blurring, rotating,

and flipping them. However, these methods are specific to images and can only rarely be applied beyond vision data. Recent studies propose more application-domain independent methods, such as mixup [260] and SMOTE [52, 134], which can be applied to networking data. However, as demonstrated in Section 5.5, these methods have limited efficacy in networking applications due to the correctness of the augmented data. They also generate samples that are typically very similar to the given training data, thus limiting the examination of model generalizability. Another line of data augmentation methods generates adversarial samples by adding carefully crafted perturbations to training samples (e.g., [55, 102, 196]). Since these perturbations are just noises with a Non-Gaussian distribution, they suffer from similar limitations as adding Gaussian noise.

Model-side efforts. Various model-side efforts have also been considered to improve model generalizability. In particular, (reinforcement learning-based) domain adaptation methods (e.g., [86, 212]) maintain an ML model's efficacy across multiple domains. To generalize across different learning problems, existing research proposed multi-task learning [205, 262]) and few-shot learning methods [101, 202]. Researchers have also developed advanced models to combat shortcuts [93] or out-of-distribution (ood) issues [117], such as detecting oods with contrastive learning [257]. All the model-side efforts assume that the training data is fixed and already given. These techniques are orthogonal and complementary to our method, which focuses on improving datasets.

5.9 Conclusion

In this paper, we present a novel closed-loop ML pipeline to curate high-quality datasets for developing generalizable ML-based solutions for network security problems. Our approach is based on a new data-collection method that leverages advances in explainable ML and emphasizes the need for a flexible “in vivo" collection of training

datasets. It takes inspiration from the classic “hourglass” abstraction, where the different learning problems make up the hourglass’ top layer, and the different network environments constitute its bottom layer. We realize the “thin waist" of this hourglass abstraction with a new data-collection platform, netUnicorn. In effect, for each learning problem, netUnicorn enables data collection in multiple network environments, and for each network environment, it facilitates data collection for multiple learning problems. Through extensive experiments that involve different network security problems and consider multiple network infrastructures, we demonstrate how netUnicorn, in conjunction with the use of explainable ML tools, simplifies data collection for different learning problems from diverse network environments, enables iterative data collection for advancing the development of generalizable ML models, and improves the reproducibility, reusability, and shareability of network security experiments.

Part III

Towards Generalizable Models

This part presents the dissertation's model-side response to the credibility problems diagnosed earlier. Rather than relying on narrow supervised datasets and task-specific shortcuts, the works in this part explore how foundation models can improve generalizability when they are grounded in the realities of network data and deployment. Together, they show two complementary paths: building a network-native foundation model whose architecture is explicitly informed by prior failure analysis, and adapting general-purpose text-based LLMs with networking-specific context so they can produce useful and trustworthy interpretations.

Chapter 6 introduces *netFound*, a network foundation model that translates the lessons from Chapter 2 and Chapter 3 into concrete design choices, including protocol-aware tokenization, operational context embeddings, hierarchical modeling of burst and flow structure, and privacy-conscious input design. Trained on large-scale real-world traffic enabled by the infrastructure developed in Part II, *netFound* serves as the dissertation's main model contribution and demonstrates that foundation models can learn more reusable traffic representations and generalize better across downstream networking tasks. Chapter 7 complements this direction by exploring how pretrained LLMs can be grounded with geolocation and retrieval over historical measurements to explain speed-test results for non-expert users.

In the overall dissertation, these chapters complete the arc from diagnosing failures and building better data-collection infrastructure to designing models that are more deployable, more interpretable in context, and more generalizable.

Chapter 6

netFound: From Diagnostics-Informed Design to Production-Ready Network Foundation Model

6.1 Introduction

Deployable network intelligence depends on trustworthy traffic representations. Self-driving networks, capable of monitoring, diagnosing, and optimizing themselves with minimal human intervention, have been a long-standing aspiration of the networking community [87, 163, 172]. Recent advances in large language models have brought this vision closer: LLMs can parse network configurations, generate management code [157], and reason about network intent [251]. However, application of text-based LLMs to raw traffic data is limited and network operators increasingly need models that can turn packet traces into reusable representations for tasks such as traffic classification, anomaly detection, diagnosis, or even agentic-based decision making. With over 95% of web traffic encrypted under TLS [6] and due to privacy considerations, in production these

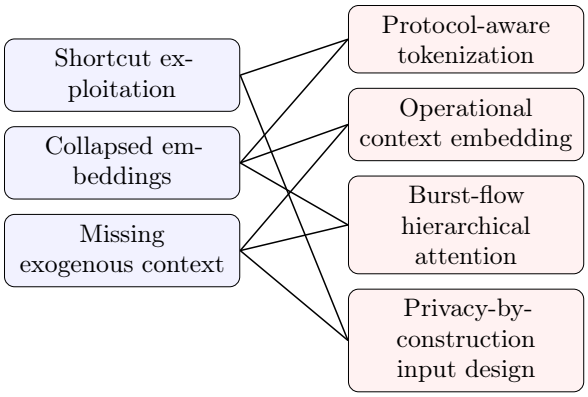


Figure 6.1: Design principles (right) motivated by diagnostic findings (left).

representations must be learned from packet headers, timing, and flow behavior rather than payload, which limits the information available for analysis. All these requirements call for more trustworthy, reusable traffic representations that can be learned at scale and deployed in real-world workflows beyond pure benchmark performance.

Existing network foundation models fail to yield reusable traffic representations. Network foundation models (NFMs) promise reusable traffic encoders through large-scale self-supervised pretraining [112, 144, 242, 264, 266]. Yet recent diagnostic works show that many current NFMs function at best as benchmark solvers and yield at worst non-reusable representations. Trustee [126] showed that network ML systems often exploit dataset shortcuts rather than learning genuine traffic behavior. Pcap-Encoder [265] demonstrated that NFM results can be inflated by leakage pathways such as per-packet train/test splits and timestamp correlations. The Intrinsic Evaluation Framework (IEF) [30] further showed that existing NFMs often produce anisotropic (collapsed) embeddings, align weakly with domain-expert traffic features, and fail to capture exogenous network conditions that shape observed behavior. Together, these results suggest that current NFMs are not yet a sufficient foundation for privacy-preserving, production-facing network understanding.

The missing piece is a network-native, diagnostics-informed design. These failures are not incidental. Most existing NFMs inherit tokenization, embedding, and

attention mechanisms from natural language processing or vision, even though network traffic has protocol-defined field structure, rich operational metadata, a natural burst-flow hierarchy, and strict privacy constraints. Our thesis is that a deployable network foundation model must be designed around these properties from the start. It should preserve protocol semantics during tokenization, encode metadata alongside header content, model behavior at both burst and flow scales, and restrict identity-bearing inputs by construction. Section 6.2.3 formalizes these requirements as four design principles that guide the rest of the paper.

netFound is our answer: a production-ready, privacy-preserving network foundation encoder. We present *netFound*, an encoder-only, network-native foundation model built around this design thesis. netFound applies protocol-aware tokenization, injects operational context into token representations, uses burst-flow hierarchical attention to capture multi-scale traffic structure, and operates on packet headers without payload or endpoint IP identities. Beyond the model architecture, netFound is built as a practical system: we pretrain three model sizes on a billion-token-scale corpus, release full code, pretraining dataset, weights, and provide a containerized end-to-end pipeline from raw PCAPs to downstream inference so that the model can be reproduced, extended, and deployed rather than treated as a one-off artifact.

This design yields stronger reusable representations in both intrinsic and downstream evaluation. Using the same diagnostic lenses that revealed the limitations of prior work, we show that netFound produces substantially better representations. netFound reaches an F_1 of 0.95 on exogenous context discrimination, compared to 0.62 for the best prior state-of-the-art baseline, and it is the strongest model by a wide margin in frozen-encoder evaluation, demonstrating that the pretrained embeddings themselves carry useful structure before task-specific adaptation. At the same time, on downstream benchmarks, netFound remains top-performing across five tasks in both frozen and end-

to-end fine-tuned settings. These results show that privacy-by-construction input design, representation quality, and production-readiness can be improved together rather than traded off against one another.

Contributions.

1. **From diagnosis to deployable design.** We show how recent diagnostic findings on shortcut exploitation, weak embedding structure, and poor context sensitivity [30, 126, 265] translate into concrete requirements for a deployable network foundation model, and we formalize these requirements as four design principles (§6.2.3).

2. **A production-ready, privacy-preserving network foundation encoder.** We build netFound, a network-native encoder that implements these principles through protocol-aware tokenization, operational context embedding, burst-flow hierarchical attention, and privacy-by-construction input selection, and we pair it with an end-to-end reproducible system for preprocessing, pretraining, fine-tuning, and deployment (§6.3, §6.4). We fully release open-source implementation under permissive license, including pretrained weights for three different model sizes (53M–663M parameters) on HuggingFace, a containerized end-to-end pipeline, native C++ acceleration for preprocessing, and an automated test suite (§6.4), to enable the community to reproduce, extend, and deploy netFound rather than treating it as a one-off artifact. We also publish the *full pretraining dataset*, containing *4.2 billion* anonymized network flows in the Apache Arrow format, suitable for netFound pretraining for full reproducibility of the results.

3. **Evidence of stronger reusable representations.** We demonstrate, through intrinsic evaluation and downstream benchmarks, that netFound yields higher-quality reusable traffic representations than prior NFMs. These representations

especially shine in the better discrimination of exogenous network context compared to six existing models (including Pcap-Encoder) and in frozen-encoder downstream settings, while remaining top-performing after full end-to-end fine-tuning across five downstream tasks (§6.5.1, §6.5.2).

6.2 Background and Problem Scope

This section reviews representation learning in networking and identifies the problems in current network foundation model designs that motivate our approach.

6.2.1 Representation Learning in Networking

The core premise of representation learning is that raw data (packet captures) can be transformed into compact vector embeddings that capture the underlying structure relevant to downstream tasks. A good network representation should satisfy four requirements derived from the failure modes identified by recent diagnostic work [30, 265]: (1) *semantic preservation*, where protocol field boundaries and their meanings survive tokenization; (2) *operational grounding*, where the representation encodes the operational metadata (timing, burst statistics, directionality) that domain experts rely on; (3) *contextual depth*, where the representation captures not only what is in the packet headers but also the exogenous conditions, such as congestion control, queue management, or cross-traffic, that shape traffic behavior; and (4) *leakage resilience*, where the representation avoids dependence on fields that leak identity or split-specific information, such as IP addresses, checksums, payload bytes, or other fields that a model could memorize to artificially inflate generalization.

Existing approaches to network representation learning fall into two categories. Task-specific methods extract hand-crafted features and train a supervised model per task [3,

166]; they can be effective but do not generalize across tasks and require labeled data. Foundation models instead pre-train on large unlabeled corpora and produce reusable representations that can be adapted to many downstream tasks via fine-tuning. The promise is clear, but as we show next, both the models and their representations are seriously flawed.

6.2.2 Diagnosis: Why Existing Models Fall Short

Inspired by the success of foundation models in other domains, multiple network foundation models have been developed in recent years [112,114,144,194,241,256,264,265]. These solutions vary in tokenization, embedding, and pre-training strategies, but share a common root cause: most treat network data as natural language or images and employ transformer architectures developed for these domains. ET-BERT treats network data as natural language using BERT with domain-specific pre-training tasks. YaTC [264] and Flow-MAE [112] treat data as images using Vision Transformer [76] and Masked Autoencoders [115]. Pcap-Encoder [265] adopts a T5-based architecture with a corrected per-flow evaluation protocol, but uses simple packet hexadecimal representation. Table 6.1 summarizes how each model handles the four design dimensions. Existing models default to generic NLP/vision choices instead of making domain-specific decisions for each. Two

Table 6.1: Design choices across network foundation models. Existing models import NLP/vision defaults; netFound makes domain-specific choices for each dimension.

Model	**Tokenization**	**Metadata**	**Attention**	**Input**
ET-BERT [144]	2B BPE (payload)	—	flat	payload bytes
PERT [114]	2B chunks (payload)	—	flat	payload bytes
LENS [241]	fixed-byte (payload)	—	flat	payload bytes
TrafficGPT [194]	byte-level (payload)	—	flat	payload bytes
YaTC [264]	fixed-byte (hdr+pay)	—	pkt→flow	hdr+payload
Flow-MAE [112]	fixed-byte (hdr+pay)	—	pkt→flow	hdr+payload
Pcap-Encoder [265]	fixed-byte (hdr+pay)	—	flat	hdr+payload
netFound	**protocol-field**	**IAT, burst, dir.**	**burst→flow**	**anon. headers**

recent diagnostic studies reveal the consequences of these design gaps.

Intrinsic representation failures (IEF). The Intrinsic Evaluation Framework [30] introduced a task-agnostic methodology for probing NFM representations through three lenses. *Embedding geometry*: mean cosine similarity between flow embeddings reveals that existing models produce highly anisotropic representations, meaning diverse flows are mapped to nearly identical vectors, making it impossible for consequent layers to separate. *Metric alignment*: The centered kernel alignment (CKA) index that measures similarity between model embeddings and CICFlowMeter domain-expert features [140] is near zero, indicating that representations fail to encode the operational statistics that network engineers use. *Exogenous context discrimination*: linear probes trained on flow embeddings achieve low $F_1 < 0.62$ for discriminating congestion control algorithms, queue management policies, and cross-traffic patterns. These conditions significantly shape traffic behavior but are usually invisible in packet headers, making it a challenging task for ML models to work with.

Evaluation pitfalls and shortcut exploitation (Pcap-Encoder). Pcap-Encoder [265] and Trustee [126] demonstrated that models trained on network datasets frequently rely on shortcuts and spurious correlations rather than learning meaningful traffic patterns. Among common examples, per-packet splitting allows models to use TCP sequence numbers as flow identifiers, and timestamp correlations in simultaneously-collected datasets often leak application identity. To demonstrate the resulting problems in generalizability,

Table 6.2: F_1 score of models on original versions of datasets (with shortcuts) and fixed (without shortcuts). Performance drop signals vulnerability to shortcuts.

	Crossmarket (*Acc*@10)		**CIC-IDS (Heartbleed)**	
	Original	Fixed	Original	Fixed
ET-BERT	99.82 ± 0.03	32.83 ± 0.17	99.99 ± 0.01	0.0 ± 0.0
YaTC	99.69 ± 0.03	63.48 ± 0.12	99.99 ± 0.01	0.01 ± 0.01
Pcap-Encoder	97.18 ± 0.21	86.99 ± 0.32	99.99 ± 0.00	0.0 ± 0.0
netFound-large	65.38 ± 0.81	65.38 ± 0.81	99.98 ± 0.01	99.98 ± 0.01

we find and fix shortcuts for two commonly used downstream datasets: Crossmarket [237] and CIC-IDS2017 [214], by removing the TCP Options field that contains class-unique timestamp in Crossmarkets and fixing the Heartbleed shortcut by splitting the flow between attacks as proposed by Trustee [126] for CIC-IDS2017. As shown in Table 6.2, existing models achieve near-perfect performance on original datasets but performance drops dramatically on fixed versions, confirming heavy reliance on spurious features. Pcap-Encoder [265] recognized this problem and proposed several corrections in the evaluation protocol, being able to resolve some of the well-known shortcuts, such as Crossmarket, but not all of them. However, while Pcap-Encoder addresses the evaluation methodology, it retains a protocol-agnostic T5-based architecture: it does not use protocol-aware tokenization, operational metadata, or network-suitable attention designs. This means Pcap-Encoder corrects *how* models are evaluated but does not address *why* they produce poor representations in the first place.

6.2.3 From Diagnosis to Design Principles

The diagnostic findings from preliminary works form a coherent picture of what network foundation models must do differently. We synthesize these findings into four design principles that guide our overall design and are illustrated in Figure 6.1. Crucially, the mapping is many-to-many: each principle addresses multiple diagnostic findings, and each finding motivates multiple principles. In effect, this interconnected structure accentuates why netFound's design is principled rather than ad-hoc.

Principle 1: Tokenization must respect protocol structure. IEF's finding that embeddings have near-zero CKA alignment with domain-expert features, combined with Pcap-Encoder's finding that fixed-sized byte chunks strategy corrupts protocol field semantics, jointly motivate a tokenization strategy that preserves field boundaries.

CICFlowMeter features are computed from specific protocol fields (flags, IP TTL, etc.); if tokenization destroys these fields, the model can never learn features that align with them. Furthermore, protocol-aware tokenization enables selective exclusion of fields known to leak identity, providing a structural defense against shortcut exploitation. This motivates *protocol-aware tokenization* (§6.3.1).

Principle 2: Representations must encode operational metadata. IEF showed that existing model embeddings have near-zero alignment with the operational features that domain experts rely on (CKA < 0.07), and that models cannot discriminate exogenous network conditions ($F_1 < 0.62$). Both failures share a root cause: models ingest only raw bytes and have no explicit representation of the timing, burst statistics, and directional metadata through which exogenous conditions manifest. IEF demonstrates that models struggle with these features the most, resulting in overall reduced quality of the embeddings. Injecting this metadata also provides additional axes of variation that can help spread representations in the latent space, mitigating the anisotropy problem. This motivates *operational context embedding* (§6.3.2).

Principle 3: Attention must capture multi-scale temporal structure. IEF's context discrimination results reveal that existing models cannot capture exogenous network conditions that manifest at different temporal scales: congestion control affects burst-level inter-arrival times (milliseconds), while cross-traffic patterns shape flow-level behavioral trends (seconds). Ignoring these temporal patterns leads to poor representation capabilities, and addition of a simple flat attention over truncated sequences cannot model both scales simultaneously. YaTC and Flow-MAE attempt hierarchy but without parameter sharing across granularities, limiting cross-level reasoning. This motivates *burst-flow hierarchical attention* with shared parameters and CLS-token propagation (§6.3.3).

Principle 4: Input design must prevent shortcut exploitation and privacy

issues by construction. Pcap-Encoder and Trustee showed that existing models exploit TCP sequence numbers, timestamp correlations, and IP-based identifiers as shortcuts, achieving near-perfect accuracy that collapses under proper evaluation (Table 6.2). In addition, to protect user privacy, we require that a production-ready model is unable to memorize any potentially sensitive information that can be gleaned from either payload data or certain header fields. Rather than relying on post-hoc regularization, we adopt a prevention-by-design approach: if the model never sees the leaky features or sensitive information, it cannot exploit them. This motivates *privacy-by-construction input design* (§6.3.4).

6.3 netFound Architecture

The four principles derived in Section 6.2.3 are not independent and form an interconnected set that constrains the design space but does not uniquely determine an architecture. In this section we describe how each principle is realized in netFound, and for each design choice we discuss the alternatives we considered and the reasoning that guided our decisions.

6.3.1 Protocol-Aware Tokenization

Design space. Network foundation models (as shown in Table 6.1) can utilize one of three different strategies for tokenization of raw network data. (1) *Fixed-size byte chunks* strategy (which splits a raw byte stream into 2-byte tokens, as in PERT [114] and Pcap-Encoder [265]) is simple but protocol-agnostic; e.g., a 2-byte window at the wrong offset splits a 16-bit TCP window-size value across two tokens, forcing the model to learn cross-token reconstruction before it can reason about window behavior. (2) *Learned subword tokenization* (e.g., Byte-Pair Encoding, or BPE, creates a translation table for

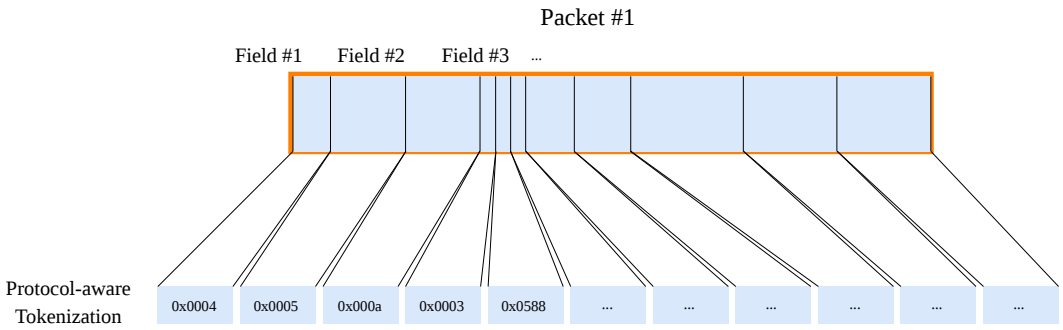


Figure 6.2: Protocol-aware tokenization preserves packet field semantics by splitting headers along protocol field boundaries.

the most popular byte chunks) works by discovering frequently co-occurring byte patterns, but these patterns reflect byte-frequency statistics rather than protocol semantics; BPE cannot guarantee that a token boundary will not bisect a protocol field, and it provides no mechanism for selectively excluding known leaky fields (e.g., IP address) because field identity is not represented in its vocabulary. (3) *Protocol-aware tokenization* aligns tokens with protocol field boundaries, ensuring that each token corresponds to exactly one semantically meaningful field. This approach produces correct field splits and associations but requires stricter packet parsing and is computationally more expensive.

Preserving field semantics. Diagnostic studies [265] have shown that protocol-agnostic fixed-size tokenization is a root cause of semantic corruption in existing network foundation models: token boundaries routinely split protocol fields, destroying the very information the model needs to learn. Because packet headers are structured according to protocols such as TCP, UDP, and IP, we employ strategy (3), *protocol-aware tokenization* strategy, that segments headers based on their protocol-specific fields, as shown in Figure 6.2. Each token corresponds to exactly one semantically meaningful field (e.g., source port, TTL, TCP flags), allowing variable byte lengths without complicating embedding or model training. This directly addresses Principle 1 from Section 6.2.3: protocol-aware

tokenization preserves the semantic units that domain-expert features are computed from, and enables selective exclusion of known leaky fields.

6.3.2 Operational Context Embedding

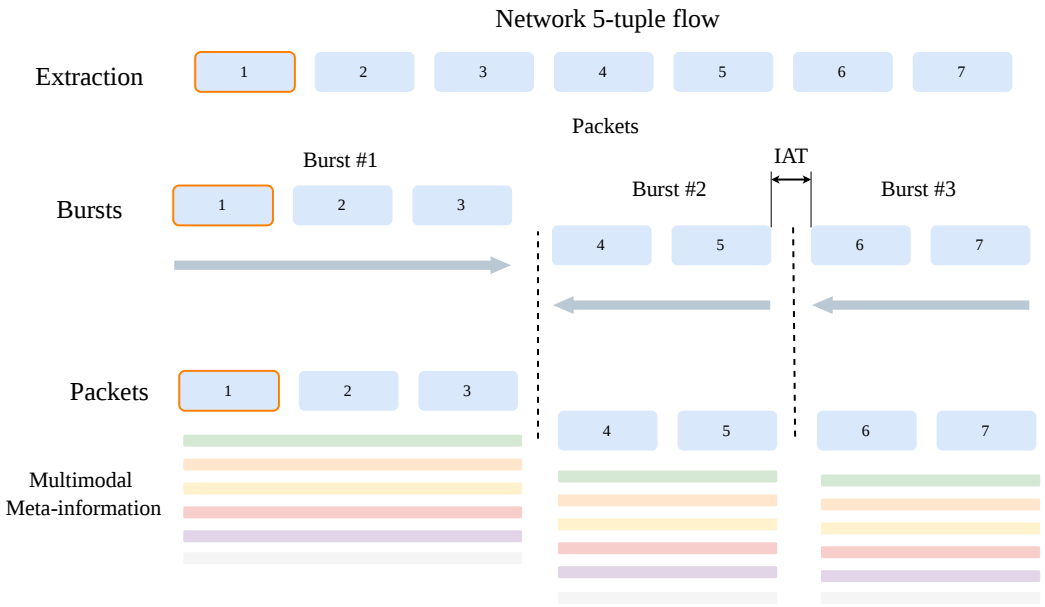


Figure 6.3: Network-aware data extraction and featurization enriches the model's embeddings with operational metadata.

Design space. Existing NFMs operate on raw bytes alone, discarding the operational metadata that domain experts rely on and reducing the model's quality. To include this operational metadata, there are several possible approaches. (1) *Feature engineering as input* replaces raw bytes with hand-crafted features. By definition, this method achieves high alignment with domain expert-derived features, but the generality of the learned representations suffers: the model can only encode features that were anticipated at design time, and it loses access to raw header content that may serve as a useful but unanticipated signal. (2) *Additive embedding* projects metadata into the same space as

token embeddings and sums them. While simple, addition is a lossy operation: when the metadata and token embedding vectors occupy overlapping subspaces, the sum conflates the two sources of information and downstream layers cannot recover which contribution came from which modality. (3) *Concatenation and compression* preserves both raw bytes and operational data by concatenating them into a wider vector and then projecting back to the model dimension via a learned linear layer. The projection can learn to allocate capacity between modalities, and no information is destroyed before the model has a chance to see both sources. We adopt approach (3) because it addresses Principle 2 (grounding representations in operational metadata) while preserving the raw-byte signal that enables the model to discover unexpected features.

Grounding representations in domain-expert features. Intrinsic evaluation of existing models reveals that their embeddings have near-zero alignment (measured via CKA) with the operational features (timing, burst statistics, directionality) that domain experts rely on for traffic analysis [30]. Approach (3) allows us to close this alignment gap and encode operational context directly into the token embedding. As shown in Figure 6.3, we calculate and embed inter-arrival time information, burst size (number of packets, total number of bytes), direction of the packets in the flow, and relative positions of the packets in the burst and flow. These features are projected through a two-layer MLP with activation into the hidden dimension, then *concatenated* with the word embedding rather than added to it. We additionally embed the transport-layer protocol (e.g., TCP, UDP, ICMP) via a dedicated learned embedding table, broadcast to every token position. A linear projection layer then fuses the three concatenated components (word, metadata, protocol embeddings) back to the model's hidden dimension. This design is similar to the *unified embedding* architecture used in modern multi-modal encoders [94].

6.3.3 Burst-Flow Hierarchical Attention

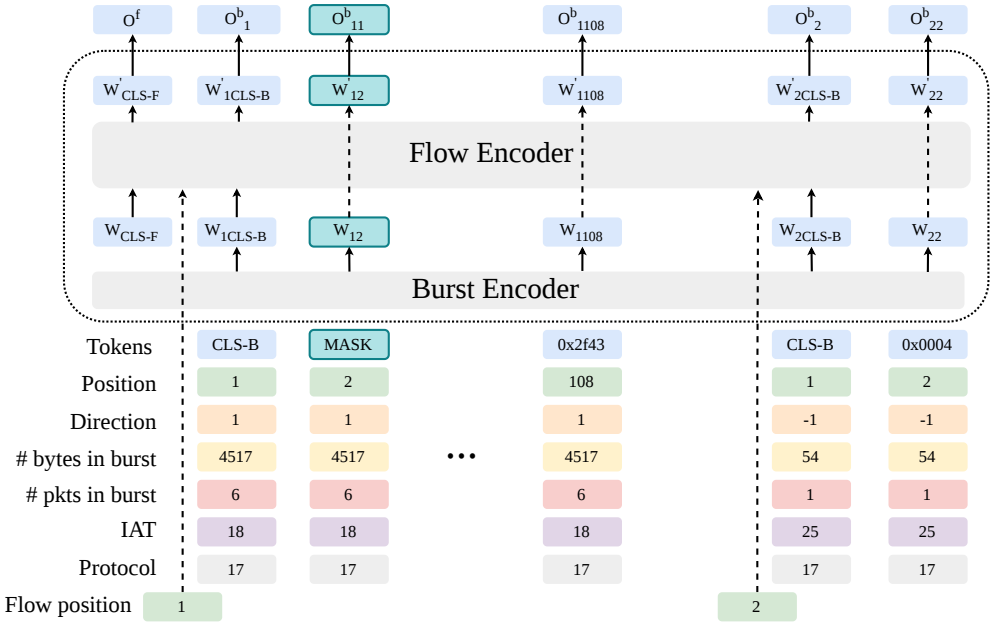


Figure 6.4: The hierarchical transformer architecture decomposes attention into burst-level and flow-level stages.

Design space. Attention mechanisms give models an opportunity to account for nearby tokens at every step in the calculations. Recent works [27, 67] proposed various attention patterns, but the patterns that are suitable for network data fall into four categories: (1) *Flat attention* over the entire token sequence (used by ET-BERT, PcapEncoder, etc.) takes all tokens regardless of their position in the burst-flow hierarchy. This attention pattern incurs quadratic cost in sequence length and, more importantly, forces the model to learn the burst-flow structure implicitly from positional embeddings alone. (2) *Windowed/sparse attention* (e.g., Longformer [27]) reduces cost by restricting each token's attention to a fixed window, but the window size remains a hyperparameter and ignores the natural burst boundaries in traffic. A window that is too small misses cross-packet dependencies within a burst; one that is too large conflates adjacent bursts.

(3) *Hierarchical attention* with burst-level and flow-level stages aligns the attention structure with the data's natural hierarchy: burst encoders capture within-burst packet interactions, flow encoders capture cross-burst behavioral patterns. (4) *Using disaggregated models* is an attention mechanism where separate foundation models are trained at the packet, burst, and flow levels and combined at inference. It avoids the architectural complexity of hierarchical attention but prevents cross-level parameter sharing and requires coordinating multiple models at inference time. We adopt approach (3) shown on Figure 6.4 because this architecture explicitly models both temporal scales within a single model while enabling cross-level information flow through CLS-token propagation to satisfy Principle 3.

Capturing multi-scale dependencies. Existing models either flatten entire flows into a single token sequence, losing the natural grouping of packets into bursts, or attempt hierarchy without parameter sharing across granularities [112, 264]. Intrinsic evaluation shows that this prevents models from capturing network conditions that manifest at different time scales [30]. Network data exhibits a natural two-level hierarchy: tokens within a burst capture fine-grained packet-field interactions, while bursts within a flow capture coarser temporal and behavioral patterns. To reflect this behavior, each netFound transformer layer is composed of two separate sub-layers, a *burst encoder* and a *flow encoder*, that operate at different granularities.

Given an input sequence of token embeddings, we first reshape the flat token sequence into a burst-structured tensor, grouping tokens by their burst membership. The burst encoder then applies self-attention independently within each burst, allowing tokens to attend only to other tokens in the same burst. This step captures local dependencies such as relationships among packet header fields and inter-packet patterns within a burst. Each burst is prepended with a CLS token whose output representation serves as a holistic summary of that burst.

After the burst encoder, we extract the obtained burst-level CLS representations and feed them into the flow encoder. By applying self-attention across all bursts in a flow, this encoder enables the model to capture cross-burst dependencies such as long-range temporal patterns, flow-level behavioral shifts, and correlations between distant parts of a conversation. Crucially, individual token representations *skip* the flow encoder via a residual connection and only the CLS tokens are updated by the flow-level attention. The updated CLS representations are then written back into the full sequence, enriching each burst's summary with flow-level context while preserving the fine-grained token representations. This hierarchical decomposition offers both computational and modeling advantages. By restricting burst-level attention to tokens within each burst (typically up to 108 tokens) rather than the entire flow (which can exceed 1,300 tokens), the quadratic cost of self-attention is substantially reduced. The flow encoder operates only over the burst CLS tokens (i.e., 12), keeping its cost minimal.

6.3.4 Privacy-by-Construction Input Design

Design space. There exist three popular general input design strategies that define what information a model sees during training and inference. (1) *Full packet ingestion* (ET-BERT, YaTC, Pcap-Encoder) feeds the model raw bytes including payload, IP addresses, and all header fields. This strategy maximizes the information available to the model but also maximizes the attack surface for shortcut exploitation: e.g., payload bytes often contain application-level identifiers and IP addresses encode network topology. (2) *Post-hoc regularization* retains all features but adds augmentations during training to discourage shortcut use. However, this method is fragile: the model may find new shortcuts not anticipated by the regularizer, and there is no formal guarantee that shortcuts are eliminated. (3) *Prevention by design* restricts the input space to features with known

behavioral semantics, excluding those that could plausibly leak identity. Doing so sacrifices some information but provides a structural guarantee: the model never sees leaky features, so it cannot exploit them.

Most papers in the existing literature utilize approach (1) which allows them to demonstrate the highest scores on the fine-tuning datasets, either by shortcut exploitation or memorization of data patterns. However, network data is also known to contain information that can be considered *sensitive* from the users' perspective, such as IP addresses, payload (regardless of encryption), and other info (e.g., domain names in SNI). To protect such information and avoid failures under proper evaluation (as Pcap-Encoder results demonstrated for approaches (1) and (2)), we adopt approach (3) that avoids the use of sensitive or encrypted fields. Prevention by design is the only strategy that addresses Principle 4 with a provable guarantee rather than a statistical one.

Preventing shortcut exploitation. Concretely, netFound operates on packet headers without payload, uses directional information instead of endpoint IP addresses, and applies protocol-aware field selection that can exclude known leaky fields. A modular design allows researchers to include or exclude additional fields (e.g., by using a model's different configurations), enabling them to tailor privacy guarantees to their unique needs.

Dynamic padding and bucketing. Network flows exhibit high variability in length, both in the number of packets per burst and the number of bursts per flow, posing a challenge for fixed-sized architectures. We address this through a three-pronged strategy of *truncation*, *dynamic padding*, and *length-bucketed batching*. First, during tokenization, each burst is truncated to a configurable maximum burst length (i.e., 109 tokens including a CLS token, encoding 6 packets within a single burst), and each flow is truncated to a maximum number of bursts (i.e., 12), bounding the input dimensionality, but still reliably representing the overall flow structure. Second, rather than padding all sequences to global maximums, our data collator computes per-batch maximums, padding only to

those values, avoiding saturating models with empty sequences. Third, we employ a length-bucketed sampler that groups similarly-sized flows into batches ensuring efficient batch packing. Together, these mechanisms allow netFound to efficiently handle the wide range of sequence lengths inherent in network traffic.

6.3.5 Modern Transformer Components

Following recent advances in efficient transformer design from ModernBERT [245], we incorporate several architectural improvements into each transformer layer. We replace learned absolute position embeddings with Rotary Position Embeddings (RoPE) [228], adopt Flash Attention [67] and BF16 for accelerated training, and use the ModernBERT MLP design with GeGLU activation [216] and a reduced intermediate size (1.5-2.5$\times$ the hidden dimension). We also remove bias terms from both the MLP and LayerNorm layers, and apply pre-normalization before the MLP with a residual connection. Our tokenizer's vocabulary comprises 65,600 entries, of which 65,539 correspond to byte-level token values and model-specific tokens, and the remainder is reserved to allow researchers to use them for specific downstream applications.

6.3.6 Pre-Training Objectives

A pretrained encoder must learn representations that are useful across diverse downstream tasks without task-specific supervision. We use a multitask pretraining objective with four complementary losses. Each loss is detailed below and targets a different aspect of traffic structure. The rationale for multiple objectives is that no single self-supervised loss covers all the representational requirements from Section 6.2.3: while the first loss teaches token-level semantics, the remaining three losses teach burst-level and flow-level structure that the first loss alone cannot capture.

Masked Language Modeling (MLM). Our primary objective follows the masked language modeling paradigm [70], adapted for packet header tokens. During training, 30% of tokens are selected for masking: 80% replaced with `[MASK]`, 10% with a random token, 10% unchanged. The model is trained to reconstruct original tokens using cross-entropy loss.

Swapped Burst Detection. To encourage flow-level coherence, we introduce a *swapped burst detection* objective. Here, randomly selected bursts are replaced with ones from a different flow. A binary classifier on the pooled flow-level representation detects the swap, helping the encoder to learn the burst correlation information.

Metadata Prediction. We train the model to predict three continuous burst-level metadata features: inter-arrival time, total bytes, and packet count. For each burst whose metadata is masked (probability 0.3), the model regresses these values from the burst's CLS representation using L1 loss, ensuring that CLS representation learns useful information from flows.

Direction Prediction. A direction classifier predicts burst direction (inbound vs. outbound) on burst CLS representations, further helping the model with flow understanding.

Loss Weighting. Each loss is independently weighted, and the total pretraining loss is their weighted sum. This multi-task formulation ensures that netFound simultaneously learns fine-grained token-level semantics, burst-level statistical properties, and flow-level coherence.

6.4 Implementation

An architecture that satisfies the design principles specified in Section 6.2.3 is necessary but not sufficient for a practical network foundation model. From a practical perspective, the engineering decisions matter as much as the architectural choices and concern

issues such as how the model is trained, how data flows from raw PCAPs to learned representations, and whether the system can be reproduced and deployed by others. In this section, we describe the key engineering decisions we made and the reasoning behind each decision.

6.4.1 Pretraining Data and Model Variants

Pretraining data. We pretrain netFound on a dataset collected from a large U.S. university campus, comprising multiple weeks of traffic captured at the campus border. The resulting corpus contains **four billion** diverse network flows (∼1.2TB after preprocessing), spanning a mix of residential, enterprise, and research traffic.[1] The source, scale, and diversity of the pretraining data matter for a number of reasons.

For one, NFMs trained on narrow traffic mixes risk learning dataset-specific or application-specific patterns rather than generalizable traffic representations, and the same shortcut problem that Pcap-Encoder [265] identified at the evaluation level can also occur at the pretraining level. As we show later in §6.5.2, since most of the publicly available models use the same datasets for pretraining and fine-tuning [144, 242, 264, 266], they benefit from leakage of test data into the pretraining data which, in turn, enables them to achieve excellent performance on the considered fine-tuning tasks. In addition, the burst-flow hierarchical attention mechanism (§6.3.3) requires sufficient diversity of burst structures to learn meaningful cross-burst patterns rather than memorizing the burst-length distribution of a single network. Using diverse network data collected from a production network with tens of thousands of users allows us to expose netFound to a wide range of different network conditions and application behaviors, simultaneously avoiding skewing the fine-tuning results and a model's ability to generalize. Limitations

[1] We attempted to increase diversity by adding MAWI [57] and CAIDA [198] to the mix, but did not observe any significant improvements in terms of intrinsic evaluation or downstream performance and discarded this approach.

of this approach are discussed in §6.7.

Three model sizes. We independently pretrain three model variants: small (four layers and four attention heads, 512-dim, 53M parameters), base (12 layers and 12 attention heads, 768-dim, 225M parameters), and large (24 layers and 16 attention heads, 1024-dim, 663M parameters). The rationale for multiple sizes is deployment flexibility: a small model may suffice for edge or near-real-time inference where latency is critical, while a large model is appropriate for offline analysis where accuracy matters most. The efficiency evaluation in Section 6.6 quantifies this trade-off.

Multi-stage pretraining. We implement multiple pretraining stages, varying loss weights to encourage progressively more complex dependencies: early stages emphasize MLM for token-level semantics, while later stages increase the weight on burst-level and flow-level objectives. For the entire duration of pretraining, we use AdamW [150] with a Warmup-Stable-Decay learning rate schedule (peak 1e-4), reducing the learning rate on subsequent stages. In total, we utilize 128 NVIDIA A100 GPUs, with approximately 10 billion tokens seen across stages for each model, with flows randomly sampled in a streaming manner from the pretraining dataset.

6.4.2 Data Pipeline and Reproducibility

C++ acceleration. The data preprocessing pipeline comprises five automated stages: protocol filtering, flow-based PCAP splitting, field extraction, tokenization, and Apache Arrow shard assembly. The first three stages (filtering, splitting, field extraction) are computationally intensive; instead of blindly splitting packets by byte borders, they process raw packet captures byte-by-byte, searching for specific fields and flags. We implement them in C++ using PcapPlusPlus [187], providing an order-of-magnitude speedup over pure-Python alternatives. This investment is essential because the 4-billion-

flow pretraining corpus would be infeasible to process in Python within a reasonable time budget; efficient intermediate representation structure makes the full data pipeline a one-time cost rather than a recurring bottleneck. The pipeline supports both pretraining (unlabeled) and fine-tuning (labeled) modes, and produces Apache Arrow shards that enable memory-mapped streaming, directly integrating with HuggingFace interface and accelerating data access by workers. In addition, we split big datasets into hundreds of shards to enable parallel data access for training.

Containerized, reproducible workflow. The model's entire workflow (raw PCAP ingestion through preprocessing, pretraining, and fine-tuning) is containerized via Docker and orchestrated through a single Makefile. The Docker image bundles all system-level dependencies, including PcapPlusPlus for C++ packet parsing, CUDA toolkits, and Python libraries with pinned versions. Containerization matters because network traffic analysis tools have notoriously fragile dependency chains (specific versions of libpcap, protocol dissectors, CUDA drivers), and a user should be able to reproduce any experiment by simply providing input PCAPs and a configuration file, without any manual environment setup.

6.4.3 Extensibility and Testing

All aspects of netFound, such as model architecture, protocol field selection, tokenizer behavior, training hyperparameters, and loss weights, are controlled through a unified, hierarchical configuration system compatible with HuggingFace configurations. This makes it straightforward to extend netFound to new protocols or customize the representation for a specific downstream task without modifying source code.

The codebase is organized into self-contained modules (tokenizer, embeddings, layers, attention, collator, sampler, trainer, metrics, configuration, and utilities), each with a well-

defined interface. The fine-tuning and pretraining models share a common base encoder, so downstream tasks can be added by implementing a new prediction head without modifying the core model. The trainer extends the HuggingFace Trainer with domain-specific columns and callbacks, preserving compatibility with the broader ecosystem (e.g., Weights & Biases logging, DeepSpeed integration, gradient accumulation strategies, Accelerate, etc).

We maintain a model-specific test suite covering core infrastructure components: the custom tokenizer, length-bucketed sampler, data collator with burst swapping and metadata masking, attention mask transformations, and tensor reshape operations. These tests exercise edge cases (e.g., empty buffers, single-example batches, variable burst lengths) and verify numerical correctness using parametrized fixtures, providing confidence that modifications to the codebase or model's extensions do not introduce silent regressions.

6.5 Evaluation

In this section, we evaluate and analyze netFound's performance across multiple dimensions, including intrinsic representation quality of the encoder's embeddings and downstream task performance. We also report on the overall computational efficiency of netFound in Section 6.6.

Models and datasets. We compare netFound (small, base, and large checkpoints) against five publicly available network foundation models that represent distinct design philosophies: ET-BERT [144], a BERT-based model pretrained on encrypted payload bytes; YaTC [264], a vision-transformer approach that treats traffic as images; NetMamba [242], a state-space model for network traffic; TrafficFormer [266], a BERT-style transformer using both packet headers and payload information; and Pcap-Encoder [265], a T5-based model proposed alongside a corrected evaluation protocol. For all competitors, we use

the publicly released pretrained checkpoints without modification. The datasets used for intrinsic and downstream evaluations differ, and the relevant differences are detailed in the next two subsections. Note that for the reasons mentioned in Section 6.7, we intentionally omitted an ablation study.

6.5.1 Intrinsic Evaluation

Recent works [30] demonstrated that evaluating network foundation models solely through downstream fine-tuning performance conflates the quality of pretrained representations with confounding factors such as model capacity, classification head design, and optimization dynamics, making it difficult to attribute gains to the pretraining itself. An intrinsic evaluation that directly probes the learned representations in a task-agnostic manner isolates the contributions of these confounding factors and allows for a more meaningful assessment of what the encoder has actually learned during pretraining.

Here, we adopt the Intrinsic Evaluation Framework (IEF) [30], a task-agnostic methodology that probes the representational quality of network foundation models through three complementary lenses: embedding geometry analysis quantifies how effectively models distribute representations across the latent space; metric alignment assessment measures correspondence between learned embeddings and established features derived by domain experts; and causal sensitivity testing evaluates whether models capture higher-order network context invisible in the raw traffic bytes, such as congestion control behavior and queue management policies.

In this subsection, we rely on the four datasets that were used in IEF: Crossmarket [237], CIC-IDS2017 [214], CAIDA [198], and MAWI [57]. These datasets cover a range of different traffic conditions, including endogenously (synthetically) and exogenously (production) generated data, and provide a comprehensive suite for intrinsic evaluation.

Embedding Geometry

Mean cosine similarity between random pairs of flow embeddings measures the average distance between different network flows from the model's perspective: the further the distance, the more different these flows are in the embedding space of the model. For a given dataset, this average distance is expressed in terms of the anisotropy metric, where values close to 1 indicate that the model collapses flows into nearly identical representations, while lower values suggest that the model distributes information across the available dimensions and can distinguish meaningful variations in network behavior. At the same time, values close to 0 demonstrate that the model cannot find any similar flows and spread embeddings far, making clusterization and analysis impossible. We calculate mean cosine similarity between the embeddings produced for the benchmark datasets and report the results in Table 6.3.

Table 6.3: Mean cosine similarity between flow embeddings. Bold shows results that differ significantly from competitors.

Model	Crossmarket	CIC-IDS2017	CAIDA	MAWI	Avg.
Random	0.00	0.00	0.00	0.00	0.00
ET-BERT	0.88	0.74	0.87	0.85	0.84
YaTC	0.85	0.85	0.86	0.85	0.85
NetMamba	0.93	0.92	0.99	0.99	0.96
TrafficFormer	0.74	0.68	**0.60**	**0.54**	**0.64**
Pcap-Encoder	0.86	0.77	0.69	0.84	0.79
netFound-small	0.68	0.65	0.79	0.78	0.72
netFound-base	**0.63**	**0.60**	0.73	0.77	**0.68**
netFound-large	**0.64**	**0.55**	0.68	0.76	**0.66**

Takeaway. Our findings demonstrate that while YaTC and NetMamba collapse embeddings into near-identical representations, TrafficFormer, Pcap-Encoder, ET-BERT, and all versions of netFound maintain substantially lower cosine similarity, indicative of a non-degenerate geometry. TrafficFormer achieves the lowest average value (0.64), but anisotropy is a diagnostic rather than a winner-take-all metric: lower cosine similarity

is only of relevance if it reflects some meaningful structure, and TrafficFormer's low anisotropy is plausibly explained by its access to encrypted payload bytes. While these bytes are not semantically informative, they can be expected to add input diversity and result in richer geometric structure.

When examined through the lens of anisotropy, netFound achieves the second-lowest average cosine similarity (0.66–0.72), preserves user's privacy, and outperforms TrafficFormer on the endogenous Crossmarket and CIC-IDS2017 datasets, where netFound-large achieves 0.64 and 0.55 versus 0.74 and 0.68. These findings confirm that the combination of protocol-aware tokenization (Principle 1), operational context embedding (Principle 2), and burst-flow hierarchical attention (Principle 3) helps to distribute representations more uniformly across the latent space, directly addressing the embedding collapse problem identified by IEF. We also note that netFound typically exhibits higher anisotropy on the exogenous CAIDA and MAWI datasets due to them containing less expressed packet header variations. Given that the privacy-by-construction input design (Principle 4) excludes any payload information, this is expected behavior rather than a model limitation.

Metric Alignment

In IEF, Centered Kernel Alignment (CKA) [133] measures the structural similarity between a model's flow embeddings and domain expert-derived features (e.g., flow duration, packet size distributions, etc.) computed by CICFlowMeter [140]. This metric measures whether the hidden network representation from a foundation model indeed contains the statistical information that correlates with each flow and is used by network domain experts and does not simply represent the raw byte values of the packets. Higher CKA values (near 1) indicate that the model's latent space implicitly encodes well-known features, validating that it has learned generalizable network semantics rather than superficial correlations.

We calculate the CKA metric for each model and report the computed values in Table 6.4. For comparison, we also report the CKA values for randomly generated embeddings and for oracle embeddings that directly represent the computed CICFlowMeter features.

Table 6.4: CKA similarity between model embeddings and CICFlowMeter features (higher is better).

Model	Crossmarket	CIC-IDS2017	CAIDA	MAWI	Avg.
Random	0.000	0.000	0.000	0.000	0.000
ET-BERT	0.012	0.064	0.033	0.026	0.047
YaTC	0.098	0.092	0.014	0.070	0.068
NetMamba	0.047	0.042	0.030	0.051	0.042
TrafficFormer	0.138	0.083	**0.053**	0.057	0.082
Pcap-Encoder	0.103	0.096	0.036	**0.082**	0.079
Oracle (ideal)	1.000	1.000	1.000	1.000	1.000
netFound-small	**0.154**	**0.149**	0.046	0.059	**0.102**
netFound-base	**0.154**	**0.143**	0.045	0.030	**0.093**
netFound-large	**0.155**	**0.144**	0.046	0.033	**0.094**

Takeaway. All versions of netFound achieve the highest average CKA (0.093–0.102), with the best competitors (Pcap-Encoder and TrafficFormer) scoring 0.079 and 0.082, respectively. The gap is most pronounced for the endogenous datasets (Crossmarket and CIC-IDS2017), where netFound achieves 0.154 and 0.149, improving over 0.138 and 0.096 from the closest competitors (TrafficFormer and Pcap-Encoder). These results confirm that the explicit operational context embedding (Principle 2) grounds netFound's representations in the operational features that domain experts rely on. The relatively modest alignment results for the CAIDA and MAWI datasets are consistent with those datasets' skewed ratio of shorter network flows which prevents the CICFlowMeter features from exhibiting realistic variability, causing netFound's CKA values to be close to those of the competitors.

Exogenous Network Context

To test whether models capture unobserved higher-order network conditions, we use the synthetically generated flows published by IEF [30] which contain controlled variations in congestion control algorithm, active queue management policy, and cross-traffic patterns. Since these flows represent the same application-level behavior and have identical payload under different network contexts, they allow us to examine whether embeddings reflect these contexts or simply memorize the encrypted data. A model that meaningfully encodes these exogenous factors should produce embeddings that are linearly separable by context, a property that can be measured via a logistic-regression probe using an F_1 score. As noted in IEF, this evaluation differs from a downstream performance evaluation in the sense that it utilizes the simplest linear probe design to only assess a linear combination of the encoder's output embeddings. In effect, this evaluation isolates the representational quality of the pretrained model and prevents conflating it with the capacity of a learned classification head.

Table 6.5: Exogenous context discrimination: linear probing performance using F_1 score (higher is better). CC, AQM, CTP, and All represent Congestion Control, Active Queue Management, Cross-Traffic Patterns classifications, and all changes combined respectively.

Model	CC	AQM	CTP	All
ET-BERT	0.41	0.68	0.43	0.46
YaTC	0.48	0.51	0.24	0.47
NetMamba	0.29	0.70	0.33	0.62
TrafficFormer	0.36	0.55	0.49	0.57
Pcap-Encoder	0.00	0.68	0.17	0.59
netFound-small	**0.64**	**0.86**	**0.79**	**0.90**
netFound-base	**0.57**	**0.84**	**0.74**	**0.93**
netFound-large	**0.65**	**0.90**	**0.81**	**0.95**

Takeaway. All three versions of netFound achieve excellent F_1 scores (0.90–0.95) on the combined exogenous context discrimination task ("All"). In contrast, the score of the best competitor (NetMamba) is 0.62. We observe the same pattern across all individual

Table 6.6: Weighted F_1 score on flow classification downstream tasks with bold highlighting the results that are significantly higher than competitors. Underlined red scores show benchmarks that were utilized during pretraining by corresponding models. Even in this unfair setting, netFound demonstrates overwhelming performance for frozen and on-par for unfrozen scenarios.

Model	USTC (20 classes)		USTC (binary)		ISCX-VPN		CIC-IDS2017		Crossmarket (Acc@10)	
	Frozen	Unfrozen	Frozen	Unfrozen	Frozen	Unfrozen	Frozen	Unfrozen	Frozen	Unfrozen
ET-BERT	0.8533	0.8935	**0.9998**	0.9999	0.6684	0.8645	0.8580	0.8783	0.1857	0.3283
YaTC	**0.9396**	0.9396	0.9974	0.9978	0.5603	0.5603	0.9843	0.9999	0.2077	0.6348
NetMamba	0.4191	**0.9881**	0.6496	0.9999	0.1362	**0.9364**	0.9146	0.9998	0.1733	0.7149
TrafficFormer	**0.9555**	0.9789	**0.9999**	1.0000	0.6459	0.8712	**0.9916**	0.9995	0.2322	0.5923
Pcap-Encoder	0.8799	0.9775	**1.0000**	0.9999	0.3967	0.9032	0.9691	0.9883	0.2257	**0.8699**
netFound-small	0.8550	0.9645	**0.9999**	1.0000	**0.7508**	0.8920	**0.9928**	0.9996	**0.3804**	0.4727
netFound-base	0.8908	0.9695	**0.9999**	1.0000	**0.8053**	**0.9175**	**0.9918**	0.9997	**0.4330**	0.6066
netFound-large	**0.9259**	0.9674	**0.9999**	1.0000	**0.8098**	0.9004	**0.9938**	0.9998	**0.4561**	0.6538

contexts: even netFound-small scores 0.64/0.86/0.79 on CC/AQM/CTP versus the best competitors scores of 0.48/0.70/0.49 (YaTC, NetMamba, TrafficFormer).

These findings are particularly telling. Since the flows have the same payload across different contexts, a model must rely on temporal patterns (e.g., inter-packet arrival times, burst structures) to achieve a high level of discrimination. For netFound, the operational context embedding (Principle 2) and burst-flow hierarchical attention mechanism (Principle 3) are the enablers: congestion control, queue management, and cross-traffic shape the *temporal* structure of bursts rather than packet content, and netFound's architecture is explicitly designed to capture these multi-scale patterns. This observation reinforces the benefits of the diagnostic-to-design approach advocated in this paper: the principles derived from the context discrimination findings reported in [30] give rise to a model that excels exactly at the task of network context discrimination.

6.5.2 Downstream Performance

To assess the practical utility of representations, we evaluate netFound on five downstream tasks spanning encrypted traffic classification, malware detection, intrusion detection, and application fingerprinting. We compare against the same five published

state-of-the-art network foundation models: ET-BERT [144], YaTC [264], NetMamba [242], TrafficFormer [266], and Pcap-Encoder [265]. All models are evaluated in two regimes: **frozen**, where the pretrained encoder weights are fixed and only the model-defined head is trained on top of the encoder's output representation, and **unfrozen**, where the full model is fine-tuned end-to-end. The frozen setting allows us to directly assess the quality of the learned representations: a high score is a strong indication that the pretrained embeddings already capture task-relevant structure, making the model usable as a feature extractor without expensive retraining. The unfrozen setting enables us to measure the model's overall capacity when all parameters can be adapted to the target task, including the model's potential for learning a dataset's shortcuts. Importantly, where prior work explicitly benefited from leveraging inconspicuous dataset shortcuts (see Pcap-Encoder [265], Table 6.2, and the discussion in §6.2.2), the results we report below are based on the *corrected* versions of datasets that prevent learning the mentioned shortcuts.

Datasets. Table 6.7 summarizes the five downstream benchmarks used in our evaluation, chosen to cover a diverse range of task types, class granularities, and traffic conditions.

Table 6.7: Downstream evaluation datasets.

Dataset	Task	Classes	Metric
USTC-TFC [244]	App classification	20	F_1
USTC-TFC [244]	Malware detection	2	F_1
ISCX-VPN [77]	VPN traffic classif.	17	F_1
CIC-IDS-2017 [214]	Intrusion detection	8	F_1
Crossmarket [237]	App fingerprinting	210	Acc@10

The USTC-TFC dataset provides two complementary classification tasks over the same encrypted traffic: a fine-grained 20-class application identification task and a binary malware-vs-benign detection task, testing both multi-class discrimination and coarse security triage. ISCX-VPN evaluates the ability to characterize traffic within encrypted tunnels, a setting where header information is partially obscured. CIC-IDS-2017 targets

intrusion detection across 7 attack categories (a separate 8th category is used for benign traffic), stressing the model's capacity to distinguish subtle behavioral anomalies. Finally, Crossmarket is a large-scale application fingerprinting benchmark with 210 application classes, evaluated via *Acc*@10 (accuracy within the top-10 predictions), which allows for rigorous representation quality testing that is especially relevant for retrieval- and similarity-based tasks.

Analysis

The frozen columns in Table 6.6 reveal how much task-relevant structure the pretrained encoder captures *before any task-specific encoder weight updates*. Across netFound's three model sizes, frozen representations are already highly discriminative: even the smallest model variant achieves weighted F_1 scores of $0.75 - 0.99$, demonstrating that a 53M-parameter frozen encoder can efficiently represent diverse network flows.

A clear scaling trend emerges in the frozen setting. On USTC-20, the frozen F_1 scores increase monotonically from 0.855 (small) to 0.891 (base) to 0.926 (large), a 7-point improvement driven solely by additional model capacity during pretraining. The same pattern holds for ISCX-VPN ($0.751 \rightarrow 0.805 \rightarrow 0.810$) and Crossmarket ($0.380 \rightarrow 0.433 \rightarrow 0.456$), confirming that larger models internalize richer traffic representations. The effect is most pronounced on more challenging tasks: Crossmarket, which requires discriminating among 210 applications via ranking, shows a relative improvement of $\sim$20% from small to large, whereas nearly-saturated tasks like USTC binary show no headroom for further gains.

The observed frozen-to-unfrozen gap can be viewed as providing an additional diagnostic tool. On tasks where frozen scores are already high (USTC-binary and CIC-IDS-2017), fine-tuning yields negligible additional benefit, indicating that the pretrained representations already capture the relevant decision boundary. Conversely, tasks with a

larger frozen-to-unfrozen gap, such as VPN ($0.751 \rightarrow 0.892$ for small) and Crossmarket ($0.380 \rightarrow 0.473$ for small), suggest that while the pretrained model provides a good starting point, task-specific adaptation can unlock additional performance. Notably, this gap narrows with scale: for USTC-20 and ISCX-VPN, the gap between the small and large model variants shrinks ($0.109 \rightarrow 0.042$ and $0.141 \rightarrow 0.091$, respectively). Moreover, on Crossmarket, the large model's frozen score (0.456) already approaches the small model's unfrozen score (0.473), illustrating that scaling pretraining can partially substitute for task-specific fine-tuning.

In addition to being the best model in the frozen setting, the unfrozen version of netFound emerges as the top performer across all the benchmarks, exceeding or matching the competitors' performance in most of the cases (see Table 6.6). This property holds even for the small model variant, indicating that netFound's architecture and pretraining strategy provide a practical and exceptionally strong starting point for downstream adaptation.

Test-into-train leakage. It is worth noting that for many models, their reported performance is not only a function of architectural decisions but, due to the limited number of publicly available pretraining datasets, can also be impacted by the use of pretraining data that includes fine-tuning datasets. The use of such augmented pretraining data allows models to generate good quality fine-tuning task-specific embeddings but limits their ability to generalize to previously unseen data. The results for such models and datasets are highlighted in red in Table 6.6. We observe that most of the frozen models achieve low F_1 scores on the Crossmarket dataset, which was not used for pretraining by any of the models with the exception of NetMamba. netFound's choice of pretraining data prevents any leakage of test data to the frozen encoder, resulting in smaller gaps between frozen and unfrozen setting than other models. Even in this disadvantageous for netFound setting and despite its fully privacy-preserving design (i.e., no payload data),

netFound achieves the best performance in the frozen setting and is comparable with the best competitors in the unfrozen setting.

6.6 Cost & Efficiency

Table 6.8: Unfrozen model performance on a single GPU A100 80Gb with BF16 enabled.

Model	Params	Batch	Training			Inference		
			Mem (MB)	Latency (ms)	flows/sec	Mem (MB)	Latency (ms)	flows/sec
small		1	1,549	34.9 ± 1.2	29 ± 1	803	11.6 ± 0.3	86 ± 2
small	53M	8	2,641	37.7 ± 0.7	212 ± 4	967	15.2 ± 0.9	528 ± 32
small		32	5,951	81.9 ± 2.6	390 ± 13	1,563	38.5 ± 0.7	832 ± 16
base		1	3,899	89.3 ± 1.6	11 ± 1	1,527	29.8 ± 0.4	33 ± 0.5
base	225M	8	8,189	106.4 ± 7.4	75 ± 6	1,933	40.0 ± 0.8	200 ± 4
base		32	22,103	303.0 ± 3.6	106 ± 1	2,871	116.3 ± 2.6	275 ± 6
large		1	12,591	185.2 ± 9.7	5 ± 1	4,521	56.3 ± 0.8	18 ± 1
large	663M	8	28,353	277.8 ± 33.9	29 ± 2	5,225	100.0 ± 9.1	80 ± 8
large		32	77,307	952.4 ± 8.9	34 ± 1	6,717	337.8 ± 1.1	96 ± 1

To support deployment decisions of netFound, we report the model's computational profile across different sizes. These measurements validate the engineering investments described in Section 6.4 and enable practitioners to select the appropriate model variant for their deployment constraints.

Training Cost. We utilize a dynamic pretraining strategy, beginning with 128xA100 40Gb GPUs for the initial pretraining stage and reducing the number of GPUs to 80 for subsequent stages together with lowering learning rate and changing objectives weights. In total, the entire pretraining process for all three model variants consumed ∼5,000 GPU-hours, with small, base, and large models pretraining for approximately 45K, 75K, and 35K steps respectively, and took ∼12-36hrs for each checkpoint. To provide a comparison baseline, NVIDIA estimates pretraining of a single 220M T5 base checkpoint (which is used in Pcap-Encoder) from scratch as ∼3000 GPU hours [230] and 1T of tokens using internal highly optimized training pods setup. Our lower pretraining cost, together with intrinsic and extrinsic evaluation results, highlights the benefits of network-optimized

architecture.

Throughput and Memory Footprint. Table 6.8 shows peak consumed GPU memory (in MB), latency per batch (in ms), and throughput (flows/sec) for both training and inference across different model and batch sizes. The three model sizes offer a clear throughput–accuracy trade-off, as low as 11ms per flow, enabling deployment on hardware ranging from edge devices (small) to GPU clusters (large). Compared with Pcap-Encoder, which uses 2,989 MB GPU memory and 230ms for a single inference step with a batch size of 32 on the same hardware, our base model achieves 2x faster inference with the same number of parameters and memory taken.

Preprocessing Throughput. The C++-based preprocessing pipeline described in §6.4 processes 2000 flows/sec during filtering, 216 flows/sec during splitting, and 970 flows/sec during fields extraction on 128 cores, taking in total 12hrs to process a single dataset chunk of 7mln flows. Tokenization and Apache Arrow shard assembly add 4 hours walltime (500 flows/sec in Python). As a result, the full 4-billion-flow pretraining corpus is estimated to take ∼7000 CPU hours on a single 128-core machine. Implementing the same preprocessing in pure Python is estimated to take x10-x100 more walltime for the full dataset. To reduce efforts for preprocessing a dataset of such scale, we also publish the full preprocessed pretraining dataset (see Availability for details).

6.7 Limitations

Absence of Full Ablation Study. netFound incorporates multiple interacting design decisions: protocol-aware tokenization, operational context embedding, burst-flow hierarchical attention, four pretraining objectives, and others. A comprehensive ablation study would require pretraining each variant of netFound from scratch for every combination of removed substituted model components, consuming thousands of GPU-hours on up to

128 A100 GPUs, rendering the resulting cost prohibitive. Large language model efforts (e.g., LLaMA [231], ModernBERT [245]) face a similar problem and omit ablation studies over their design space for the same reasons. Instead, we justify individual design choices through (i) prior evidence in the literature (e.g., RoPE, GeGLU, Flash Attention have been extensively validated in NLP), (ii) domain-specific motivation grounded in the structure of network data (e.g., hierarchical attention mirrors the burst/flow hierarchy), and (iii) intrinsic and downstream evaluation of the final trained models across multiple tasks and datasets. We acknowledge that isolating the marginal contribution of each component remains an open question, and we encourage future efforts that are less encumbered by tight compute budgets to explore this topic.

Encoder-Only Architecture. netFound is built on an encoder-only transformer, which makes it well suited to discriminative tasks such as classification, anomaly detection, and traffic characterization. However, the current design precludes the use of netFound for generative capabilities, e.g. synthetic traffic generation, packet-level forecasting, or open-ended reasoning about network traffic traces. Extending the model to an encoder-decoder or decoder-only design to support such generative use cases is a promising direction for future work.

Pretraining data limitations. The single-campus pretraining data is geographically concentrated and U.S.-specific, potentially limiting the applicability of the pretrained models to different environments, such as enterprise datacenter networks, IoT ecosystems, or countries with significantly different backbone traffic. To mitigate this limitation, we release the full source code so as to facilitate pretraining on user-specific data and encourage the reproducibility of our approach.

Reliance on evaluation benchmarks. Pcap-Encoder [265] and other works [30,126,148] demonstrated that many existing evaluation benchmarks can have shortcut-susceptible label distributions, are based on problematic evaluation designs, or can be biased towards

measuring specific model qualities. Given that our design effort is in large parts inspired by the findings reported in these prior works, we acknowledge a potential bias in our choice of relevant design principles. To keep an eye on such a potential bias and prevent it from affecting our approach, we examine and validate our design decisions from multiple perspectives (e.g., intrinsic evaluation, studying frozen and unfrozen model settings, and efficiency).

6.8 Conclusion

In this paper, we present netFound, a diagnostic-informed, production-ready network foundation model. Building on the NFM failure analysis conducted by preliminary works, we translate these findings into four network-native design principles and realize them in an efficient, privacy-preserving, system-driven approach, implementing multiple novel architectural components and training techniques. Across intrinsic evaluation, frozen, and unfrozen downstream analysis, netFound demonstrates stronger reusable representations via lower anisotropy, higher correlation with domain-expert features, better sensitivity to exogenous network context, robustness to shortcuts, and top performance across all five downstream tasks. We pair this model with a full release of code, pretrained checkpoints, the pretraining dataset, and an end-to-end pipeline from raw PCAPs to inference so that the community can reproduce, extend, and deploy the system in practice. By this, we aim to provide a foundation for future application of diagnosis-to-design methodology and motivate the community to drive the next generation of architectural improvements as new diagnostic tools and public datasets emerge.

Availability

We make netFound publicly available as a reusable research artifact. The release includes the pretrained checkpoints for all three model sizes (small at `https://huggingface.co/snlucsb/netFound-small`, base at `https://huggingface.co/snlucsb/netFound-base`, and large at `https://huggingface.co/snlucsb/netFound-large`) on HuggingFace, as well as the source code for system at `https://github.com/SNL-UCSB/netFound` and all claims from this paper at `https://github.com/maybe-hello-world/netfound-paper`. In addition, we also *publish the whole pretraining dataset*, containing more than 4 billion network flows in Apache Arrow format, suitable for pretraining of netFound or any future model utilizing a similar tokenizer. The dataset download instructions are available in the netFound GitHub repository at `https://github.com/SNL-UCSB/netFound`. Together, these artifacts provide an end-to-end, reproducible pipeline for training and applying netFound to various network analysis tasks and problems.

Chapter 7

Leveraging Large Language Models to Contextualize Network Measurements

7.1 Introduction

With the worldwide growth of remote communication and telepresence [100], network measurements form a cornerstone of effective performance assessment and diagnostics for Internet users. The value of such measurements is particularly evident when pinpointing network bottlenecks or validating service-level agreements with the Internet Service Provider. Most often, users seek for overall connection performance measurement using publicly available tools (also known as 'speed tests' [60, 159, 181]) that provide an overview of their connection's throughput and latency.

However, extracting meaningful insights from these measurements remains a challenging task for a non-technical audience. Interpreting network measurement data often requires considerable domain expertise to account not only for subtle variations of the connection stability and metrics, but even for simpler concepts such as latency under load or packet loss influence towards connection performance. In the absence of proper

expertise, common misconceptions can easily arise. A notable example is the widespread reliance on download bandwidth measurements [185] reported by speed test tools, which might give a false sense that the connection achieves the expected performance. In reality, the connection stability, packet loss, latency variations, and other factors can significantly impact the test interpretation, introducing disconnection between reported test results and subjective user experience, or leading to misguided decision-making when provisioning or troubleshooting network services.

To address these issues, researchers should recognize the importance of making network measurements not only more comprehensive but also more accessible for wider audience without deep technical knowledge. A promising direction to achieve this goal involves leveraging recent advancements in large language models (LLMs), which have demonstrated capabilities in conducting an analysis of complex data in other fields, such as laboratory test results interpretation [116], news summarization [261], and personal assistance [74].

In this paper, we describe an ongoing effort to apply large language models and historical data to enhance the interpretation of network measurements in real-world environments. We aim to automate the translation of low-level metric data into accessible explanations, allowing non-experts to make more informed decisions regarding network performance and reliability.

7.2 Proof of Concept

To demonstrate the potential of LLMs in enhancing network measurement interpretation, we adapted a publicly available framework for network measurements, netUnicorn [29], to incorporate Large Language Models (LLMs) for parsing and interpreting the results. We developed a custom netUnicorn pipeline that allows users to run network measure-

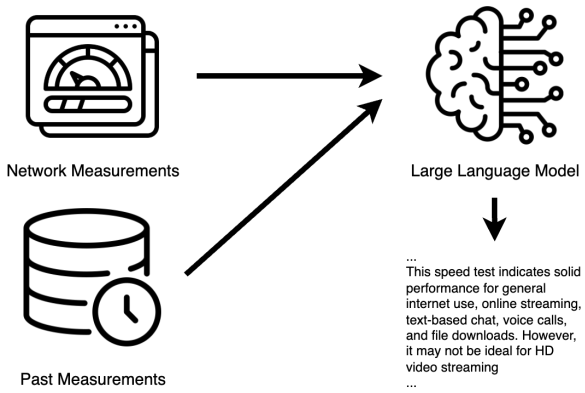


Figure 7.1: Components of the network measurements framework.

ments (speedtest) using the Ookla CLI tool [181]. The results contain widely used metrics (such as download and upload throughput, latency, etc) which are often presented to the user in the raw format. To improve understanding of these results, we incorporated an additional step in the pipeline that sends the raw results to a separate endpoint for further interpretation and analysis. This endpoint filters the results, removing unnecessary or private information, i.e., IP and MAC addresses, test ID, timestamp, unique URLs, etc.

After cleaning up the raw results, we apply a Large Language Model to interpret and write a summary for the user. For this purpose, we used a locally hosted version of the DeepSeek-R1 Distill Qwen model with 7B parameters, trained on a diverse corpus of text data, including technical documentation and websites content. The model was hosted using LM Studio [149], which provides an endpoint for text analysis and generation requests using an OpenAI API-compatible interface. The choice of the interface and the hosting solution allows us to easily integrate different models (including open-source and proprietary) into our pipeline, providing flexibility and scalability for future improvements.

To improve the accuracy of the results, we also added a geolocation step to the pipeline. This step uses the user's IP address to determine their location via IP Geolocation API [123] and then provides approximate anonymized information (with city-level granularity) about the user's location to the LLM. This information is used to provide more relevant and

localized insights into the network performance, taking into account the user's geographical context and associated with it network conditions.

To further enhance the fidelity of network measurements interpretation, we explored the usage of Retrieval-Augmented Generation (RAG) techniques which combine the strengths of LLMs with external knowledge sources containing historical data, allowing the model to retrieve relevant information from a database or knowledge base and incorporate it into the model responses. For our framework, we implemented a knowledge database of past network measurements using publicly available data from the FCC Measuring Broadband America program [88]. This data contains various measurements from different locations and ISPs, including download and upload throughput obtained via running speed test software. After receiving the raw measurement results from the netUnicorn pipeline, our framework queries the local PostgreSQL knowledge database to find similar measurements based on the measurement server location and time of day. We calculate statistics over retrieved data, such as average fetch time, throughput mean, median, and standard deviation values, and provide them to the LLM as an additional context together with the raw results and geolocation information to use the historical context of their measurements to provide more accurate and relevant insights into the current network performance.

We designed and implemented a custom prompt for the LLM that includes instructions to focus on key metrics while also considering the contextual information (i.e., key spatial, temporal, and semantic relationships between measurements from different tools, locations, and times). This guides the model to provide a concise and informative summary that highlights the most important aspects of the results, targeting a non-technical audience to facilitate the analysis.

The resulting summary is sent back to the user, providing them with overall information, a detailed explanation of each metric, and their potential impact on user experience for

different use cases, such as online gaming, video streaming, internet browsing, etc. The summary also might include recommendations for improving network performance if applicable, such as optimizing wireless access point placement, using wired connections, or upgrading the internet plan.

7.3 Further research

The current implementation of our framework has several significant limitations. First, the choice of large language models can significantly impact the generated summaries and bigger models may provide more accurate results and prevent hallucinations or misinterpretations by increasing the cost of running the framework and require more computational resources (for both inference and fine-tuning the model on specific networking literature for analysis improvement). Second, the accuracy of the geolocation step depends on the quality of the IP address lookup services and methods the user or ISPs are using to hide their current location. Finally, the current knowledge database is limited to the FCC MBA program data, which may not provide all desired metrics or cover all geographical areas.

More fundamentally, we aim to explore challenges that arise when using LLMs for measurements interpretation:

Trust and reliability. How can we ensure that the LLM-generated summaries are accurate and trustworthy? What measures can be taken to validate the results and prevent misinformation or hallucinations of the model?

Overemphasis on certain metrics or contextual information. Can we prevent the model from highlighting certain metrics or contextual information that may not be relevant to the user's specific situation? How can we ensure that the model provides a balanced and comprehensive analysis of the network performance?

Data diversity and bias. Given that performance of LLM is shaped by the training data, how can we make results less biased and more representative of the real-world network?

Privacy issues. How can we strike a balance between providing relevant contextual information and user privacy?

User experience. Can we ensure that provided results significantly improve the user experience and understanding of network performance? How can we design the framework to be user-friendly and accessible to a non-technical audience?

We believe that addressing these questions and challenges is crucial for the reliable and useful implementation of LLMs in network measurements interpretation, which will avoid misleading or incorrect conclusions and provide users with accurate and actionable insights into their network performance.

Part IV

Conclusion and Future Directions

Chapter 8

Conclusion

This dissertation argues that the central challenge for machine learning in networking is not only how to achieve high accuracy, but how to make learned systems credible enough to support real operational use. Across network security, measurement, and traffic representation learning, high benchmark performance repeatedly proves weaker than deployment credibility: models must learn signals that survive changes in environment and produce decisions operators can rely on.

The central conclusion of this dissertation is that diagnosis, data collection, and model design are not independent steps that can be optimized in isolation, but mutually reinforcing parts of a single methodology. When these concerns are separated, failures compound across the pipeline: data artifacts shape model behavior, evaluations hide those artifacts, and generic architectures ignore the structure of network traffic. When they are treated together, diagnosis reveals what models learn, data collection targets the discovered failures, and model design translates those lessons into network-native representations.

Part I establishes the diagnostic foundation for this methodology by shifting the unit of analysis from aggregate benchmark performance to the behavior of the learned

system itself. It shows that network ML artifacts and network foundation models can appear successful while relying on shortcuts, spurious correlations, brittle out-of-distribution behavior, collapsed representations, weak domain-feature alignment, or payload dependence. The broader contribution of this part is therefore not only the identification of individual failures, but the development of practical ways to make such failures visible. Through model explanations, trust reports, intrinsic representation analysis, and controlled perturbations, Part I turns credibility from a vague deployment concern into an empirical question that researchers can ask of concrete models and representations.

Part II addresses the data side of the same problem by treating data collection as infrastructure rather than as a one-time experimental setup. PINOT and netUnicorn make realistic and reusable data collection practical across production, campus, cloud, and experimental environments, and they do so while preserving the ability to adapt data collection to the failure modes discovered during diagnosis. This part emphasizes that generalizability cannot be obtained simply by asking a model to be more robust after training; it depends on whether the training and evaluation process exposes the model to the right operational diversity. Together, these systems support the closed-loop workflow that credible network ML requires: diagnose failures, collect targeted in-vivo data, retrain, and repeat until the model's behavior is supported by evidence rather than by assumptions about a static dataset.

Part III completes the arc by translating the diagnostic and data-centric lessons into model design. netFound represents the model-side synthesis of the dissertation because its architecture is shaped by the failures exposed earlier: protocol-aware tokenization preserves field structure, operational context embeddings ground representations in information that network experts use, burst-flow hierarchical attention reflects the natural organization of traffic, and privacy-by-construction inputs reduce dependence on signals

that are unavailable or inappropriate in deployment. The LLM contextualization work complements this direction by showing that even general-purpose pretrained models become useful for networking only when they are grounded with measurement context, retrieval, geolocation, and privacy filtering. In both cases, the point is the same: model capacity alone is not enough, and credible performance emerges when powerful models are guided by domain knowledge.

Taken together, these parts shift the goal from producing isolated benchmark artifacts to building a methodology for credible network machine learning. A credible artifact should be diagnosable, trained and evaluated on realistic data, robust to deployment shifts, conscious of privacy constraints, and reproducible enough for others to inspect and extend. As networks become more encrypted, complex, and automated, machine learning becomes useful only if credibility remains the organizing principle for evaluation, data collection, and model design.

Chapter 9

Future Directions

This dissertation argues that credible machine learning for networking requires connecting diagnosis, data collection, and model design. The future directions below therefore focus less on isolated technical extensions and more on research programs that continue this loop in academic networking research: raising standards for publishable ML artifacts, building reusable data-collection ecosystems, and designing context-aware models and data-collection workflows.

1. **Aim toward credibility-centered ML lifecycles for networking.** A central lesson of the dissertation is that credibility cannot be established at a single point in the ML pipeline. Chapter 2 and Chapter 3 show that models can fail silently, Chapter 4 and Chapter 5 show that data collection must respond to those failures, and Chapter 6 shows that model design can internalize the resulting lessons. A natural future direction is to turn this methodology into a full lifecycle for network ML research artifacts. Such a lifecycle would shift the community from one-time benchmark evaluation toward continuous credibility assessment, where papers, datasets, pipelines, and models are checked for obvious failure modes before publication and remain inspectable after release. Concretely, this program should include:

- shortcut-resistant evaluation protocols that make leakage checks, corrected train-test splits, o.o.d. validation, and stress tests routine rather than exceptional;
- intrinsic and extrinsic evaluation suites that connect representation probes, adversarial perturbations, causal protocol interventions, temporal drift, cross-site transfer, and downstream behavior across tasks and datasets;
- reproducible preprocessing artifacts, including feature extraction, flow construction, packet parsing, anonymization, and label-generation code, because these steps often determine what a model can learn;
- practical mechanisms for privacy-preserving data and model sharing, such as secure enclaves, synthetic trace release, anonymized data usage, and carefully designed responsible-access policies; and
- community repositories for tasks, pipelines, datasets, models, and diagnostics, so that researchers can reproduce not only final scores but also the data-collection procedures and credibility checks that produced them.

Within academic networking research, the immediate goal is to raise the standard for publishable ML artifacts: results should be reproducible, evaluations should be auditable, and papers should make it difficult for shortcuts, leakage, undocumented preprocessing, or unverifiable data assumptions to pass unnoticed.

2. **Build shared ecosystems for in-vivo network data collection.** PINOT and netUnicorn demonstrate that realistic data collection can be made programmable, portable, and reusable, but the dissertation also makes clear that no single campus or platform can cover the diversity of networks encountered in practice. A major future direction is to expand this idea into a federated ecosystem of in-vivo data-

collection sites spanning campuses, clouds, enterprise networks, data centers, home networks, mobile environments, and international vantage points. The goal would be to make data diversity a reusable academic resource while preserving local privacy and operational constraints. Concretely, such an ecosystem should include:

- shared experiments, tasks, pipelines, metadata schemas, and responsible-access policies that can be reused across sites;
- tighter integration between passive and active measurements, so that passive observations can suggest active experiments and active experiments can explain ambiguous passive patterns; and
- broader support for heterogeneous measurement targets, including OpenWRT routers, programmable switches, mobile devices, browser clients, IoT systems, and constrained edge nodes.

This would directly address one of the root causes of weak generalization in network ML: the narrowness of available training and evaluation data.

3. **Design context-aware network intelligence.** netFound shows that diagnostics-informed design can produce stronger and more reusable traffic representations, while Chapter 7 shows that general-purpose language models become useful for networking only when their outputs are grounded in reliable network context. Together, these results point to a broader research direction: network intelligence should be designed around the structure, constraints, and interpretive context of networking rather than imported directly from generic ML domains. Future work should include:

   - richer network-native foundation models, including encoder–decoder or decoder-only architectures for generation and forecasting;

- more systematic use of multimodality to enrich model inputs and research datasets, for example by combining packet traces with protocol metadata, application descriptions, topology summaries, labels, textual descriptions, or experiment context; and
- agentic workflows that connect pretrained network embeddings to data collection and analysis, so that models can help propose measurements, select relevant traces, retrieve comparable examples, and guide iterative dataset construction.

This direction should remain tied to the dissertation's central warning: larger models and larger datasets are not enough unless their inputs, objectives, evaluations, and explanations reflect protocol semantics, measurement context, and the structure of network data.

Bibliography

[1] "1998 darpa intrusion detection evaluation dataset." `https://www.ll.mit.edu/r-d/datasets/1998-darpa-intrusion-detection-evaluation-dataset`.

[2] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapé, *Mobile encrypted traffic classification using deep learning*, in *2018 Network Traffic Measurement and Analysis Conference (TMA)*, pp. 1–8, 2018.

[3] Z. Ahmad, A. Shahid Khan, C. Wai Shiang, J. Abdullah, and F. Ahmad, *Network intrusion detection system: A systematic study of machine learning and deep learning approaches*, .

[4] A. Ahmed, R. Mok, and Z. Shafiq, *FlowTrace: A Framework for Active Bandwidth Measurements Using In-band Packet Trains*, *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* **12048 LNCS** (2020) 37–51.

[5] "Apache airflow." `https://airflow.apache.org`.

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

[7] Z. Akhtar, Y. S. Nam, R. Govindan, S. Rao, J. Chen, E. Katz-Bassett, B. Ribeiro, J. Zhan, and H. Zhang, *Oboe: Auto-tuning video ABR algorithms to network conditions*, in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, (New York, NY, USA), pp. 44–58, Association for Computing Machinery, Aug., 2018.

[8] A. Alomar, P. Hamadanian, A. Nasr-Esfahany, A. Agarwal, M. Alizadeh, and D. Shah, *CausalSim: A Causal Framework for Unbiased Trace-Driven Simulation*, *Proceedings of the 20th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2023* (Jan., 2022) 1115–1147, [arXiv:2201.0181].

[9] R. Amin, E. Rojas, A. Aqdus, S. Ramzan, D. Casillas-Perez, and J. M. Arco, *A survey on machine learning techniques for routing optimization in sdn*, *IEEE Access* **9** (2021) 104582–104611.

[10] B. Anderson and D. McGrew, *Machine learning for encrypted malware traffic classification: Accounting for noisy labels and non-stationarity*, in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, (New York, NY, USA), p. 1723–1732, Association for Computing Machinery, 2017.

[11] "Anonymized Two-Way Traffic Packet Header Traces 100G (5 sec) sampler." `https://catalog.caida.org/dataset/passive_100g_sampler`. Dates used: November 2024. Accessed: 01/21/2025.

[12] Anonymous, A. A. Niaki, N. P. Hoang, P. Gill, and A. Houmansadr, *Triplet censors: Demystifying great Firewall's DNS censorship behavior*, in *FOCI*, 2020.

[13] "Ansible automation platform." `https://www.ansible.com/`.

[14] 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** (2020), no. 4 1059–1086, [https://rss.onlinelibrary.wiley.com/doi/pdf/10.1111/rssb.12377].

[15] G. Apruzzese, L. Pajola, and M. Conti, *The Cross-evaluation of Machine Learning-based Network Intrusion Detection Systems*, *IEEE Transactions on Network and Service Management* (2022) 1–1.

[16] S. O. Arik and T. Pfister, *Tabnet: Attentive interpretable tabular learning*, 2020.

[17] M. Arjovsky, L. Bottou, I. Gulrajani, and D. Lopez-Paz, *Invariant Risk Minimization*, *arXiv preprint arXiv:1907.02893* (2020) [arXiv:1907.0289].

[18] D. Arp, E. Quiring, F. Pendlebury, A. Warnecke, F. Pierazzi, C. Wressnegger, L. Cavallaro, and K. Rieck, *Dos and Dont's of Machine Learning in Computer Security*, in *31st USENIX Security Symposium (USENIX Security 22)*, (Boston, MA), USENIX Association, Aug., 2022.

[19] "Ripe atlas." `https://atlas.ripe.net/`.

[20] V. Bajpai, A. Brunstrom, A. Feldmann, W. Kellerer, A. Pras, H. Schulzrinne, G. Smaragdakis, M. Wählisch, and K. Wehrle, *The Dagstuhl Beginners Guide to Reproducibility for Experimental Networking Research*, *SIGCOMM Comput. Commun. Rev.* **49** (Feb., 2019) 24–30.

[21] I. Baldin, A. Nikolich, J. Griffioen, I. I. S. Monga, K.-C. Wang, T. Lehman, and P. Ruth, *FABRIC: A national-scale programmable experimental network infrastructure*, *IEEE Internet Computing* (2019).

[22] "balena - the complete iot management platform." `https://www.balena.io/`.

[23] K. Bartos, M. Sofka, and V. Franc, *Optimized invariant representation of network traffic for detecting unseen malware variants*, in *USENIX Security*, 2016.

[24] O. Bastani, C. Kim, and H. Bastani, *Interpreting Blackbox Models via Model Extraction*, *arXiv preprint arXiv:1705.08504* (2017) [arXiv:1705.0850].

[25] O. Bastani, Y. Pu, and A. Solar-Lezama, *Verifiable Reinforcement Learning via Policy Extraction*, in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, (Red Hook, NY, USA), p. 2499–2509, Curran Associates Inc., 2018.

[26] M. Beck, *On the hourglass model*, *Commun. ACM* **62** (jun, 2019) 48–57.

[27] I. Beltagy, M. E. Peters, and A. Cohan, *Longformer: The long-document transformer*, 2020.

[28] R. Beltiukov, S. Chandrasekaran, A. Gupta, and W. Willinger, *Pinot: Programmable infrastructure for networking*, in *ANRW*, 2023.

[29] R. Beltiukov, W. Guo, A. Gupta, and W. Willinger, *In search of netUnicorn: A data-collection platform to develop generalizable ML models for network security problems*, in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, (Copenhagen, DK), 2023.

[30] R. Beltiukov, S. Guthula, W. Guo, W. Willinger, and A. Gupta, *Demystifying network foundation models*, *Advances in neural information processing systems (NeurIPS)* (2025).

[31] J. Bergstra and Y. Bengio, *Random Search for Hyper-Parameter Optimization*, *J. Mach. Learn. Res.* **13** (feb, 2012) 281–305.

[32] A. Bhaskar and P. Pearce, *Many roads lead to rome: How packet headers influence DNS censorship measurement*, in *USENIX Security*, 2022.

[33] H. Birge-Lee, L. Wang, D. McCarney, R. Shoemaker, J. Rexford, and P. Mittal, *Experiences deploying Multi-Vantage-Point domain validation at let's encrypt*, in *USENIX Security*, 2021.

[34] C. Bothra, J. Gao, S. Rao, and B. Ribeiro, *Veritas: Answering Causal Queries from Video Streaming Traces*, .

[35] 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** (2018), no. 1 16.

[36] G. Bovenzi, F. Cerasuolo, D. Ciuonzo, D. D. Monda, I. Guarino, A. Montieri, V. Persico, and A. Pescapé, *Mapping the Landscape of Generative AI in Network Monitoring and Management*, .

[37] M. Bramer, *Avoiding Overfitting of Decision Trees*, pp. 119–134. Springer London, London, 2007.

[38] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA, 1984.

[39] L. Breiman, *Random forests*, *Machine learning* **45** (2001) 5–32.

[40] 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*, *Proc. ACM Meas. Anal. Comput. Syst.* **3** (dec, 2019).

[41] M. Brundage, S. Avin, J. Wang, *et. al.*, *Toward Trustworthy AI Development: Mechanisms for Supporting Verifiable Claims*, *arXiv preprint arXiv:1705.08504* (2020).

[42] 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** (2020) 132950–132959.

[43] C. Busse-Grawitz, R. Meier, A. Dietmüller, T. Bühler, and L. Vanbever, *pForest: In-Network Inference with Random Forests*, *arXiv preprint arXiv:1909.05680* (2019) [arXiv:1909.0568].

[44] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, *nuScenes: A multimodal dataset for autonomous driving*, *arXiv preprint arXiv:1903.11027* (2019). Available at `https://www.nuscenes.org`.

[45] X. Cai, J. Huang, Y. Bian, and K. Church, *Isotropy in the Contextual Embedding Space: Clusters and Manifolds*, .

[46] "Cloud computing services - amazon web services." `https://aws.amazon.com/`.

[47] "Cloud computing services - digitalocean." `https://www.digitalocean.com/`.

[48] "Cloud computing services - google cloud." `https://cloud.google.com/`.

[49] "Chi@edge." `https://chameleoncloud.org/experiment/chiedge/`.

[50] M. Chang, J. Lambert, P. Sangkloy, J. Singh, S. Bak, A. Hartnett, D. Wang, P. Carr, S. Lucey, D. Ramanan, and J. Hays, *Argoverse: 3D Tracking and Forecasting With Rich Maps*, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June, 2019. Available at `https://www.argoverse.org/`.

[51] E. Chatzoglou, V. Kouliaridis, G. Karopoulos, and G. Kambourakis, *Revisiting quic attacks: A comprehensive review on quic security and a hands-on study*, *International Journal of Information Security* (2022).

[52] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, *SMOTE: Synthetic minority over-sampling technique*, *JAIR* (2002).

[53] "Chef infra." `http://www.chef.io/chef/`.

[54] 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)*, pp. 1271–1276, 2017.

[55] Z. Chen, Q. Li, and Z. Zhang, *Towards robust neural networks via close-loop control*, *arXiv preprint arXiv:2102.01862* (2021).

[56] G. Cherubin, R. Jansen, and C. Troncoso, *Online website fingerprinting: Evaluating website fingerprinting attacks on tor in the real world*, in *USENIX Security*, 2022.

[57] K. Cho, K. Mitsuya, and A. Kato, *Traffic Data Repository at the {WIDE} Project*, .

[58] "Canadian institute for cybersecurity datasets." `https://www.unb.ca/cic/datasets/index.html`.

[59] "Cicflowmeter-v4.0." `https://github.com/ahlashkari/CICFlowMeter`.

[60] Cloudflare, "Internet Speed Test - Measure Network Performance." `https://speed.cloudflare.com/`.

[61] M. W. Craven and J. W. Shavlik, *Extracting Tree-Structured Representations of Trained Networks*, in *Proceedings of the 8th International Conference on Neural Information Processing Systems*, NIPS'95, (Cambridge, MA, USA), p. 24–30, MIT Press, 1995.

[62] A. Cuzzocrea, F. Martinelli, F. Mercaldo, and G. Vercelli, *Tor traffic analysis and detection via machine learning techniques*, in *Big Data*, 2017.

[63] "Dagster." `https://dagster.io/`.

[64] A. D'Amour, K. Heller, D. Moldovan, *et. al.*, *Underspecification Presents Challenges for Credibility in Modern Machine Learning*, *arXiv preprint arXiv:2011.03395* (2020) [arXiv:2011.0339].

[65] J. Daneshamooz, S. Guthula, J. Nguyen, W. Chen, S. Chandrasekaran, A. Gupta, A. Gupta, and W. Willinger, *Netforge: A programmable substrate for bottleneck-centric network data generation*, 2026.

[66] J. Daneshamooz, J. Nguyen, W. Chen, S. Chandrasekaran, S. Guthula, A. Gupta, A. Gupta, and W. Willinger, *Addressing the ml domain adaptation problem for networking: Realistic and controllable training data generation with netreplica*, *arXiv preprint arXiv:2507.13476* (2025).

[67] T. Dao, D. Y. Fu, S. Ermon, A. Rudra, and C. Ré, *Flashattention: Fast and memory-efficient exact attention with io-awareness*, 2022.

[68] J. Deng, W. Dong, R. Socher, L. L. a, K. Li, and L. Fei-Fei, *ImageNet: A large-scale hierarchical image database*, in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009.

[69] A. Dethise, M. Canini, and S. Kandula, *Cracking Open the Black Box: What Observations Can Tell Us About Reinforcement Learning Agents*, in *Proceedings of the 2019 Workshop on Network Meets AI & ML*, NetAI'19, (New York, NY, USA), p. 29–36, Association for Computing Machinery, 2019.

[70] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, *Bert: Pre-training of deep bidirectional transformers for language understanding*, 2019.

[71] Y. Ding, K. Martinkus, D. Pascual, S. Clematide, and R. Wattenhofer, *On Isotropy Calibration of Transformer Models*, in *Proceedings of the Third Workshop on Insights from Negative Results in NLP* (S. Tafreshi, J. Sedoc, A. Rogers, A. Drozd, A. Rumshisky, and A. Akula, eds.), pp. 1–9, Association for Computational Linguistics, 2022.

[72] "Docker." `https://www.docker.com/`.

[73] P. Dodia, M. AlSabah, O. Alrawi, and T. Wang, *Exposing the rat in the tunnel: Using traffic analysis for tor-based malware detection*, in *Ccs*, 2022.

[74] X. L. Dong, S. Moon, Y. E. Xu, K. Malik, and Z. Yu, *Towards next-generation intelligent assistants leveraging llm techniques*, in *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '23, (New York, NY, USA), p. 5792–5793, Association for Computing Machinery, 2023.

[75] R. Doriguzzi-Corin, S. Millar, S. Scott-Hayward, J. M. del Rincón, and D. Siracusa, *Lucid: A Practical, Lightweight Deep Learning Solution for DDoS Attack Detection*, *IEEE Transactions on Network and Service Management* **17** (2020), no. 2 876–889.

[76] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, *An image is worth 16x16 words: Transformers for image recognition at scale*, 2021.

[77] G. Draper-Gil., A. H. Lashkari., M. S. I. Mamun, and A. A. Ghorbani, *Characterization of Encrypted and VPN Traffic using Time-related Features*, in *Proceedings of the 2nd International Conference on Information Systems Security and Privacy - ICISSP,*, pp. 407–414, INSTICC, SciTePress, 2016.

[78] M. Du, F. Li, G. Zheng, and V. Srikumar, *Deeplog: Anomaly detection and diagnosis from system logs through deep learning*, in *CCS*, 2017.

[79] Z. Durumeric, F. Li, J. Kasten, J. Amann, J. Beekman, M. Payer, N. Weaver, D. Adrian, V. Paxson, M. Bailey, and J. A. Halderman, *The Matter of Heartbleed*, in *Proceedings of the 2014 Conference on Internet Measurement Conference*, IMC '14, pp. 475–488, Association for Computing Machinery, 2014.

[80] S. Dwivedi, M. Vardhan, and S. Tripathi, *An effect of chaos grasshopper optimization algorithm for protection of network infrastructure*, *Computer Networks* **176** (2020) 107251.

[81] L. D'hooge, T. Wauters, B. Volckaert, and F. De Turck, *Inter-dataset generalization strength of supervised machine learning methods for intrusion detection*, *Journal of Information Security and Applications* **54** (2020) 102564.

[82] "Edgenet." `https://www.edge-net.org/`.

[83] N. Erickson, J. Mueller, A. Shirkov, H. Zhang, P. Larroy, M. Li, and A. Smola, *AutoGluon-Tabular: Robust and Accurate AutoML for Structured Data*, *arXiv preprint arXiv:2003.06505* (2020).

[84] F. Esposito, D. Malerba, G. Semeraro, and J. Kay, *A comparative analysis of methods for pruning decision trees*, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **19** (1997), no. 5 476–491.

[85] K. Ethayarajh, *How Contextual are Contextualized Word Representations? Comparing the Geometry of BERT, ELMo, and GPT-2 Embeddings*, in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 55–65, Association for Computational Linguistics, 2019.

[86] A. Farahani, S. Voghoei, K. Rasheed, and H. R. Arabnia, *A brief review of domain adaptation*, in *Advances in Data Science and Information Engineering*, 2021.

[87] N. Feamster and J. Rexford, *Why (and how) networks should run themselves*, *CoRR* **abs/1710.11583** (2017) [arXiv:1710.1158].

[88] Federal Communications Commission, "Measuring Broadband America." `https://www.fcc.gov/oet/mba/raw-data-releases`.

[89] T. O. I. S. Foundation., *Project Clearwater*, Jan., 2018.

[90] J. H. Friedman, *Greedy function approximation: A gradient boosting machine.*, *The Annals of Statistics* **29** (2001), no. 5 1189 – 1232.

[91] A. Fuster Baggetto and V. Fresno, *Is anisotropy really the cause of BERT embeddings not being semantic?*, in *Findings of the Association for Computational Linguistics: EMNLP 2022* (Y. Goldberg, Z. Kozareva, and Y. Zhang, eds.), pp. 4271–4281, Association for Computational Linguistics, 2022.

[92] J. Gao, D. He, X. Tan, T. Qin, L. Wang, and T.-Y. Liu, *Representation Degeneration Problem in Training Natural Language Generation Models*, arXiv:1907.1200.

[93] R. Geirhos, J. Jacobsen, C. Michaelis, R. Zemel, W. Brendel, M. Bethge, and F. A. Wichmann, *Shortcut learning in deep neural networks*, *Nature Machine Intelligence* **2** (Nov, 2020) 665–673.

[94] "Gemini Embedding 2: Our first natively multimodal embedding model." `https://blog.google/innovation-and-ai/models-and-research/gemini-models/gemini-embedding-2/`.

[95] A. Gepperth and S. Rieger, *A survey of machine learning applied to computer networks.*, in *ESANN*, 2020.

[96] E. Ghiasvand, S. Ray, S. Iqbal, and S. Dadkhah, *Resilience Against APTs: A Provenance-based IIoT Dataset for Cybersecurity Research*, .

[97] A. Ghorbani, A. Abid, and J. Zou, *Interpretation of Neural Networks Is Fragile*, *Proceedings of the AAAI Conference on Artificial Intelligence* **33** (Jul., 2019) 3681–3688.

[98] "Github actions." `https://docs.github.com/en/actions`.

[99] "Gitlab ci/cd." `https://docs.gitlab.com/ee/ci/`.

[100] "Global Internet traffic growth forecast: Looking forward from 2024." `https://edgeoptic.com/global-internet-traffic-growth-forecast-looking-forward-from-2024/`.

[101] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[102] I. J. Goodfellow, J. Shlens, and C. Szegedy, *Explaining and harnessing adversarial examples*, *arXiv preprint arXiv:1412.6572* (2014).

[103] M. Gouel, K. Vermeulen, M. Mouchet, J. P. Rohrer, O. Fourmaux, and T. Friedman, *Zeph iris map the internet: A resilient reinforcement learning approach to distributed ip route tracing*, *SIGCOMM Computer Communication Review* (2022).

[104] L. Grinsztajn, E. Oyallon, and G. Varoquaux, *Why do tree-based models still outperform deep learning on tabular data?*, 2022.

[105] R. Guidotti and R. Ruggieri, *On The Stability of Interpretable Models*, in *2019 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, 2019.

[106] W. Guo, D. Mu, J. Xu, P. Su, G. Wang, and X. Xing, *LEMNA: Explaining Deep Learning Based Security Applications*, in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, (New York, NY, USA), p. 364–379, Association for Computing Machinery, 2018.

[107] A. Gupta, C. Mac-Stoker, and W. Willinger, *An effort to democratize networking research in the era of ai/ml*, in *Proceedings of the 18th ACM Workshop on Hot Topics in Networks*, HotNets '19, (New York, NY, USA), p. 93–100, Association for Computing Machinery, 2019.

[108] S. Gupta and A. Gupta, *Dealing with noise problem in machine learning data-sets: A systematic review*, *Procedia Computer Science* (2019).

[109] S. Guthula, R. Beltiukov, N. Battula, W. Guo, A. Gupta, and I. Monga, *netFound: Foundation Model for Network Security*, arXiv:2310.1702.

[110] S. Guthula, J. Daneshamooz, C. Fleming, A. Kundu, W. Willinger, and A. Gupta, *Netburst: Event-centric forecasting of bursty, intermittent time series*, 2025.

[111] C. Gutterman, K. Guo, S. Arora, T. Gilliland, X. Wang, L. Wu, E. Katz-Bassett, and G. Zussman, *Requet: Real-time QoE metric detection for encrypted YouTube traffic*, *ACM Transactions on MCCA* (2020).

[112] Z. Hang, Y. Lu, Y. Wang, and Y. Xie, *Flow-MAE: Leveraging masked AutoEncoder for accurate, efficient and robust malicious traffic classification*, in *Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses*, Raid '23, (New York, NY, USA), pp. 297–314, Association for Computing Machinery, 2023.

[113] M. Harrity, K. Bock, F. Sell, and D. Levin, *GET /out: Automated discovery of Application-Layer censorship evasion strategies*, in *USENIX Security*, 2022.

[114] H. Y. He, Z. Guo Yang, and X. N. Chen, *PERT: Payload encoding representation from transformer for encrypted traffic classification*, in *2020 ITU Kaleidoscope: Industry-driven Digital Transformation (ITU K)*, pp. 1–8, 2020.

[115] K. He, X. Chen, S. Xie, Y. Li, P. Dollár, and R. Girshick, *Masked autoencoders are scalable vision learners*, 2021.

[116] Z. He, B. Bhasuran, Q. Jin, S. Tian, K. Hanna, C. Shavor, L. G. Arguello, P. Murray, and Z. Lu, *Quality of answers of generative large language models versus peer users for interpreting laboratory test results for lay patients: Evaluation study*, *J Med Internet Res* **26** (Apr, 2024) e56655.

[117] D. Hendrycks and K. Gimpel, *A baseline for detecting misclassified and out-of-distribution examples in neural networks*, *arXiv:1610.02136* (2016).

[118] J. Holland, P. Schmitt, N. Feamster, and P. Mittal, *New Directions in Automated Traffic Analysis*, in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, CCS '21, (New York, NY, USA), p. 3366–3383, Association for Computing Machinery, 2021.

[119] J. Huang, D. Tang, W. Zhong, S. Lu, L. Shou, M. Gong, D. Jiang, and N. Duan, *WhiteningBERT: An Easy Unsupervised Sentence Embedding Approach*, arXiv:2104.0176.

[120] A. Hussein, M. G. Medhat, E. Elyan, and C. Jayne, *Imitation Learning: A Survey of Learning Methods*, *ACM Comput. Surv.* **50** (Apr., 2017).

[121] "Hydra." `https://github.com/vanhauser-thc/thc-hydra`.

[122] K. Hämmerl, A. Fastowski, J. Libovický, and A. Fraser, *Exploring Anisotropy and Outliers in Multilingual Language Models for Cross-Lingual Semantic Sentence Similarity*, .

[123] IP API, "IP-API.com - Geolocation API." `https://ip-api.com/`.

[124] A. S. Jacobs, R. Beltiukov, W. Willinger, R. A. Ferreira, A. Gupta, and L. Z. Granville, *AI/ML and Network Security: The Emperor has no Clothes (Extended Version). Technical Report*, May, 2022. Available at `https://github.com/TrusteeML/emperor`.

[125] A. S. Jacobs, R. Beltiukov, W. Willinger, R. A. Ferreira, A. Gupta, and L. Z. Granville, *Paper supplemental material*, May, 2022. Reproducibility artifacts available at `https://github.com/TrusteeML/emperor`. The Trustee framework was implemented in a separate Python library for ease of use, available at `https://github.com/TrusteeML/trustee`.

[126] A. S. Jacobs, R. Beltiukov, W. Willinger, R. A. Ferreira, A. Gupta, and L. Z. Granville, *AI/ML for network security: The emperor has no clothes*, in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2022.

[127] "Jenkins." `https://www.jenkins.io/`.

[128] X. Jiang, S. Liu, A. Gember-Jacobson, A. N. Bhagoji, P. Schmitt, F. Bronzino, and N. Feamster, *NetDiffusion: Network Data Augmentation Through Protocol-Constrained Traffic Generation*, arXiv:2310.0854.

[129] X. Jiang, S. Liu, S. Naama, F. Bronzino, P. Schmitt, and N. Feamster, *AC-DC: Adaptive Ensemble Classification for Network Traffic Identification*, arXiv:2302.1171.

[130] 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 *USENIX Security*, 2017.

[131] H. Kaur, H. S. Pannu, and A. K. Malhi, *A Systematic Review on Imbalanced Data Challenges in Machine Learning: Applications and Solutions*, *ACM Comput. Surv.* **52** (aug, 2019).

[132] J. Kim, Y. Jung, H. Yeo, J. Ye, and D. Han, *Neural-enhanced live streaming: Improving live video ingest via online learning*, in *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '20, (New York, NY, USA), p. 107–125, Association for Computing Machinery, 2020.

[133] S. Kornblith, M. Norouzi, H. Lee, and G. Hinton, *Similarity of Neural Network Representations Revisited*, arXiv:1905.0041.

[134] G. Kovács, *An empirical comparison and evaluation of minority oversampling techniques on a large number of imbalanced datasets*, *ASC* (2019).

[135] "Kubernetes - production-grade container orchestraction." `https://kubernetes.io/`.

[136] I. Kunakorntum, W. Hinthong, and P. Phunchongharn, *A synthetic minority based on probabilistic distribution (symprod) oversampling for imbalanced datasets*, *IEEE Access* (2020).

[137] H. Lakkaraju and O. Bastani, *"How Do I Fool You?": Manipulating User Trust via Misleading Black Box Explanations*, in *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*, AIES '20, (New York, NY, USA), p. 79–85, Association for Computing Machinery, 2020.

[138] H. Lakkaraju, E. Kamar, R. Caruana, and J. Leskovec, *Interpretable & Explorable Approximations of Black Box Models*, *arXiv preprint arXiv:1707.01154* (2017) [arXiv:1707.0115].

[139] B. Lantz, B. Heller, and N. McKeown, *A Network in a Laptop: Rapid Prototyping for Software-defined Networks*, in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets-IX, (New York, NY, USA), pp. 19:1–19:6, ACM, 2010.

[140] A. H. Lashkari, G. D. Gil, M. S. I. Mamun, and A. A. Ghorbani, *Characterization of Tor Traffic using Time based Features*, .

[141] G. Lemaître, F. Nogueira, and C. K. Aridas, *Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning*, *Journal of Machine Learning Research* **18** (2017), no. 17 1–5.

[142] B. Li, H. Zhou, J. He, M. Wang, Y. Yang, and L. Li, *On the Sentence Embeddings from Pre-trained Language Models*, in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (B. Webber, T. Cohn, Y. He, and Y. Liu, eds.), pp. 9119–9130, Association for Computational Linguistics, 2020.

[143] Y. Li, H. Liu, W. Yang, D. Hu, and W. Xu, *Inter-data-center network traffic prediction with elephant flows*, in *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, pp. 206–213, 2016.

[144] X. Lin, G. Xiong, G. Gou, Z. Li, J. Shi, and J. Yu, *ET-BERT: A contextualized datagram representation with pre-training transformers for encrypted traffic classification*, in *Proceedings of the ACM Web Conference 2022*, Www '22, (New York, NY, USA), pp. 633–642, Association for Computing Machinery, 2022.

[145] Z. C. Lipton, *The Mythos of Model Interpretability: In Machine Learning, the Concept of Interpretability is Both Important and Slippery.*, *Queue* **16** (June, 2018) 31–57.

[146] C. Liu, L. He, G. Xiong, Z. Cao, and Z. Li, *Fs-net: A flow sequence network for encrypted traffic classification*, in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, pp. 1171–1179, 2019.

[147] J. Liu, D. Lerner, J. Chung, U. Paul, A. Gupta, and E. Belding, *Watching Stars in Pixels: The Interplay Of Traffic Shaping and YouTube Streaming QoE over GEO Satellite Networks*, in *Passive and Active Measurement* (P. Richter, V. Bajpai, and E. Carisimo, eds.), vol. 14538, pp. 153–169. Springer Nature Switzerland, 2024.

[148] L. Liu, G. Engelen, T. Lynar, D. Essam, and W. Joosen, *Error prevalence in NIDS datasets: A case study on CIC-IDS-2017 and CSE-CIC-IDS-2018*, in *2022 IEEE Conference on Communications and Network Security (CNS)*, pp. 254–262, 2022.

[149] LM Studio, "LM Studio - Discorver, download, and run local LLMs." `https://lmstudio.ai/`.

[150] I. Loshchilov and F. Hutter, *Decoupled weight decay regularization*, 2019.

[151] 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* (2018).

[152] "Luigi." `https://github.com/spotify/luigi`.

[153] S. M. Lundberg and S. Lee, *A Unified Approach to Interpreting Model Predictions*, in *Advances in Neural Information Processing Systems 30* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), pp. 4765–4774. Curran Associates, Inc., 2017.

[154] K. Macmillan, T. Mangla, J. Saxon, and N. Feamster, *Measuring the performance and network utilization of popular video conferencing applications*, *Proceedings of the ACM SIGCOMM Internet Measurement Conference, IMC* (Nov., 2021) 229–244, [arXiv:2105.1347].

[155] K. Maharana, S. Mondal, and B. Nemade, *A review: Data pre-processing and data augmentation techniques*, *Global Transitions Proceedings* (2022).

[156] A. Mahimkar, A. Sivakumar, Z. Ge, S. Pathak, and K. Biswas, *Auric: Using data-driven recommendation to automatically generate cellular configuration*, in *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, SIGCOMM '21, (New York, NY, USA), p. 807–820, Association for Computing Machinery, 2021.

[157] A. Mani *et. al.*, *Towards llm-powered network management*, in *Proceedings of ACM HotNets*, 2023.

[158] H. Mao, R. Netravali, and M. Alizadeh, *Neural Adaptive Video Streaming with Pensieve*, in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, (New York, NY, USA), pp. 197–210, Association for Computing Machinery, Aug., 2017.

[159] Measurement Lab, "The M-Lab NDT data set." `https://measurementlab.net/tests/ndt`, (2009-02-11 – 2025-04-13).

[160] "memory-profiler." `https://pypi.org/project/memory-profiler/`.

[161] X. Meng, C. Lin, Y. Wang, and Y. Zhang, *NetGPT: Generative Pretrained Transformer for Network Traffic*, arXiv:2304.0951.

[162] Z. Meng, M. Wang, J. Bai, M. Xu, H. Mao, and H. Hu, *Interpreting Deep Learning-Based Networking Systems*, in *Proceedings of the Annual Conference of the ACM SIGCOMM*, SIGCOMM '20, (New York, NY, USA), p. 154–171, Association for Computing Machinery, 2020.

[163] A. Mestres, A. Rodriguez-Natal, J. Carner, P. Barlet-Ros, E. Alarcón, M. Solé, J. Rexford, and A. Cabellos-Aparicio, *Knowledge-defined networking*, *ACM SIGCOMM Computer Communication Review* **47** (2017), no. 3 2–10.

[164] "Microsoft azure - cloud computing services." `https://azure.microsoft.com/`.

[165] B. Miller, L. Huang, A. D. Joseph, and J. D. Tygar, *I know why you went to the clinic: Risks and realization of https traffic analysis*, in *Privacy Enhancing Technologies* (E. De Cristofaro and S. J. Murdoch, eds.), (Cham), pp. 143–163, Springer International Publishing, 2014.

[166] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, *Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection*, in *Network and Distributed System Security Symposium 2018*, NDSS'18, 2018.

[167] F. Molder, K. Jablonski, B. Letcher, M. Hall, C. Tomkins-Tinch, V. Sochat, J. Forster, S. Lee, S. Twardziok, A. Kanitz, A. Wilm, M. Holtgrewe, S. Rahmann, S. Nahnsen, and J. Koster, *Sustainable data analysis with snakemake*, *F1000Research* **10** (2021), no. 33.

[168] C. Molnar, G. König, J. Herbinger, T. Freiesleben, S. Dandl, C. A. Scholbeck, G. Casalicchio, M. Grosse-Wentrup, and B. Bischl, *Pitfalls to Avoid when Interpreting Machine Learning Models*, *arXiv preprint arXiv:2007.04131* (2020) [arXiv:2007.0413].

[169] C. Molnar, *Interpretable machine learning*. Lulu. com, 2020.

[170] A. Natekin and A. Knoll, *Gradient boosting machines, a tutorial*, *Frontiers in neurorobotics* **7** (2013) 21.

[171] National Academies of Sciences Engineering and Medicine, *Implications of Artificial Intelligence for Cybersecurity: Proceedings of a Workshop*. The National Academies Press, Washington, DC, 2019.

[172] National Science Foundation: Workshop on Self-Driving Networks.

[173] R. Netravali, A. Sivaraman, S. Das, A. Goyal, K. Winstein, J. Mickens, and H. Balakrishnan, *Mahimahi: Accurate Record-and-Replay for HTTP*, in *USENIX ATC*, 2015.

[174] "Netrics." `https://github.com/chicago-cdac/nm-exp-active-netrics`.

[175] "System code of netunicorn." `https://github.com/netunicorn/netunicorn`.

[176] "Library of tasks for netunicorn." `https://github.com/netunicorn/netunicorn-library`.

[177] "Supplementary materials for netunicorn paper." https://github.com/netunicorn/netunicorn-search.

[178] H. Nori, S. Jenkins, P. Koch, and R. Caruana, *Interpretml: A unified framework for machine learning interpretability*, *arXiv preprint arXiv:1909.09223* (2019).

[179] B. A. Nosek, G. Alter, G. C. Banks, *et. al.*, *Promoting an open research culture*, *Science* **348** (2015), no. 6242 1422–1425, [https://www.science.org/doi/pdf/10.1126/science.aab2374].

[180] "ns-3 | a discrete-event network simulator for internet systems." https://www.nsnam.org/.

[181] Ookla, "Speedtest by Ookla - The Global Broadband Speed Test." https://www.speedtest.net/.

[182] "P4runtime specification." https://p4.org/p4-spec/p4runtime/main/P4Runtime-Spec.html.

[183] A. Panchenko, F. Lanze, J. Pennekamp, T. Engel, A. Zinnen, M. Henze, and K. Wehrle, *Website fingerprinting at internet scale.*, in *NDSS*, 2016.

[184] "Patator." https://github.com/lanjelot/patator.

[185] U. Paul, J. Liu, M. Gu, A. Gupta, and E. Belding, *The importance of contextualization of crowdsourced active speed test measurements*, in *Proceedings of the 22nd ACM Internet Measurement Conference*, IMC '22, (New York, NY, USA), p. 274–289, Association for Computing Machinery, 2022.

[186] "Platforms for advanced wireless research." https://advancedwireless.org/.

[187] "Pcapplusplus - a multiplatform c++ library for capturing, parsing and crafting of network packets.." https://github.com/seladb/PcapPlusPlus.

[188] J. Petch, S. Di, and W. Nelson, *Opening the black box: The promise and limitations of explainable machine learning in cardiology*, *Canadian Journal of Cardiology* (2022).

[189] M. Peuster, H. Karl, and S. van Rossem, *MeDICINE: Rapid prototyping of production-ready network services in multi-PoP environments*, in *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pp. 148–153, Nov., 2016.

[190] "Pinot swagger." https://pinot.cs.ucsb.edu/api/v1/info/swagger/#/.

[191] "Pinot project website." https://pinot.cs.ucsb.edu.

[192] "Puppet." `https://puppet.com/`.

[193] C. Qian, X. Li, Q. Wang, G. Zhou, and H. Shao, *NetBench: A Large-Scale and Comprehensive Network Traffic Benchmark Dataset for Foundation Models*, Mar., 2024.

[194] J. Qu, X. Ma, and J. Li, *TrafficGPT: Breaking the Token Barrier for Efficient Long Traffic Analysis and Generation*, arXiv:2403.0582.

[195] J. Quinonero-Candela, M. Sugiyama, A. Schwaighofer, and N. D. Lawrence, *Dataset shift in machine learning*. Mit Press, 2008.

[196] S.-A. Rebuffi, S. Gowal, D. A. Calian, F. Stimberg, O. Wiles, and T. A. Mann, *Data augmentation can improve robustness*, in *NeurIPS*, 2021.

[197] J. Ren, D. J. Dubois, and D. Choffnes, *An International View of Privacy Risks for Mobile Apps*, .

[198] "The CAIDA Anonymized Internet Traces 2016 Dataset." `https://www.caida.org/data/passive/passive_2016_dataset.xml`.

[199] M. Ribeiro, S. Singh, and C. Guestrin, *"Why Should I Trust You?": Explaining the Predictions of Any Classifier*, in *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, (San Diego, California), pp. 97–101, Association for Computational Linguistics, June, 2016.

[200] M. Richards, *Software Architecture Patterns: Understanding Common Architecture Patterns and when to Use Them*. O'Reilly Media, 2015.

[201] H. Riiser, P. Vigmostad, C. Griwodz, and P. Halvorsen, *Commute path bandwidth traces from 3G networks: analysis and applications*, in *Proceedings of the 4th ACM Multimedia Systems Conference*, pp. 114–118, 2013.

[202] J. Rivero, B. Ribeiro, N. Chen, and F. S. Leite, *A grassmannian approach to zero-shot learning for network intrusion detection*, in *ICONIP*, 2017.

[203] S. Ross, G. J. Gordon, and J. A. Bagnell, *A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning*, *arXiv preprint arXiv:1011.0686* (2011) [arXiv:1011.0686].

[204] T. Roth, Y. Gao, A. Abuadbba, S. Nepal, and W. Liu, *Token-modification adversarial attacks for natural language processing: A survey*, *AI Communications* **37** (2024), no. 4 655–676, [https://journals.sagepub.com/doi/pdf/10.3233/AIC-230279].

[205] S. Ruder, *An overview of multi-task learning in deep neural networks*, *arXiv:1706.05098* (2017).

[206] C. Rudin, *Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead*, *Nature Machine Intelligence* **1** (May, 2019) 206–215.

[207] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Pearson, USA, 4rd ed., 2020.

[208] "Salt project." `https://saltproject.io/`.

[209] R. Schuster, V. Shmatikov, and E. Tromer, *Beauty and the burst: Remote identification of encrypted video streams*, in *Proceedings of the 26th USENIX Conference on Security Symposium*, SEC'17, (USA), p. 1357–1374, USENIX Association, 2017.

[210] "Seclists." `https://github.com/danielmiessler/SecLists`.

[211] M. Shafi, A. H. Lashkari, and A. H. Roudsari, *NTLFlowLyzer: Towards generating an intrusion detection dataset and intruders behavior profiling through network and transport layers traffic analysis and pattern extraction*, .

[212] S. Shankar, V. Piratla, S. Chakrabarti, S. Chaudhuri, P. Jyothi, and S. Sarawagi, *Generalizing across domains via cross-gradient training*, *arXiv preprint arXiv:1804.10745* (2018).

[213] L. S. Shapley, *17. A Value for n-Person Games*, pp. 307–318. Princeton University Press, 2016.

[214] I. Sharafaldin., A. H. Lashkari., and A. A. Ghorbani., *Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization*, in *Proceedings of the 4th International Conference on Information Systems Security and Privacy - ICISSP,*, pp. 108–116, INSTICC, SciTePress, 2018.

[215] K. Shaukat, S. Luo, V. Varadharajan, I. A. Hameed, and M. Xu, *A Survey on Machine Learning Techniques for Cyber Security in the Last Decade*, *IEEE Access* **8** (2020) 222310–222354.

[216] N. Shazeer, *Glu variants improve transformer*, 2020.

[217] S. Shi, X. Zhang, and W. Fan, *Explaining the predictions of any image classifier via decision trees*, 2019.

[218] Y. Shi and S. Biswas, *Protocol-independent identification of encrypted video traffic sources using traffic analysis*, in *2016 IEEE International Conference on Communications (ICC)*, pp. 1–6, 2016.

[219] C. Shorten and T. M. Khoshgoftaar, *A survey on image data augmentation for deep learning*, *Journal of big data* (2019).

[220] R. Shwartz-Ziv and A. Armon, *Tabular data: Deep learning is not all you need*, *Information Fusion* (2022).

[221] "Sidecar pattern." `https://docs.microsoft.com/en-us/azure/architecture/patterns/sidecar`.

[222] A. Sivanathan, H. H. Gharakheili, F. Loi, A. Radford, C. Wijenayake, A. Vishwanath, and V. Sivaraman, *Classifying IoT Devices in Smart Environments Using Network Traffic Characteristics*, *IEEE Transactions on Mobile Computing* **18** (2019), no. 8 1745–1759.

[223] J.-P. Smith, L. Dolfi, P. Mittal, and A. Perrig, *QCSD: A QUIC Client-Side Website-Fingerprinting defence framework*, in *USENIX Security*, 2022.

[224] J. Snoek, H. Larochelle, and R. P. Adams, *Practical Bayesian Optimization of Machine Learning Algorithms*, in *Advances in Neural Information Processing Systems* (F. Pereira, C. Burges, L. Bottou, and K. Weinberger, eds.), vol. 25, Curran Associates, Inc., 2012.

[225] 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*, pp. 305–316, 2010.

[226] H. Steck, C. Ekanadham, and N. Kallus, *Is cosine-similarity of embeddings really about similarity?*, in *Companion Proceedings of the ACM Web Conference 2024*, WWW '24, p. 887–890, ACM, May, 2024.

[227] J. Su, J. Cao, W. Liu, and Y. Ou, *Whitening Sentence Representations for Better Semantics and Faster Retrieval*, arXiv:2103.1531.

[228] J. Su, Y. Lu, S. Pan, A. Murtadha, B. Wen, and Y. Liu, *Roformer: Enhanced transformer with rotary position embedding*, 2023.

[229] T. Swamy, A. Rucker, M. Shahbaz, I. Gaur, and K. Olukotun, *Taurus: A Data Plane Architecture for Per-Packet ML; Taurus: A Data Plane Architecture for Per-Packet ML*, .

[230] "T5 Pretraining Results." `https://docs.nvidia.com/nemo-framework/user-guide/24.07/llms/t5/performance.html`.

[231] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, *Llama: Open and efficient foundation language models*, 2023.

[232] C.-F. Tsai, Y.-F. Hsu, C.-Y. Lin, and W.-Y. Lin, *Intrusion detection by machine learning: A review*, *Expert Systems with Applications* **36** (2009), no. 10 11994–12000.

[233] M. Turek, 2016. Available at `https://www.darpa.mil/program/explainable-artificial-intelligence`. Accessed on May 25th, 2021.

[234] P. Turney, *Technical Note: Bias and the Quantification of Stability*, *Machine Learning* **20** (1995), no. 1 23–33.

[235] "Unsw datasets." `https://iotanalytics.unsw.edu.au/`.

[236] D. A. Van Dyk and X.-L. Meng, *The art of data augmentation*, *Journal of Computational and Graphical Statistics* (2001).

[237] T. van Ede, R. Bortolameotti, A. Continella, J. Ren, D. J. Dubois, M. Lindorfer, D. R. Choffnes, M. van Steen, and A. Peter, *Flowprint: Semi-supervised mobile-app fingerprinting on encrypted network traffic*, *Proceedings 2020 Network and Distributed System Security Symposium* (2020).

[238] M. Vasić, A. Petrović, K. Wang, M. Nikolić, R. Singh, and S. Khurshid, *MoËT: Mixture of expert trees and its application to verifiable reinforcement learning*, *Neural Networks* **151** (jul, 2022) 34–47.

[239] "Vmware vsphere." `https://www.vmware.com/products/vsphere.html`.

[240] C. Wang, M. Scazzariello, A. Farshin, S. Ferlin, D. Kostić, and M. Chiesa, *NetConfEval: Can LLMs Facilitate Network Configuration?*, .

[241] Q. Wang, C. Qian, X. Li, Z. Yao, G. Zhou, and H. Shao, *Lens: A Foundation Model for Network Traffic*, .

[242] T. Wang, X. Xie, W. Wang, C. Wang, Y. Zhao, and Y. Cui, "NetMamba: Efficient Network Traffic Classification via Pre-training Unidirectional Mamba." https://arxiv.org/abs/2405.11449v4, May, 2024.

[243] W. Wang, M. Zhu, J. Wang, X. Zeng, and Z. Yang, *End-to-end encrypted traffic classification with one-dimensional convolution neural networks*, in *2017 IEEE International Conference on Intelligence and Security Informatics (ISI)*, pp. 43–48, 2017. Code available at GitHub repository `https://github.com/echowei/DeepTraffic`.

[244] W. Wang, M. Zhu, X. Zeng, X. Ye, and Y. Sheng, *Malware traffic classification using convolutional neural network for representation learning*, in *2017 International Conference on Information Networking (ICOIN)*, pp. 712–717, 2017.

[245] B. Warner, A. Chaffin, B. Clavié, O. Weller, O. Hallström, S. Taghadouini, A. Gallagher, R. Biswas, F. Ladhak, T. Aarsen, N. Cooper, G. Adams, J. Howard, and I. Poli, *Smarter, better, faster, longer: A modern bidirectional encoder for fast, memory efficient, and long context finetuning and inference*, 2024.

[246] F. Wei, H. Li, Z. Zhao, and H. Hu, *Xnids: Explaining deep learning-based network intrusion detection systems for active intrusion responses*, in *Security*, 2023.

[247] N. Wickramasinghe, A. Shaghaghi, G. Tsudik, and S. Jha, *SoK: Decoding the Enigma of Encrypted Network Traffic Classifiers*, .

[248] W. Willinger, R. A. Ferreira, A. Gupta, R. S. Beltiukov, S. Guthula, L. Z. Granville, and A. S. Jacobs, *When something looks too good to be true, it usually is! ai is causing a credibility crisis in networking*, *SIGCOMM Comput. Commun. Rev.* **55** (Apr., 2025) 10–15.

[249] W. Willinger, A. Gupta, R. Beltiukov, and W. Guo, *A netai manifesto (part ii): Less hubris, more humility*, *SIGMETRICS Perform. Eval. Rev.* **51** (Oct., 2023) 109–111.

[250] W. Willinger, A. Gupta, A. S. Jacobs, R. Beltiukov, R. A. Ferreira, and L. Granville, *A netai manifesto (part i): Less explorimentation, more science*, *SIGMETRICS Perform. Eval. Rev.* **51** (Oct., 2023) 106–108.

[251] D. Wu *et. al.*, *NetLLM: Adapting large language models for networking*, in *Proceedings of ACM SIGCOMM*, 2024.

[252] Y. Xin, L. Kong, Z. Liu, Y. Chen, Y. Li, H. Zhu, M. Gao, H. Hou, and C. Wang, *Machine Learning and Deep Learning Methods for Cybersecurity*, *IEEE Access* **6** (2018) 35365–35381.

[253] Z. Xiong and N. Zilberman, *Do Switches Dream of Machine Learning? Toward In-Network Classification*, in *Proceedings of the 18th ACM Workshop on Hot Topics in Networks*, HotNets '19, (New York, NY, USA), p. 25–33, Association for Computing Machinery, 2019.

[254] F. Y. Yan, H. Ayers, C. Zhu, S. Fouladi, J. Hong, K. Zhang, P. Levis, and K. Winstein, *Learning in situ: A randomized experiment in video streaming*, in *NSDI*, 2020.

[255] F. Y. Yan, J. Ma, G. D. Hill, D. Raghavan, R. S. Wahby, P. Levis, and K. Winstein, *Pantheon: The training ground for Internet congestion-control research*, in *Usenix Atc*, 2018.

[256] J. Yang, X. Jiang, Y. Lei, W. Liang, Z. Ma, and S. Li, *Mtsecurity: Privacy-preserving malicious traffic classification using graph neural network and transformer*, *IEEE Transactions on Network and Service Management* (2024) 1–1.

[257] 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 *USENIX Security*, 2021.

[258] T. Yang, Y. Jin, Y. Chen, and Y. Jin, *RT-WABest: A novel end-To-end bandwidth estimation tool in IEEE 802.11 wireless network*, *International Journal of Distributed Sensor Networks* **13** (Feb., 2017).

[259] H. Zhang, L. Huang, C. Q. Wu, and Z. Li, *An effective convolutional neural network based on SMOTE and Gaussian mixture model for intrusion detection in imbalanced dataset*, *Computer Networks* **177** (2020) 107315.

[260] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, *mixup: Beyond empirical risk minimization*, *arXiv preprint arXiv:1710.09412* (2017).

[261] T. Zhang, F. Ladhak, E. Durmus, P. Liang, K. McKeown, and T. B. Hashimoto, *Benchmarking large language models for news summarization*, *Transactions of the Association for Computational Linguistics* **12** (01, 2024) 39–57, [https://direct.mit.edu/tacl/article-pdf/doi/10.1162/tacl_a_00632/2325685/tacl_a_00632.pdf].

[262] Y. Zhang and Q. Yang, *An overview of multi-task learning*, *NSR* (2018).

[263] R. Zhao, X. Deng, Z. Yan, J. Ma, Z. Xue, and Y. Wang, *MT-FlowFormer: A semi-supervised flow transformer for encrypted traffic classification*, in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, Kdd '22, (New York, NY, USA), pp. 2576–2584, Association for Computing Machinery, 2022.

[264] R. Zhao, M. Zhan, X. Deng, Y. Wang, Y. Wang, G. Gui, and Z. Xue, *Yet another traffic classifier: A masked autoencoder based traffic transformer with multi-level flow representation*, *Proceedings of the AAAI Conference on Artificial Intelligence* **37** (June, 2023) 5420–5427.

[265] Y. Zhao, G. Dettori, M. Boffa, L. Vassio, and M. Mellia, *The sweet danger of sugar: Debunking representation learning for encrypted traffic classification*, in *Proceedings of the ACM SIGCOMM 2025 Conference*, SIGCOMM '25, p. 296–310, ACM, Aug., 2025.

[266] G. Zhou, X. Guo, Z. Liu, T. Li, Q. Li, and K. Xu, *Trafficformer: An efficient pre-trained model for traffic data*, in *2025 IEEE Symposium on Security and Privacy (SP)*, pp. 1844–1860, 2025.

[267] Q. Zhou and D. Pezaros, *Evaluation of machine learning classifiers for zero-day intrusion detection–an analysis on CIC-AWS-2018 dataset*, *arXiv preprint arXiv:1905.03685* (2019) [arXiv:1905.0368].