UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

ARTHUR SELLE JACOBS

Enabling Self-Driving Networks with Machine Learning

Thesis presented in partial fulfillment of the requirements for the degree of Doctor of Computer Science

Advisor: Prof. Dr. Lisandro Zambenedetti Granville
Coadvisor: Prof. Dr. Ronaldo Alves Ferreira

Porto Alegre
December 2022

CIP — CATALOGING-IN-PUBLICATION

Selle Jacobs, Arthur

Enabling Self-Driving Networks with Machine Learning / Arthur Selle Jacobs. – Porto Alegre: PPGC da UFRGS, 2022.

146 f.: il.

Thesis (Ph.D.) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2022. Advisor: Lisandro Zambenedetti Granville; Coadvisor: Ronaldo Alves Ferreira.

1. Self-Driving Networks. 2. Machine Learning. 3. Explainability. 4. Intent-Based Networking. 5. Operator Feedback. 6. Network Security. I. Zambenedetti Granville, Lisandro. II. Alves Ferreira, Ronaldo. III. Título.

ABSTRACT

As modern networks grow in size and complexity, they also become increasingly prone to human errors. This trend has driven both industry and academia to try to automate management and control tasks, aiming to reduce human interaction with the network and human-made mistakes. Ideally, researchers envision a network design that is not only automatic (*i.e.,* dependent of human instructions) but autonomous (*i.e.,* capable of making its own decisions). Autonomous networking has been a goal sought for years, with many different concepts, designs and implementations, but it was never fully realized, mainly due to technological limitations. Recent advances in Artificial Intelligence (AI) and Machine Learning (ML) introduced a breath of fresh air into this concept, reemerging as the re-branded concept of *self-driving networks*, in view of its autonomous car counterparts. In broad terms, a self-driving network is an autonomous network capable of acting according to high-level intents from an operator and automatically adapting to changes in traffic and user behavior. To achieve that vision, a network would need to fulfill four major requirements: *(i)* understand high-level intents from an operator to dictate its behavior, *(ii)* monitor itself based on input intents, *(iii)* predict and identify patterns from monitored data and *(iv)* adapt itself to new behaviors without the intervention of an operator.

As fulfilling the requirements of a self-driving network requires heavily relying on ML models to make decisions and classifications that directly impact the network, one particular issue becomes prominent with this design: trust. Applying ML to solve networking management tasks, such as the ones described above, has been a popular trend among researchers recently. However, despite the topic receiving much attention, industry operators have been reluctant to take advantage of such solutions, mainly because of the black-box nature of ML models which produce decisions without any explanation or reason as to which those decisions were made. Given the high-stakes nature of production networks, it becomes impossible to trust a ML model that may take system-breaking actions automatically, and most important to the scope of this thesis, a prohibitive challenge that must be addressed if a self-driving network design is ever to be achieved.

The present thesis aims to enable self-driving networks by tackling the problem of the inherent lack of trust in ML models that empower it. To that end, we assess and scrutinize the decision-making process of ML-based classifiers used to compose a self-driving network. First, we investigate and evaluate the accuracy and credibility of classifications made by ML models used to process high-level intents from the operator. For that eval-

uation, we propose a novel conversational interface called LUMI that allows operators to use natural language to describe how the network should behave. Second, we analyze and assess the accuracy and credibility of ML models' decisions to self-configure the network according to monitored data. In that analysis, we uncover the need to reinvent how researchers apply AI/ML to networking problems, so we propose a new AI/ML pipeline that introduces steps to scrutinize ML models using techniques from the emerging field of eXplainable AI (XAI). Finally, we investigate if there is a viable method to improve the trust of operators in the decision made by ML models that enable self-driving networks. Our investigation led us to propose a new XAI method to extract explanations from any given black-box ML model in the form of decision trees while maintaining a manageable size, which we called TRUSTEE. Our results show that ML models widely applied to solve networking problems have not been put under proper scrutiny and can easily break when put under real-world traffic. Such models, therefore, need to be corrected to fulfill their given tasks properly.

Keywords: Self-Driving Networks. Machine Learning. Explainability. Intent-Based Networking. Operator Feedback. Network Security.

Aprendizado de Máquina para Redes Autodirigidas

RESUMO

Conforme as redes modernas crescem em tamanho e complexidade, elas também se tornam cada vez mais sujeitas a erros humanos. Essa tendência tem levado a indústria e o meio acadêmico a tentar automatizar as tarefas de gestão e controle, visando reduzir a interação humana com a rede e os erros de origem humana. Idealmente, pesquisadores imaginam um projeto de rede que não seja apenas automático (*i.e.,* , dependente de instruções humanas), mas autônomo (*i.e.,* , capaz de tomar suas próprias decisões). A rede autônoma tem sido uma meta buscada há anos, com diversos conceitos, projetos e implementações, mas nunca foi totalmente realizada, principalmente devido às limitações tecnológicas. Avanços recentes em Inteligência Artificial (IA) e Aprendizado de Máquina (*Machine Learning* — ML) introduziram ar fresco neste conceito, ressurgindo como o conceito renomeado de em redes autodirigidas, em vista de suas contrapartes de automóveis autodirigidos. Em termos gerais, uma rede autodirigida é uma rede autônoma capaz de agir de acordo com as intenções de alto nível de um operador e se adaptar automaticamente às mudanças no tráfego e no comportamento dos usuários. Para alcançar essa visão, uma rede precisaria cumprir quatro requisitos principais: *(i)* compreender as intenções de alto nível de um operador para ditar seu comportamento, *(ii)* monitorar-se com base nas intenções de entrada, *(iii)* prevêr e identificar padrões em dados monitorados e *(iv)* adaptar-se a novos comportamentos sem a intervenção de um operador.

Como o cumprimento dos requisitos de uma rede autônoma exige uma grande dependência de modelos de ML para tomar decisões e classificações que afetam diretamente a rede, um problema específico se torna proeminente com esse design: confiança. Aplicar ML para resolver tarefas de gerenciamento de rede, como as descritas acima, tem sido uma tendência popular entre os pesquisadores recentemente. No entanto, apesar do tópico receber muita atenção, os operadores da indústria têm relutado em tirar proveito de tais soluções, principalmente por causa da natureza de caixa preta dos modelos de ML, que produzem decisões sem qualquer explicação ou razão para as quais essas decisões foram tomadas. Dada a natureza de alto risco das redes de produção, torna-se impossível confiar em um modelo de ML que pode tomar ações inadequadas automaticamente e, o mais importante para o escopo desta tese, um desafio proibitivo que deve ser abordado para que uma rede autodirigidas seja alcançada.

A presente tese visa habilitar redes autônomas, abordando o problema da falta de confiança inerente nos modelos de ML que a capacitam. Para tanto, avaliamos e escrutinamos o processo de tomada de decisão de classificadores baseados em ML usados para compor uma rede autônoma. Primeiro, investigamos e avaliamos a precisão e credibilidade das classificações feitas por modelos de ML usados para processar intenções de alto nível do operador. Para essa avaliação, propomos uma nova interface conversacional chamada LUMI que permite aos operadores usar linguagem natural para descrever como a rede deve se comportar. Segundo, analisamos e avaliamos a precisão e a credibilidade das decisões tomadas pelos modelos de ML usados para autoconfigurar a rede de acordo com os dados monitorados. Nessa análise, descobrimos a necessidade de reinventar como os pesquisadores aplicam IA/ML a problemas de rede, então propomos um novo *pipeline* de IA/ML que apresenta etapas para examinar modelos de ML usando técnicas do campo emergente de IA eXplicável (IAX). Por fim, investigamos se existe um método viável para melhorar a confiança dos operadores nas decisões dos modelos de ML usados no gerenciamento de redes autodirigidas. Nossa investigação nos levou a propor um novo método XAI para extrair explicações de qualquer modelo de ML caixa-preta na forma de árvores de decisão, mantendo um tamanho gerenciável, que chamamos de TRUSTEE. Nossos resultados mostram que os modelos de ML que foram amplamente aplicados para resolver problemas de rede não foram submetidos ao escrutínio adequado e podem quebrar facilmente quando colocados sob estresse. Esses modelos são, portanto, inúteis do ponto de vista prático e precisam ser corrigidos para cumprir adequadamente as tarefas que lhes são atribuídas.

Palavras-chave: Redes Autodirigidas. Aprendizado de Máquina. Explicabilidade. Redes Baseadas em Intenções. Conhecimento de Operadores. Segurança Cibernética.

“If we knew what it was we were doing,
it would not be called research, would it?”
— Albert Einstein

ACKNOWLEDGEMENTS

I would like to start by thanking my advisor Prof. Lisandro Zambenedetti Granville, for the incredible guidance and numerous teachings throughout the 11 years of my academic life. The road to my Ph.D. and my life in academia started in 2011 as a first-year undergraduate researcher under Lisandro. From the start, Lisandro saw and believed in my potential as a researcher and computer scientist. I lost count of how many times I would just pop up in his office, uninvited, looking for research directions and, many times, advice on career and life choices. Every time I would walk away reassured I was on the right path.

Likewise, I want to extend my gratitude to my co-advisor, Prof. Ronaldo Alves Ferreira. What started as an eventful collaboration in early 2018 turned into years of fantastic work, dedication, and teachings from Ronaldo. Ronaldo was an integral part of my development as a researcher, constantly pushing me to do better and improve myself. From many nights spent writing and reviewing papers to calls on weekends to discuss results, Ronaldo was always present and available to help and guide me. I cannot be more grateful to have had Ronaldo and Lisandro "in my corner" throughout my Ph.D. course. To them, I would like to give my most profound appreciation.

I would also like to thank Dr. Walter Willinger, who happily accepted to host and guide me during my Sandwich period at Princeton University and continued to guide and teach me until the end of my Ph.D. To be honest, I am not sure what Walter saw in me to accept such a request from an unknown collaborator. Still, I am very grateful to have had him be part of my research and personal development. Similarly, I would like to thank Prof. Sanjay Rao and Prof. Arpit Gupta for their help and support in our many collaborations. Indeed, I owe them much for the quality of the research developed in my Ph.D.

I must thank all the outstanding professors from the Institute of Informatics, including Prof. Luciano Paschoal Gaspary and Prof. Marinho Pilla Barcellos. Their guidance and teaching during my Ph.D. course classes were fundamental to my development as a student and researcher. I am also very grateful for Prof. Jennifer Rexford and all her research group at Princeton University. In particular, Robert, Mary, Ross, Sata, Natalie, Danny, Molly, Victor, Joon, and Shir. I want to thank them all for welcoming me into their lab, for all the opportunities and support they gave me during my time at Princeton, and most importantly, for the many Thursday happy hours and Friday watch parties of

The Bachelor. Those times to unwind and relax were crucial for me to feel welcomed in a foreign lab and gave me a much-needed rest to keep my research going. Also, Robert, once again, I am sorry for breaking your television.

During my Sandwich period at Princeton University, where my wife Gabriela and I were living in a shared townhouse, we witnessed the rise of the COVID-19 pandemic. The pandemic urged us to stay isolated in our home with our housemates, and we could not have hoped to have had better housemates than Michel and Andrea. I want to thank them so much for helping us cope with the uncertainties of the time through a lot of training and exercises and many wine dinner parties.

The remainder of my acknowledgements will be written in Portuguese, as it is my native language, so that my friends and family in Brazil can read them.

Eu não poderia deixar de agradecer os meus muitos colegas do Grupo de Redes, em especial do laboratório 210. A todos que conheci e tive o prazer de conviver desde meu ingresso no grupo em 2011, e a lista é longa, deixo meu profundo agradecimento. Em especial, em meu período inicial no laboratório, tenho que mencionar Oscar, Ricardão, Jefferson, Jedi, Weverton e Cadori, que me ensinaram o a importância do café queimado na vida acadêmica (e de muitas risadas). Já ao ingressar no doutorado, conheci novos e importantes amigos no laboratório, como Matias, Bondan, Zaca, Henrique, Ian, Ana, Lando, Marrone, Mirim, Vlad, Sena e Marcus Vinicius. Em especial, preciso agradecer ao grupo que me mais me ensinou a fazer pesquisa: a *Papers Machine*, composta pelos grandes amigos Ricardão, Tocaio, Muriel e Éder. Se eu sei escrever um artigo científico hoje é por conta das inúmeras revisões e sabatinas as quais eles me submetiam. Além disso, é claro, com todos acima mencionado, partilhei do dia a dia-na-vida acadêmica, desde muitas discussões filosóficas sobre o impacto da pesquisa na sociedade, ao futebol de todas as quintas. Este grupo de amigos e colegas são alguns dos grandes responsáveis por eu chegar ao final deste doutorado.

Agradeço também aos colegas do projeto JEMS, do qual participei por um período do meu doutorado, como Cássio, Ana, Chico, Renata, Sena, e os professores Juliano Wickboldt e Alberto Schaeffer-Filho. Aos amigos, Adolfo, Augusto, Thom e Rámon, agradeço pelas muitas tardes de risadas que passamos juntos. Em especial, ao grupo de amigos que tenho a felicidade de carregar comigo desde o início da minha graduação, Prola, Taís e Jonathan. Agradeço também aos amigos Viktor e Natalí, pela longa e duradoura amizade, desde nosso ensino fundamental.

À minha família, dedico meu mais profundo agradecimento, pelo suporte, apoio,

amor e carinho que me deram para que eu chegasse ao final desse doutorado. À minha mãe, Beatriz Regina Jacobs (*in memoriam*), que não pode ver o seu "filho nômade" terminar seu "doutorado em física, né filho?", e à minha avó, Julieta Fleck Selle (*in memoriam*). Sofri com a perda dessas duas mulheres fortes, amáveis, engraçadas e queridas, que não poupavam esforços pra demonstrar o orgulho que sentiam do seu filho/neto que fazia um doutorado. A elas, dedico meu amor eterno, e saibam que sempre as terei comigo, onde quer que estejam. Ao meu pai, Nelson Jacobs, e ao meu irmão, Lourenço Selle Jacobs, agradeço pelo amor e suporte incondicionais. À minha sogra, Yuko Yamashita, e meu cunhado, Lucas Yamashita, por me receberem tão bem em sua família.

Por fim e mais importante, eu preciso agradecer à minha esposa, Gabriela Lumi Yamashita Rodrigues. Me faltam palavras pra descrever o quão grato sou por tê-la na minha vida, por esses 11 anos juntos, e 24 anos que nos conhecemos. Graças à ela entrei no doutorado, e graças à ela eu terminei. A sua importância na minha vida, e no meu crescimento como pessoa e pesquisador é tão grande que parte do trabalho dessa tese leva seu nome. Ela é, e sempre será, a razão dos meus melhores dias. Nos meus piores dias, é ela que me segura e me reergue. A ela eu dedico essa tese, e todo meu amor. Muito obrigado por tudo!

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

1D-CNN	One-dimensional Convolutional Neural Network
AI	Artificial Intelligence
ABR	Adaptive Bitrate
ACL	Access Control List
CLI	Command-Line Interfaces
CNN	Convolutional Neural Network
CRF	Conditional Random Fields
DARPA	Defense Advanced Research Projects Agency
DDoS	Distributed Denial of Service
DPI	Deep Packet Inspection
DNN	Deep Neural Network
DT	Decision Tree
GB	Giga Byte
HTTPS	Hypertext Transfer Protocol Secure
IAT	Inter Arrival Time
IBN	Intent-Based Networking
IP	Internet Protocol
IPv4	Internet Protocol Version 4
IPv6	Internet Protocol Version 6
IDD	Independent and Identically
IDS	Intrusion Detection System
IRTF	International Research Task Force
LENMA	Local Explanation Method using Nonlinear Approximation
LIME	Local Interpretable Model-Agnostic Explanations

LSTM Long Short-Term Memory

MB Mega Byte

MAC Media Access Control

ML Machine Learning

MSN Microsoft Messenger

NER Named Entity Recognition

NFV Network Function Virtualization

NLP Natural Language Processing

NVGRE Network Virtualization using Generic Routing Encapsulation

OOD Out-of-distribution

PCAP Packet Capture

POS Parts-of-Speech

QoS Quality of Service

RL Reinforcement Learning

RFC Request For Comments

RNN Recurrent Neural Networks

SDN Software-Defined Networking

SHAP SHapley Additive exPlanations

TCP Transmission Control Protocol

UDP User Datagram Protocol

VLAN Virtual Local Area Network

VNI VXLAN Network Identifier

VPN Virtual Private Network

VLAN Virtual Local Area Network

VXLAN Virtual Extensible Local Area Network

XAI eXplainable Artificial Intelligence

CONTENTS

1 INTRODUCTION

As modern networks grow in size and complexity, they also become increasingly prone to human errors (BECKETT *et al.*, 2017; FEAMSTER; REXFORD, 2018). This trend has driven both industry and academia to try to automate management and control tasks, aiming to reduce human interaction with the network and human-made mistakes (BECKETT *et al.*, 2016; APOSTOLOPOULOS, 2020; NETWORKS, ; VMWARE, 2021). Ideally, researchers envision a network design that is not only automatic (*i.e.,* dependent on human instructions) but autonomous (*i.e.,* capable of making its own decisions). Autonomous networking has been a goal sought for years, with many different concepts, designs, and implementations, but it was never been fully realized, mainly due to technological limitations (FEAMSTER; REXFORD, 2018). Recent advances in Artificial Intelligence (AI) and Machine Learning (ML) introduced a breath of fresh air into this concept, reemerging as the re-branded concept of *self-driving networks*, in view of its autonomous car counterparts.

1.1 Self-Driving Networks

While the concept of a self-driving network has no standardized definition, with each company and researcher having its own vision and architecture (MOGUL, 2018; FEAMSTER; REXFORD, 2018; JUNIPER, 2017; NETWORKS, 2018; HUAWEI, 2019; WATTS, 2020; FOSTER *et al.*, 2020), a few design elements are common in all instances. As no standard definition has been adopted as standard by the community, we rely on the following definition of a self-driving network, summarizing the main aspects found in the literature, which is thoroughly described in Chapter 2.

Definition 1.1.1. A self-driving network is an autonomous network capable of acting according to high-level intents from an operator and automatically adapting to changes in traffic and user behavior. To achieve that vision, a network would need to fulfill four major requirements: *(i)* understand high-level intents from an operator to dictate its behavior, *(ii)* monitor itself based on input intents, *(iii)* predict and identify changing patterns from monitored data, and *(iv)* adapt itself to new behaviors without the intervention of an operator.

In Figure 1.1, we present a high-level design of a self-driving network that sum-

marizes the features and requirements found in the literature.

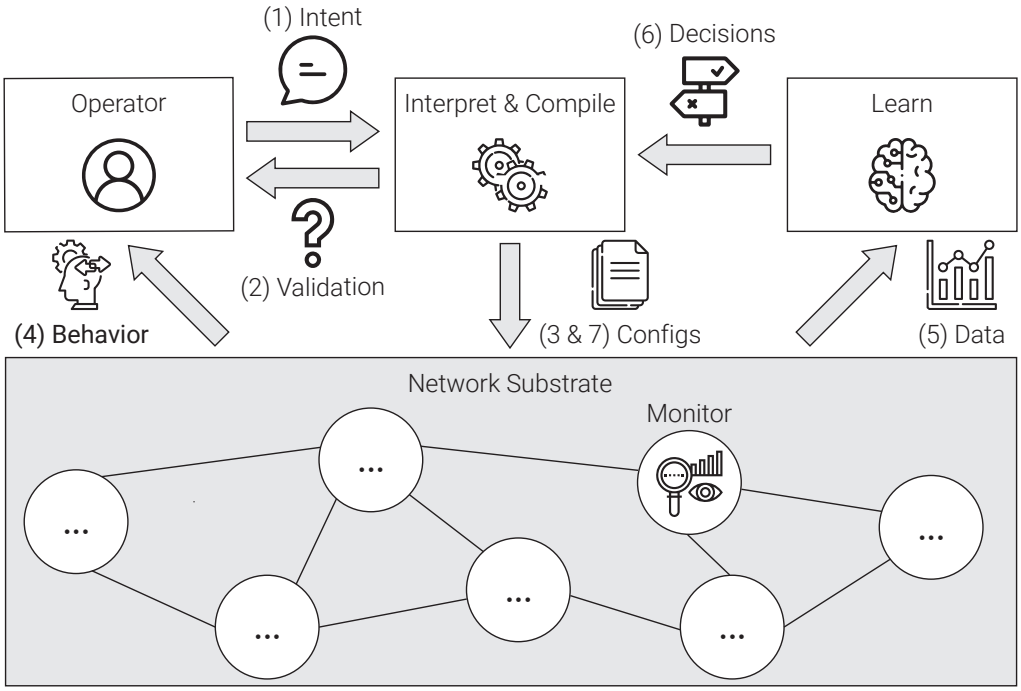


Figure 1.1 – Self-driving network design.

This self-driving network design is composed of two management loops[1]. On the left side of Figure 1.1, we see the first management loop (1, 2, 3, and 4), which starts with an operator (1) specifying high-level intents that dictate how the network should behave—*e.g.*, defining goals related to quality of service, security, and performance–without worrying about the low-level details that are necessary to program the network to achieve these goals (*a.k.a.* Intent-based Networking — IBN) (CLEMM *et al.*, 2019). Ideally, specifying such intents should be as easy as describing them in natural language. However, enabling operators to freely use natural language to describe network intents requires the network to employ state-of-the-art ML techniques from Natural Language Processing (NLP) (JURAFSKY; MARTIN, 2019), which are always prone to generate errors and misclassifications. Hence, after extracting relevant information from input intents, a self-driving network would then (2) validate the extracted data with the operator before (3) compiling them into actual configurations and deploying them to the network substrate. To close the first management loop, the network would then monitor itself according to the described intents and collect the behavior of traffic to (4) present it for operators to verify if the implemented behavior corresponds to the initial goals.

[1]One may question our use of the term 'management loop' in favor of 'control loop'. While there are good arguments for both terminologies, we rely on the definitions provided by IRTF RFC 7426 (HALEPLIDIS *et al.*, 2015) as a guide.

On the right side of Figure 1.1, the second management loop (5, 6, and 7) begins after intents are deployed into the network substrate, where devices are instrumented with monitoring capabilities to collect usage and traffic data. Such data is then (5) analyzed and processed to (6) produce (autonomous) decisions using trained ML models, which would ideally adapt and re-train themselves constantly as new data is collected. Decisions made by such ML models would then be (7) processed and compiled into configurations to fine-tune the network behavior (akin to 3), closing the second management loop. Most notably, two networking areas would benefit the most from such a management loop: network security and network performance. For instance, decisions from ML models may include the identification of attack traffic that needs to be blocked or mitigated or even resource usage optimizations based on traffic load. Notice that, despite also relying heavily on error-prone ML techniques, given the time frame and frequency such decisions and configuration deployment would occur to keep up with incoming traffic, it would be impossible to include human validation on this second management loop, in contrast with the first one. In addition, another key aspect to keep in mind is that self-configurations made based on decisions from ML models in the second management loop can undermine and contradict the configurations made through network intents in the first management loop. The possibility of such conflicts arising, coupled with the inability to include human-in-the-loop validations, raises the stakes for any decision made by ML models in the second management loop.

1.2 Problems Statement and Research Questions

As both management loops in the self-driving network presented in Figure 1.1 rely heavily on ML models to make decisions and classifications that directly impact the network, one particular issue becomes prominent with this design: trust. Applying ML to solve networking management tasks, such as the ones described above, has been a popular trend among researchers recently (BOUTABA *et al.*, 2018). However, despite the topic receiving much attention, industry operators have been reluctant to take advantage of such solutions, mainly because of the black-box nature of ML models which produce decisions without any explanation or reason as to which those decisions were made. Given the high-stakes nature of production networks, it becomes impossible to trust an ML model that may take system-breaking actions automatically, and most important to the scope of this thesis, a prohibitive challenge that must be addressed if a self-driving network design is

ever to be achieved.

Problem Statement: *The present thesis aims to enable self-driving networks with ML, tackling the problem of the inherent lack of trust in ML models that empower it by assessing their decision-making process.*

To address the described problem statement, we must first investigate and evaluate the accuracy and credibility of classifications made by ML models used to process high-level intents from the operator. Then, we must analyze and assess the accuracy and credibility of decisions made by ML models used to self-configure the network according to monitored data. Lastly, we must investigate if there is a viable method to improve the trust of operators in the decision made by ML models in both management loops. To pave the road towards solving the problem above, we formulate three research questions that guide the development of this thesis. The first research question this thesis poses concerns the use of ML techniques to parse relevant information from input network intents and compile them into a network configuration that fulfills the given intents.

RQ-1: *In a self-driving network, can operators trust decisions and classifications made by current Machine Learning models used to configure the network based on high-level intents?*

We address the first research question in Chapter 3. To that end, we need to assess the reliability and usability of an IBN-enabled system that allows network operators to describe network intents using natural language and compiles them into network configuration accordingly, as shown in the first management loop of the self-driving network presented if Figure 1.1. However, while reviewing the existing literature, we found that no end-to-end system had been proposed to allow that. Hence, our first step to answering **RQ-1** is to propose LUMI, a conversational assistant (*i.e.,* chatbot) that lets operators use natural language to describe high-level network intents. LUMI relies on highly accurate ML techniques from NLP to extract information from input intents and uses its conversational interface to confirm with operators if the parsed information is correct. To get confirmation from operators, we propose a high-level Network Intent LanguagE (Nile) that resembles structured English, using it as an abstraction layer between natural language intents and network configuration commands. More importantly, LUMI is capable of collecting feedback from operators on the *Nile* intents built with information extracted from natural language and uses that feedback to retrain and improve the underlying ML model accuracy over time. Our evaluation shows that LUMI can accurately extract information from both synthetic and real-world input intents. We also assess LUMI's usability

by conducting a user study with network practitioners from different backgrounds and levels of expertise. The results from this chapter indicate that, with minimal safeguarding, operators can, in fact, trust classifications made by current ML models used to parse information from high-level intents.

The second research question this thesis poses concerns the use of ML models to make decisions based on data collected from the network substrate devices and automatically configure the network to adjust and optimize resource usage and security.

RQ-2: *In a self-driving network, can operators trust decisions and classifications made by current Machine Learning models used to self-configure the network based on monitored data analysis?*

We address the second research question in Chapter 4. To that end, we focus our efforts on analyzing the application of ML techniques in the network security problem space as a use-case representing the second management loop of the self-driving network presented in Figure 1.1. As mentioned before, many works that apply ML-based classifiers have been proposed in the network security domain. Hence, to answer **RQ-2**, we survey the existing literature to select a few key reproducible ML-based works to put under scrutiny and reproduce their results. While most works in the literature report outstanding classification accuracy in their evaluation scenarios, we are concerned with the credibility of the classifications made. In particular, since we cannot rely on operators to review every decision made by classification models in a fast-paced management loop if a model trained with synthetic or unrealistic datasets fails when presented with unforeseen real-world traffic, it can lead to disastrous outcomes. To scrutinize the results of the selected works, we rely on the growing field of eXplainable Artificial Intelligence (XAI) techniques, which allow us to extract white-box explanations from black-box models. The particular XAI method used to scrutinize models is described later in Chapter 5. Our results show that, when scrutinized, the analyzed ML models fail to fulfill their tasks, indicating that operators are, in fact, correct not to trust ML models to automatically configure the network based solely on monitored data.

Finally, the third and last research question this thesis poses concerns the general lack of trust from operators in ML models that self-driving networks rely on, and how that trust can be increased.

RQ-3: *Is there a feasible way to increase operator trust in the decision-making process of Machine Learning models that configure a self-driving network?*

The third and last research question is answered in Chapter 5. To answer **RQ-3**,

we first equate *"an operator having trust in an AI/ML model"* with *"an operator being comfortable with relinquishing control to the AI/ML model"* (LIPTON, 2018). Given this specific definition of what it means for an AI/ML model to engender trust, we surveyed the state-of-the-art to find XAI methods to scrutinize ML models and identified a gap in existing XAI methods to produce faithful explanations for any given black-box model. Such methods would potentially increase the trust operators have in the decision-making process of ML models, provided that the produced explanations were faithful, of a manageable size, and, most of all, sound. In trying to fill that gap and answer **RQ-3**, in this chapter, we propose TRUSTEE, a novel model-agnostic XAI method to produce high-fidelity explanations in the form of decision trees that approximate the decision-making process from black-box models. In addition, we also implement a series of analyses based on the generated decision tree explanation and underlying data, condensed into a *trust report*. Our results show that the explanations produced by TRUSTEE, alongside the generated *trust reports*, allow operators and domain experts to carefully scrutinize ML models and decide whether they are trustworthy or suffer from any inductive biases. The results we obtained through answering **RQ-2** and **RQ-3** confirm our intuition that ML models widely applied to solve networking problems have not been put under proper scrutiny and can easily break if applied to real-world scenarios. Such models must then be corrected to fulfill their given tasks properly in practice if the self-driving network is ever to be achieved.

1.3 Main Contributions

In the process of answering the three research questions posed in this thesis, we make the following contributions:

- We propose an end-to-end intent-based network management system with a conversational interface that allows operators to use natural language to describe their desired intents for the network behavior. We rely on that system to evaluate the accuracy of current ML techniques in extracting relevant information from high-level intents, which can be manually verified by operators through the conversational interface.

- We survey the existing literature on the use of ML techniques for network security and scrutinize several use-cases from the literature to analyze the credibility of deci-

sions made by highly-accurate ML models that enable a self-driving network. The results we obtained uncover systematical issues with how researchers have been employing ML-based classifiers to solve network security problems.

- To increase trust and reliability of the current approach to ML models, we propose a new methodological pipeline for how researchers and operators should employ and evaluate ML techniques to solve networking problems. To implement our proposed pipeline, we introduce a novel model-agnostic XAI method to produce explanations from any given black-box ML model in the form of decision trees. We also introduce a novel pruning method to ensure extracted explanations are concise while maintaining a good fidelity, and a *trust report* that condenses and analyzes the most relevant aspects of the extracted explanation to aid domain experts in spotting issues with the black-box model.

1.4 Thesis Organization

The remainder of this thesis is organized as follows. In Chapter 2 we present a description and background on a topic important to this thesis. First, we provide an overview of existing efforts to implement IBN and NLP in networking. Then, we recognize and describe past efforts to achieve autonomous networks and highlight the key aspects from which they differ from modern self-driving network architectures. We then describe the background and challenges of employing ML for networking problems in general. Lastly, we classify and detail existing XAI approaches and methods to interpret black-box ML models and conclude the chapter.

In Chapter 3 we address the first research question. We start by describing the overall design of LUMI with an illustrative example. LUMI is composed of a pipeline of 4 modules, each responsible for one aspect of interpreting intents in natural language and compiling and deploying them as network configurations: *(i)* information extraction; *(ii)* intent assembly, *(iii)* intent confirmation, and *(iv)* intent deployment. Then, we describe in detail how each module is designed and implemented and the reasoning behind our design choices. To assess how accurate LUMI is in extracting information from input intents, we evaluate it against two datasets of synthetic and real-world intents. Then we proceed to describe the results of a small-scale user study conducted with network practitioners to assess how LUMI would fare when presented with unforeseen and unpredictable test cases

and conclude the chapter.

In Chapter 4 we address the second research question. First, we perform a limited survey to establish the current state of ML employment in network security-related problems, grouping the results according to their ability to be reproducible into three categories: *(i)* ML model is published, and training data is publicly available; *(ii)* ML model is published, and training data is not publicly available, and *(iii)* ML model is not published, and training data is not available. This limited survey helps us highlight the precarious state-of-the-art in ML-based network security research, where reported results cannot be taken at face value. We also present a comparative use case of an ML application to a network performance problem that guides our efforts on how ML models should be scrutinized. This comparative use case helps us establish the need to rethink how AI/ML is applied to network security problems, motivating our proposal of a new AI/ML pipeline. Following this new proposed pipeline, we reproduce and evaluate three outstanding research efforts of failed use of ML to solve network security-related problems. Finally, we discuss the outcomes and consequences of our evaluation and conclude the chapter.

In Chapter 5 we address the third and last research question. First, we discuss the insights obtained from answering previous research questions to establish when improving trust is made more necessary. We then provide a clear problem statement on why we need a new XAI method to produce explanations in the form of decision trees and how they can be used to improve the trust of operators and domain experts in black-box ML models. We also describe existing XAI approaches and why they fail to achieve our purpose. We then present TRUSTEE, a novel model-agnostic method to produce explanations for ML models in the form of decision trees that are both high-fidelity and manageable in size, as well as the accompanying *trust reports* that aid domain experts in spotting problems with the model or the underlying training data. We evaluate and compare TRUSTEE to existing XAI methods to show that it can produce explanations with higher fidelity and lower complexity than competing methods and briefly describe our findings on other use cases explored while developing TRUSTEE. Finally, we discuss the outcomes of the insights obtained in the evaluation and conclude the chapter.

In Chapter 6 we discuss the final remarks of this thesis and present the research directions for future works. The research questions are revisited given our presented results, and we discuss the remaining gaps in our research that are left for future investigations.

2 BACKGROUND AND STATE-OF-THE-ART

Aiming to provide a clear understanding of the challenges inherent to designing and implementing a self-driving network, we review base concepts in this chapter and discuss the state-of-the-art. To that end, we present the background knowledge and related work regarding the IBN concept, review and recognize past autonomous networking efforts, the challenges of applying ML to networking, and summarize the emerging field of XAI.

2.1 Intent-based Networking

Although IBN is a fundamental stepping stone in achieving self-driving networks, the concept exists on its merits and predates the resurgence of autonomous networks. However, akin to self-driving networks, IBN arose as a natural evolution of Policy-Based Management Systems (PBMN), much due to the recent advances in ML and NLP. Hence, this section describes related work on IBN, NLP, and network management research efforts.

2.1.1 IBN-enabled network management.

INSpIRE (SCHEID *et al.*, 2017) focuses on intents related to security middleboxes and uses a refinement process to determine which middleboxes should compose a service chain to fulfill an intent. Cheminod et al. (CHEMINOD *et al.*, 2019) propose an automatic method for refining, deploying, and enforcing ACL policies that use set notation for policy specification. PGA (PRAKASH *et al.*, 2015) applies a graph-based abstraction to compose high-level policies and deploy them in SDN networks. PGA supports ACL and service-chaining policies, leveraging a graph structure to resolve conflicts. Janus (ABHASHKUMAR *et al.*, 2017) extends PGA to support policies with QoS requirements, mobility, and temporal dynamics. However, these proposals are not concerned with using natural language or obtaining feedback from the network operator.

2.1.2 Natural language in networking.

Very few prior works use natural language to interact with the network. In (BIRKNER *et al.*, 2018), the authors present *Net2Text*, a system that allows network operators to query network-wide forwarding behaviors using natural language, but it does not allow operators to configure the network. Alsudais *et al.* (ALSUDAIS; KELLER, 2017) proposes using natural language to deploy network intents. It uses the Stanford CoreNLP Parts-of-Speech (POS) Tagger (TOUTANOVA *et al.*, 2003) to parse and structure the input text and look for specific keywords to match "entities" using a brute-force pattern-matching, which does not generalize easily. While this is a step in the right direction, their approach limits the inherent flexibility of using natural language. Also, the paper does not cover NLP aspects relevant to IBN and self-driving networks, such as intent confirmation for user feedback and learning over time.

2.1.3 Network programming languages.

Recent works on IBN feature several intent languages, frameworks, and compilers to efficiently deploy intents in network devices and middleboxes (PRAKASH *et al.*, 2015; ABHASHKUMAR *et al.*, 2017; SUNG *et al.*, 2016; FOSTER *et al.*, 2011; ANDERSON *et al.*, 2014; KIM *et al.*, 2015; SOULÉ *et al.*, 2018; RYZHYK *et al.*, 2017). At the same time, Cocoon (RYZHYK *et al.*, 2017) introduces a framework to guarantee the correctness of SDN programs that resembles our approach, but it uses first-order logic. Despite not examining the use of machine learning to convert natural language intents into lower-level configurations, network programming languages can be viable candidates to serve as abstraction levels for configuring a heterogeneous network substrate.

2.1.4 Natural language processing.

Information extraction has been addressed with different methods and techniques, including *(i)* only Conditional Random Fields (CRFs) models (FINKEL; GRENAGER; MANNING, 2005); *(ii)* character-level embeddings instead of whole words (LAMPLE *et al.*, 2016), and *(iii)* rule-based approaches (CHITICARIU *et al.*, 2018). Yet, recent studies (YADAV; BETHAR, 2018) show the benefits of ML-based approaches such as recurrent neural networks. To our knowledge, the problem of using natural language for management tasks has not received much attention in the networking domain.

2.2 Autonomous Networks

This section provides an overview of previous attempts and proposals that lead to the current notion of self-driving networks. To that end, in Figure 2.1, we present a (non-exhaustive) timeline of the evolution of critical autonomous network proposals, from autonomic computing until today. For a more in-depth survey and description of past autonomous networks and autonomic computing, please check (MARQUEZAN; Z.GRANVILLE, 2010) and references therein.

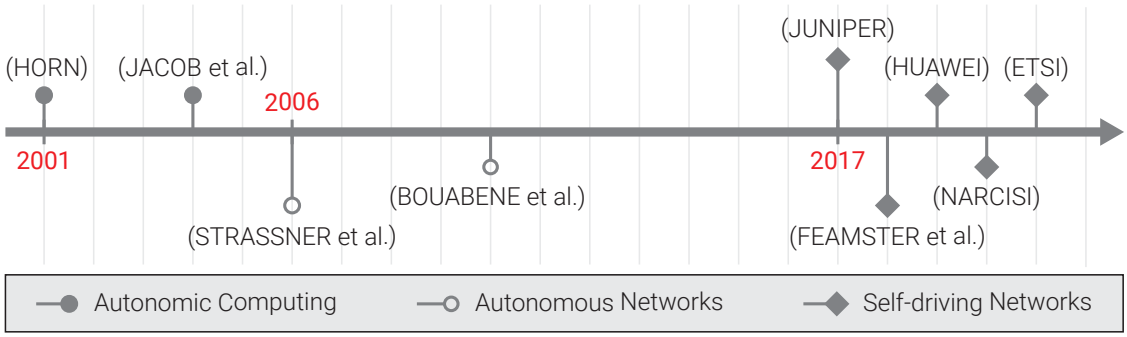


Figure 2.1 – Evolution of self-driving network proposals over time.

The concept of autonomic computing was introduced in 2001 by IBM (HORN, 2001), where the authors first proposed the notion of systems that "run themselves", reducing the need for human intervention. In their proposal, the authors describe a set of characteristic autonomous systems must-have, which later came to be known in the community as the self-* (*i.e.,* "self-star") properties: self-awareness, self-protecting, self-optimizing, self-healing, and self-configuring. Later, in 2004, IBM incorporated the self-* autonomic computing properties into an Autonomic Management Engine, responsible for executing the MAPE-K loop (JACOB *et al.*, 2004), shown in Figure 2.2, an intelligent control loop that enables Monitoring, Analyzing, Planning, and Executing tasks in an autonomous system, using a shared Knowledge base. In any autonomic system, such as an autonomous network, the Autonomic Management Engine would provide a hosting environment for the decision algorithms, regardless of the technologies used.[1]

The efforts and advances made in autonomic computing did not go unnoticed by the networking community, which saw an opportunity to take advantage of these concepts to create a self-managing network. One of the proponents of this idea was introduced in 2006 as the FOCALE autonomic networking architecture (STRASSNER; AGOULMINE;

[1] While not completely relevant to the remainder of this thesis, the autonomic computing field continued to see research in later years.

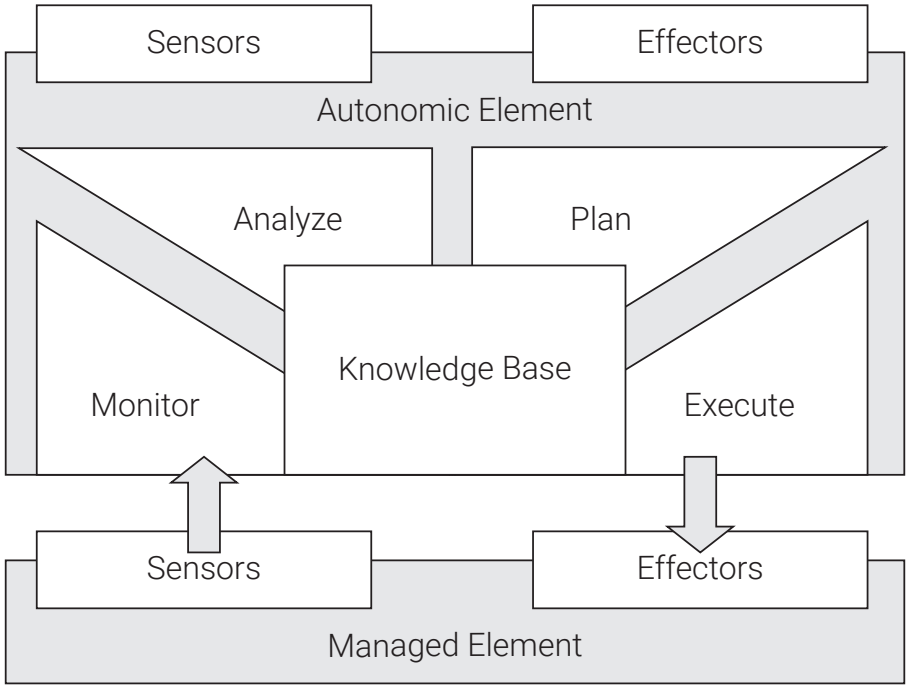


Figure 2.2 – Autonomic Management Engine control loop.

LEHTIHET, 2006). FOCALE incorporated many aspects of IBM's autonomic computing properties, including the presence of an Autonomic Manager and a shared knowledge base. One noticeable difference, though, was the inclusion of a Policy Manager responsible for translating business rules *written in a restricted natural language* into actual configuration commands for managed devices, for being executed by the Autonomic Manager. In addition, the FOCALE architecture focused a lot of effort on specifying viable communication mechanisms between the Autonomic Manager and managed resources, relying heavily on SNMP and XML.

Similarly, in 2010, the Autonomic Networking Architecture (ANA) (BOUABENE *et al.*, 2010) was proposed as an evolution of previous autonomous networking efforts by introducing an abstraction level for managing and manipulating elements at a system level rather than at a device or resource level. Still, instead of introducing novel aspects and requirements to autonomic networking on a conceptual level, the ANA is more concerned with providing tools to communicate and implement existing autonomic computing properties in a network. The ANA was one of the last significant efforts to consolidate an autonomous network architecture, after which the area saw a decline in publications and research, arguably due to the technological limitation of the time.

After years of research and development in the networking and ML communities, in 2017/2018, both industry (JUNIPER, 2017) and academia (FEAMSTER; REXFORD, 2018) saw an opportunity to materialize the concept of a network that runs itself,

re-branding the idea as a self-driving network. On paper, all of the properties and requirements inherent to autonomic computing and early autonomous networking efforts are also present in our definition of a self-driving network and its requirements. However, two differences are key: *(i)* the requirement for easily specifying the desired behavior for the network through intents, given the advent of IBN, and *(ii)* the ML technologies that empower a self-driving network. From 2001 to the early 2010s, AI and ML were still far from being mature to be widely applied as they are today, and because of that, they were not a fundamental part of autonomous systems designs. Although efforts like the MAPE-K loop and FOCALE had "learning" components, they were based on rudimentary forms of acquiring and storing knowledge of past events rather than the powerful pattern recognition capabilities and statistical analysis of modern ML. Although not crucial to its design, a self-driving network may also take advantage of technological advances in SDN-related areas, such as P4 (BOSSHART *et al.*, 2014) and Network Function Virtualization (NFV) (JACOBS *et al.*, 2017; JACOBS *et al.*, 2018).

The concept of a self-driving network quickly gained traction, prompting multiple companies to start autonomous networking initiatives, which continue to this day, such as Huawei in 2019 (HUAWEI, 2019) and Juniper Networks in 2020 (NARCISI, 2020). In an attempt to consolidate these efforts and reconcile new concepts with existing autonomous networking approaches, members of the European Telecommunications Standards Institute (ETSI) published a white paper (ETSI, 2020), in 2021, with key challenges, capabilities, and use cases of modern autonomous networks. This white paper is particularly keen on describing the pivotal role self-driving networks will have, in the near future, in helping operators manage fast-growing large-scale network environments while supporting the arrival of new business partners and solution providers.

2.3 Challenges in ML for Networking

Beyond the already-mentioned trust issue, there are several other reasons why the area of networking is a particularly challenging application domain for AI/ML. For one, by their very nature, networking-related datasets in general, and security-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 severe end-user-specific privacy concerns or reveal provider-specific details that many companies consider

proprietary and are therefore unwilling to share. The result is a general lack 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.

Moreover, even if data was abundant, 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. In turn, they typically consist of semantically rich content, and unpacking that content and properly labeling it requires substantial domain knowledge (*e.g.,* network architecture, protocols, 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 (DENG *et al.*, 2009)) 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 in the autonomous car technology area (CAESAR *et al.*, 2019; CHANG *et al.*, 2019)).

Computer vision researchers have *ImageNet* (DENG *et al.*, 2009), and researchers working on autonomous driving technology have increasing access to datasets provided by the various commercial driverless technology companies (*e.g.,* NuScenes (CAESAR *et al.*, 2019), ArgoVerse (CHANG *et al.*, 2019), Waymo Open Dataset (SUN *et al.*, 2020)). Access to these open-source datasets is widely credited with invigorating reproducible AI/ML-related research in these areas. In stark contrast, fledgling reproducibility efforts in the network security area quickly reduce demands for data sharing in an application domain of AI/ML where data is hard to come by in the first place. Such data sharing requirements have severely hamstrung reproducible AI/ML research in the network security domain.

2.4 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 (LIPTON,

2018). In summarizing past and ongoing efforts concerning this concept, our objectives are clarifying existing misconceptions, agreeing on a standard set of notational conventions, and refining the current discourse on the topic.

2.4.1 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, they provide no insight into or understanding of why the black-box model makes certain decisions (and not some other decisions) or what decision-making process gives rise to these decisions. However, in an increasing number of application domains of AI/ML and especially in those domains where the resulting learning models are used for high-stakes decision-making, domain experts have become more vocal about requiring precisely that insight and gaining exactly that understanding before deploying these models in practice. This development has resulted in a recent explosion of work on "Explainable AI," where a second (post-hoc) model is created to explain the initially obtained black-box model (TUREK, 2016). This pursuit of explainable AI has been criticized in the recent AI/ML literature and called "problematic" (see, for example (RUDIN, 2019)), mainly because such post-hoc explanations are often not reliable and can be misleading (LAKKARAJU; BASTANI, 2020; GHORBANI; ABID; ZOU, 2019). An alternative approach advocated in (RUDIN, 2019) argues for using learning models such as linear models or decision trees that are inherently (*i.e.,* ex-ante) interpretable.

Unfortunately, in many modern-day application domains of AI/ML, the complexity of the tasks for which the use of AI/ML models is being envisioned and the rich semantic content of the data that is leveraged for training these models pose significant challenges for the exclusive application of inherently interpretable learning models. In the network security domain, for example, even the most knowledgeable experts readily admit to being at a loss when it comes to deciding under which conditions which aspects of the available data matter the most (and why) for, say, developing a well-performing AI/ML model for detecting particular types of cyber events, such as ransomware attacks.

As a result, providing the means for effectively accounting for the data's rich semantic content and efficiently uncovering the patterns in the data that matter for the problem at hand have increasingly become the responsibility of trained "black-box" models rather than purposefully chosen "white-box" models. However, instead of consider-

ing this development as “problematic”, we view it as a unique opportunity to ultimately achieve the vision of interpretable ML – ensuring that AI/ML models used for high-stakes decision-making are entirely understandable by their end-users and interested third parties.

2.4.2 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 than their interpretable counterparts (*e.g.,* decision trees) and are therefore often more desirable when used in practice. Although this argument is not universally shared (*e.g.,* see (RUDIN, 2019)), it nevertheless has been a driving force behind the recent efforts on the topic of “explainable AI.” Occasionally referred to XAI (TUREK, 2016), explainable AI describes steps where the development of a trained black-box model is followed up with additional activities intended to help “explain” the originally obtained black-box model. These efforts can be divided into two disjoint categories, depending on the nature of the additional activities.

2.4.2.1 Local Explainability.

One approach to “explaining” black-box models post-hoc is to provide *local explanations* for individual predictions of the black box. Methods for providing such 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 (RIBEIRO; SINGH; GUESTRIN, 2016), SHAP (LUNDBERG; LEE, 2017), and LEMNA (GUO *et al.*, 2018).

In general, despite their popularity as readily available tools for cracking open black-box models, local explainability methods can only provide, by their very nature, limited information about the black-box model in question. For example, one can use them to examine which features the black-box models deemed most relevant for making a particular decision. However, these methods typically reveal no information about which values these features must take. They also provide no insights into how the different features interact with each other to produce a particular decision. Importantly, because they limit their attention to only a subset of individual choices, these methods are prone to

providing misleading explanations (LIPTON, 2018; MOLNAR *et al.*, 2020; ZHANG *et al.*, 2020), depending on the subset of samples analyzed. Related methods such as Partial Dependence Plots (PDP) (FRIEDMAN, 2001) and Accumulated Local Effect (ALE) plots (APLEY; ZHU, 2020) suffer from similar shortcomings.

In general, while local explanations play an important role in shedding light on how black-box models "work" in a narrow sense (*i.e.,* at the level of individual decisions), they are unable to reveal how these models work in a broader sense (*i.e.,* explain the "inner workings" of a given black-box model holistically). As such, they are of limited use when we seek explanations that we can trust in that they are faithful to how the original black-box model makes decisions.

2.4.2.2 Global Explainability.

An alternate approach to obtaining post-hoc explanations of black-box models is to provide *global explanations* that describe a given black-box model as a whole. Such explanations typically take the form of an inherently interpretable model, such rule sets or a decision tree (BASTANI; KIM; BASTANI, 2017; LAKKARAJU *et al.*, 2017). Assuming that the approximation quality, or "fidelity", of such an extracted "white-box" model, is sufficiently good compared to the original black-box model, then the extracted interpretable model becomes the main vehicle for studying the decision-making process of the underlying black-box model and examining its properties.

Despite the intuitive appeal that such global explanations have for examining the "inner workings" of black-box models, their practical applications pose several challenges. For one, there is the problem that these post-hoc explanations may not be reliable or can be misleading (LAKKARAJU; BASTANI, 2020; GHORBANI; ABID; ZOU, 2019). Moreover, existing approaches for extracting such post-hoc explanations either produce too low fidelity to be helpful in practice (BASTANI; KIM; BASTANI, 2017), target only a particular set of black-box models (BASTANI; PU; SOLAR-LEZAMA, 2018), or are difficult to reproduce (LAKKARAJU *et al.*, 2017; LAKKARAJU; BASTANI, 2020). To achieve the level of explainability required in high-stakes application domains such as network security and self-driving networks, we seek to generate high-fidelity global explanations capable of accurately describing most decisions made by any given black-box model.

2.5 Chapter Summary and Remarks

In this chapter, we presented an overview of the state-of-the-art and background knowledge on technologies that empower self-driving networks. We start by describing the current state of IBN research and related areas and follow with a timely recap of previous efforts on autonomous networking and autonomic computing. This literature review highlights how critical ML is to achieve the vision of a self-driving network, from parsing natural language network intents to automatically identifying patterns in the network traffic, and how it will shape the network management industry landscape in the coming future. From that perspective, we proceeded to present the main challenges to applying ML to networking and security problems, most of them stemming from the lack of high-quality data available, which emphasizes the crucial role the emerging field of XAI will have in scrutinizing network-related ML models. Finally, we presented a much-needed notational agreement and background on existing XAI techniques. Most importantly, by analyzing the state-of-the-art on IBN and XAI, we can perceive apparent gaps in the literature on both methods to translate natural language intents to network configuration and model-agnostic techniques to explain the decisions of black-box models globally. In the coming chapters, we address these gaps in the literature while answering the research questions posed by this thesis.

3 MACHINE LEARNING FOR INTENT-BASED NETWORKING

Deploying policies in modern enterprise networks poses significant challenges for today's network operators. Since policies typically describe high-level goals or business intents, the operators must perform the complex and error-prone job of breaking each policy down into low-level tasks and deploying them in the physical or virtual devices of interest across the entire network. Recently, IBN has been proposed to solve this problem by allowing operators to specify high-level policies that express how the network should behave (*e.g.,* defining goals for quality of service, security, and performance) without having to worry about how the network is programmed to achieve the desired goals (CLEMM *et al.*, 2019). Ideally, IBN should enable an operator to simply tell the network to, for example, *"Inspect traffic for the dorm"*, with the network instantly and correctly breaking down such an intent into configurations and deploying them in the network. Supporting IBN is one of the core principles of a self-driving network and a crucial component of the first management loop shown in Figure 3.1, as it reduces human intervention to a necessary minimum, which also reduces the number of errors introduced by human mistake.

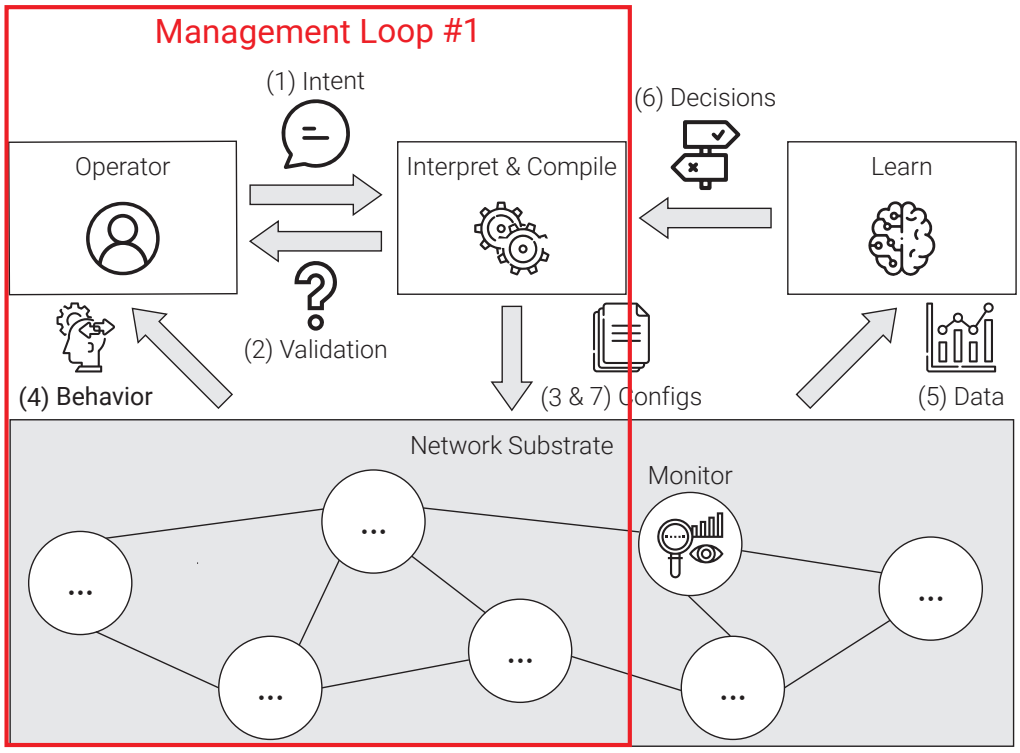


Figure 3.1 – First management loop of a self-driving network.

In its current form, IBN has not yet delivered on its promise of fast, automated, and reliable policy deployment. One of the main reasons for this shortcoming is that, while network policies are generally documented in natural language, we cannot cur-

rently use them as input to intent-based management systems. Despite growing interest from some of the largest tech companies (APOSTOLOPOULOS, 2020; NETWORKS, ; VMWARE, 2021) and service providers (LIU, 2021; KOLEY, 2016; RESEARCH, 2018), only a few research efforts (BIRKNER *et al.*, 2018; ALSUDAIS; KELLER, 2017) have exploited the use of natural language to interact with the network, but they lack support for IBN or other crucial features (*e.g.,* operator confirmation and feedback). However, expressing intents directly in natural language has numerous benefits when it comes to network policy deployment. For one, it avoids the many pitfalls of traditional policy deployment approaches, such as being forced to learn new network programming languages and vendor-specific Command-Line Interfaces (CLI), or introducing human errors while manually breaking down policies into configuration commands. At the same time, its appeal also derives from the fact that it allows operators to express the same intent using different phrasings. However, the flexibility makes it challenging to generate configurations, which must capture operator intent in an unambiguous and accurate manner, a feat not easily achieved when relying on ML models to extract information from natural language.

In our efforts of addressing the first research question, concerning trust in ML models used in NLP, we must first bridge the existing gap in IBN literature between natural language and network configurations. In this chapter, we contribute to the ongoing IBN efforts by describing the design and implementation of LUMI (JACOBS *et al.*, 2021), a new system that enables an operator “to talk to the network", focusing on campus networks as a use case. That is, LUMI takes as input an operator’s intent expressed in natural language, correctly translates these natural language utterances into configuration commands, and deploys the latter in the network to carry out the operator’s intent. We designed LUMI in a modular fashion, with the different modules performing, in order: information extraction, intent assembly, intent confirmation, and intent compilation and deployment. Our modular design allows for easy plug-and-play, where existing modules can be replaced with alternative solutions, or new modules can be included. As a result, LUMI’s architecture is extensible and evolvable and can easily accommodate further improvements or enhancements.

In addressing the various challenges above, we make the following contributions:

Information extraction and confirmation. We build on existing ML algorithms for *Named Entity Recognition (NER)* (JURAFSKY; MARTIN, 2019) to extract and label entities from the operator’s natural language utterances. In particular, we implement NER

using a chatbot-like interface with multi-platform support (§3.2) and augment the existing algorithm so that LUMI can learn from operator-provided feedback (§3.4).

Intent assembly and compilation. We introduce the *Network Intent Language* (Nile), use it as an abstraction layer between natural language intents and network configuration commands for LUMI, and illustrate its ability to account for features critical to network management such as rate-limiting or usage quotas (§3.3). We also show how LUMI enables compilations of *Nile* intents to existing network programming languages (§3.5), such as *Merlin* (SOULÉ *et al.*, 2018) .

Evaluation. We evaluate (§3.6) LUMI's accuracy in information extraction, investigate LUMI's ability to learn from operator-provided feedback and measure both the compilation and deployment times in a standard campus topology (AMOKRANE *et al.*, 2015). Using our own datasets consisting of synthesized intents as well as real-world intents derived from network policies published by 50 different campus networks in the US, we show that LUMI can extract entities with high precision, learn from the feedback provided by the operator, and compile and deploy intents in less than a second.

User study. In addition to an in-depth evaluation of LUMI, we also report our main findings of a small-scale user study, with 26 subjects (§3.7). The study was performed to get feedback from subjects on the perceived value of using natural language for network management with LUMI and soliciting user feedback during the intent confirmation stage.

Prototype. We implemented our prototype of LUMI using a combination of tools and libraries (*e.g.,* Google Dialogflow (INC., 2018), *Scikit-learn* library (PEDREGOSA *et al.*, 2011)). The full implementation as well as all datasets used in our evaluation are available on the project's website (LUMI, 2020).

Together, the results of our evaluation and user study show that LUMI is a promising step towards realizing the vision of IBN of achieving fast, automated, and reliable policy deployment. By allowing operators to express intents in natural language, LUMI makes it possible for operators to simply talk to their network and tell it what to do, thus simplifying the jobs of network operators (*i.e.,* deploying policies) and also saving them time. While promising, developing LUMI into a full-fledged production-ready system poses new and interesting challenges on the interface of networking and NLP, which we detail in §6.2.

3.1 Lumi in a Nutshell

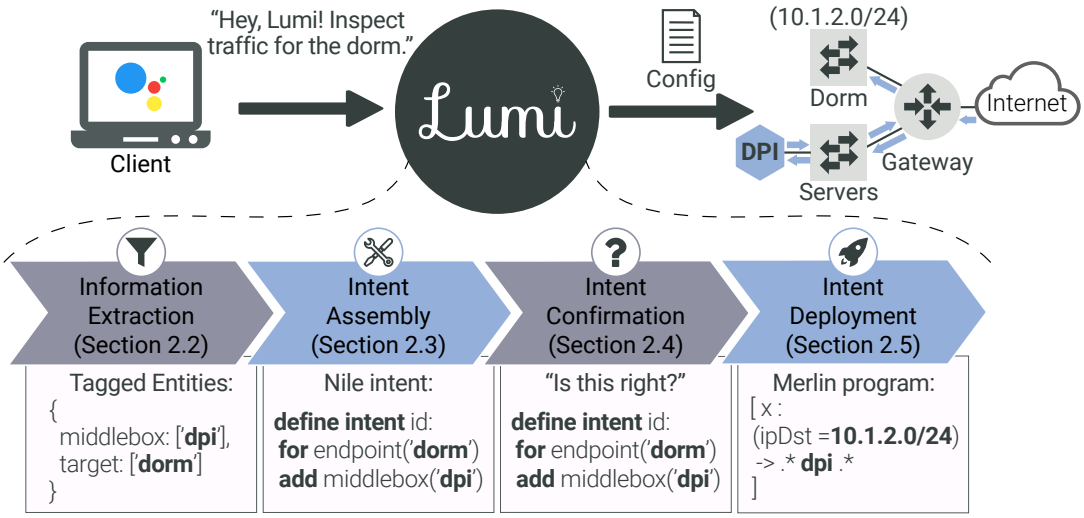


Figure 3.2 – The four modules of LUMI.

Figure 3.2 illustrates the high-level goal of LUMI with the intent example *"Hey, Lumi! Inspect traffic for the dorm"* and shows the breakdown of the workflow by which LUMI accomplishes the stated objective. Below, we provide a brief overview of the four key components that define this workflow (*i.e.,* the LUMI pipeline) and refer to the subsequent sections for an in-depth description and design choices of each of these modules.

First, for the Information Extraction module (described in Section 3.2), we rely on machine learning to extract and label entities from the operator utterances and implement them using a chatbot-like conversational interface. The extracted entities form the input of the Intent Assembly module (described in Section 3.3), where they are used to compose a *Nile* network intent. *Nile* closely resembles natural language, acts as an abstraction layer, and reduces the need for operators to learn new policy languages for different types of networks. Then, as part of the Intent Confirmation module (described in Section 3.4), the output of the Intent Assembly module (*i.e.,* a syntactically-correct *Nile* intent) is presented to the network operator, and their feedback is solicited. If the feedback is negative, the system and the operator iterate until confirmation, with the system continuously learning from the operator's feedback to improve the accuracy of information labeling over time. Finally, once the system receives confirmation from the operator, the confirmed *Nile* intent is forwarded to the Intent Deployment module. Described in Section 3.5, this module's main task is to compile *Nile* intents into network configuration commands expressed in *Merlin* and deploy them in the network. In Section 3.5, we also explain why we picked *Merlin* as a target language over other alternatives.

3.2 Information Extraction

The main building blocks for LUMI's Information Extraction module are a chatbot interface as the entry point into our system and the use of Named Entity Recognition (NER) (JURAFSKY; MARTIN, 2019) to extract and label entities from the operators' natural language intents. Given the popularity of personal assistants, such as Google Assistant, Amazon's Alexa or Apple's Siri, our goal in providing a natural language interface for LUMI goes beyond facilitating the lives of traditional network operators and seeks to also empower users with little-to-no knowledge of how to control their networks (*e.g.,* home users).

Even in the case of the traditional user base of network operators, providing a natural language interface to interact with the system benefits teams composed of operators with different levels of expertise or experience. This type of interface is particularly relevant in campus or enterprise networks with small groups and in developing countries where network teams are often understaffed and lack technical expertise. In short, while deploying a policy as simple as redirecting specific traffic for inspection can be a daunting task for an inexperienced operator, nothing is intimidating about expressing that same policy in natural language and letting the system worry about its deployment, possibly across multiple devices in the network.

Solving the NER problem typically involves applying machine learning (for extracting named entities in unstructured text) in conjunction with using a probabilistic graphical model (for labeling the identified entities with their types). Even though, in theory, NER is largely believed to be a solved problem (MARRERO *et al.*, 2013), in practice, to ensure that NER achieves its purpose with acceptable accuracy, some challenges remain, including careful "entity engineering" (*i.e.,* selecting entities appropriate for the problem at hand) and a general lack of tagged or labeled training data.

Below, we first discuss the entity engineering problem and propose a practical solution in the form of a hierarchically-structured set of entities. Next, we describe in more detail the different steps that the NER process performs to extract named entities from a natural language intent such as *"Inspect traffic for the dorm"* and label them with their types. Finally, to deal with the problem caused by a lack of labeled training data, we describe our approach that improves upon commonly-used NER implementations by incorporating user feedback to enable LUMI to learn over time.

Table 3.1 – Hierarchical set of entities defined in LUMI.

Type	Entity Class
Common	@middlebox, @location, @group, @traffic, @protocol, @service, @qos_constraint, @qos_metric, @qos_unit, @qos_value, @date, @datetime, @hour
Composite	@origin, @destination, @target @start, @end
Immutable	@operation, @entity

3.2.1 Entity Selection

To ensure that NER performs well in practice, a critical aspect of specifying the underlying model is entity engineering; that is, defining which and how many entities to label in domain application-specific but otherwise unstructured text. On the one hand, since our goal with LUMI is to allow operators to change network configurations and manipulate the network's traffic, the set of selected entities will affect which operations are supported by LUMI. As discussed in more detail in Section 3.3, LUMI-supported actions are dictated almost entirely by what intent constructions *Nile* supports. At the same time, we would like to consider a generic enough set of entities to enable users to express their intents freely, independent of *Nile*. Moreover, the selected set of entities should also allow for easy expansion (*e.g.,* through user feedback; see Section 3.4 below) while ensuring that newly added entities neither introduce ambiguities nor result in unforeseen entries that might break the training model.

On the other hand, the selected set of entities directly influences the trained model's accuracy, especially if the entities have been chosen poorly. Therefore, it is crucial to pick a set of entities that is at the same time rich enough to solve the task at hand and concise enough to avoid ambiguities that might hamper the learning process. For instance, one common source of uncertainty in network intents is highlighted with the two examples *"Block traffic from YouTube"* and *"Block traffic from the dorms"*. Here, the word 'from' appears in both intents but is used for two different purposes. While in the first example, it specifies a service, in the second example, it defines a location. If we choose to tag both entities (*i.e.,* "YouTube" and "dorms") with the same entity type (*e.g.,* "location") to avoid ambiguities, we lose information on what is being tagged (*i.e.,* service vs. location). However, if we simply use different entities for both cases (*e.g.,* "service" and "location"),

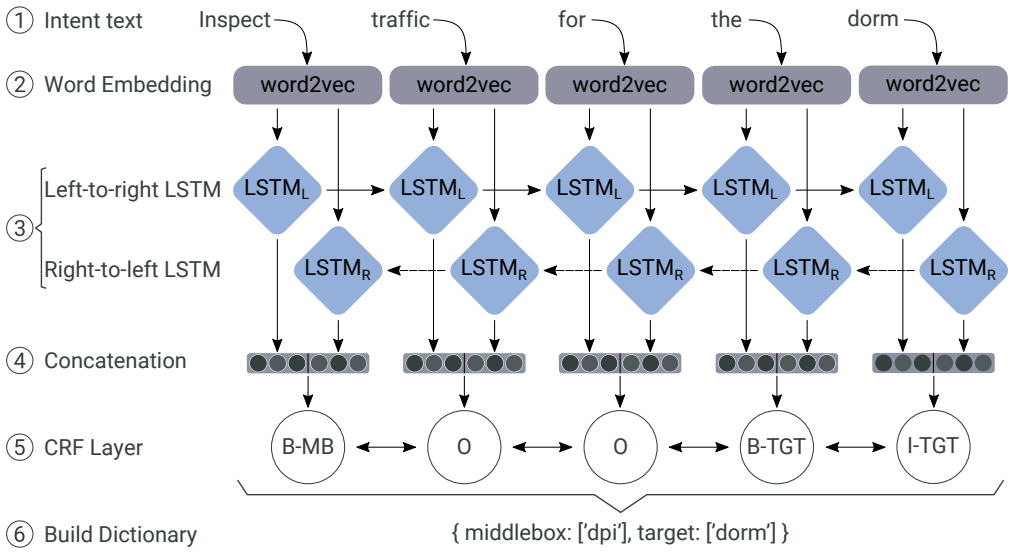


Figure 3.3 – The NER architecture with Bi-LSTM (see Section 3.2.2) and CRF (see Section 3.2.3). The entity tags are abbreviated as MB for middlebox and TGT for target.

we generate an ambiguity (*e.g.,* very similar phrasings produce entirely different results) that causes the accuracy of the NER model to decrease as similar example intents are encountered (*e.g., "Block traffic from Twitter"*).

With these design requirements in mind, we defined the set of LUMI entities hierarchically and organized them into three different categories: common, composite, and immutable entities (see Table 3.1). Here, common entities form the bottom of the hierarchy, comprise raw textual values, and largely determine what LUMI can understand. For instance, the textual values in the common entity class @*middlebox* are network functions such as firewalls, packet inspection, and traffic shaping. The hierarchy's intermediate level consists of composite entities. The entities in this class do not have any inherent nouns, verbs, or even synonyms associated with them; they only establish a relationship between common entities through the aggregation of prepositions. For instance, the composite entity class @*origin* consists of composite values such as "from @*location*" and "from @*service*". Composite entities help avoid the ambiguity problem mentioned earlier. Finally, immutable entities make up the top of the hierarchy and form the core of LUMI. In particular, while the entity class @*operation* expresses the operations that *Nile* supports, the entity class @*entity* consists of a list of LUMI-supported common entities.

3.2.2 Entity Encoding: Bi-LSTMs

Figure 3.3 shows the overall NER architecture that we use in LUMI and illustrates the critical intuition behind NER; that is, finding named entities in unstructured text (*e.g.,* "Inspect traffic for the dorm" in Step 1 in Figure 3.3) and labeling them with their types (*e.g.,* Step 6 in Figure 3.3). With respect to the machine learning part of the NER problem, standard approaches leverage Recurrent Neural Networks (RNN) (LAMPLE *et al.*, 2016); *i.e.,* a family of neural networks that processes arrays of data sequentially, where the output of one element is carried over as input to the next element in the sequence. However, RNNs typically require numerical inputs and not words. Therefore, when applied in the context of our text processing problem, each word of an intent that the operator expresses in natural language has to be first encoded into a numerical vector before an RNN can process it. Rather than using one-hot encoding (JURAFSKY; MARTIN, 2019), a simple encoding scheme that represents each encoded word as a vector of 0's and 1's, we rely in Step 2 (see Figure 3.3) on a more powerful approach that uses one-hot encoded vectors of words as input to a word embedding algorithm known as $word2vec$ (MIKOLOV *et al.*, 2013). The $word2vec$ algorithm uses a pre-trained mini-neural network that learns the vocabulary of a given language (English, in our case) and outputs a vector of decimal numbers for each word. As a neural network, $word2vec$ can provide similar vector representations for synonyms, so that words with similar meanings (*e.g.,* 'dormitories' and 'housing') have similar embedded vectors, allowing an RNN to process them similarly.

Before processing the output of $word2vec$, we need to specify the type of RNN model in our architecture. The Long Short-Term Memory (LSTM) model has been a popular choice for text processing (HOCHREITER; SCHMIDHUBER, 1997) as it can capture and carry over dependencies between words in a phrase or intent (*e.g.,* to identify multi-word expressions). It also creates a context-full representation of each processed word as an encoded array of numerical values. However, to further enhance each word's context, in our LUMI design of the information extraction stage, we rely in Step 3 on an enhanced version of LSTM, the so-called Bi-LSTM model (CHIU; NICHOLS, 2016; LIMSOPATHAM; COLLIER, 2016). The Bi-LSTM approach yields the best results for the English language in a majority of evaluated cases (YADAV; BETHAR, 2018). In a Bi-LSTM, a phrase is evaluated by two LSTMs simultaneously, from left-to-right and from right-to-left, and the outputs of the two LSTMs are then concatenated to produce a single output layer at a given position, as shown in Step 4.

3.2.3 Entity Labeling: CRFs

The labeling part of the NER problem consists of using the context-full encoded vectors of words to represent the "observed" variables for a type of probabilistic graphical models known as Conditional Random Fields (CRFs) (LAFFERTY.; MCCALLUM; PEREIRA, 2001), as shown in Step 5. CRFs are a widely-used technique for labeling data, especially sequential data arising in natural language processing. Their aim is to determine the conditional distribution $P(Y|X)$, where $X = \{x_1, x_2, ..., x_n\}$ represents a sequence of observations (*i.e.,* encoded vectors of words in a sentence) and $Y = \{y_1, y_2, ..., y_m\}$ represents the "hidden" or unknown variable (*i.e.,* NER tags or labels) that needs to be inferred given the observations. CRFs admit efficient algorithms for learning the conditional distributions from some corpus of training data (*i.e.,* model training), computing the probability of a given label sequence Y given observations X (*i.e.,* decoding), and determining the most likely label sequence Y given X (*i.e.,* inference).

The technical aspects of the specific CRF model we use in this work are described in detail in LUMI's website (LUMI, 2020) and show the flexibility afforded by CRF models to account for domain-specific aspects of labeling sequential data (*e.g.,* accounting for the likelihood of one entity label being succeeded by another). However, irrespective of the specific CRF model used, the method outputs as the most likely tag for a given word the one with the highest probability among all tags. Specifically, at the end of Step 5, the result of the NER algorithm is provided in the form of named entities with IOB tagging (JURAFSKY; MARTIN, 2019). In IOB tagging, each named entity tag is marked with an indicator that specifies if the word is the beginning (B) of an entity type or inside (I) of an entity type. If a word does not match any entity, then the algorithm outputs an (O) indicator (for "outside"). Note that by using IOB tagging, it is possible to distinguish if two side-by-side words represent a single entity or two completely different entities. For instance, without IOB-tagging, it would be impossible to tag an operation like "rate limiting" as one single named entity. Finally, we parse the IOB-tagged words resulting from the NER model, build a dictionary with the identified entities, and output it in Step 6 as the final result of LUMI's Information Extraction module.

3.2.4 NER and Learning

The described NER process is an instance of a supervised learning algorithm. It uses a corpus of training data in the form of input-output examples where input is an intent (*i.e.,* phrase with named entities defined in LUMI and other words or non-entities), and an output is the list of named entities with IOB tagging (*i.e.,* correct entity tags or labels). The primary training step consists of both adapting the weights of the Bi-LSTM model to extract the desired X vector and re-calculating the conditional distribution $P(X|Y)$ to infer the correct NER tags Y and may have to be repeated until convergence (*i.e.,* Steps 3-5).

Note that retraining can be done each time the existing corpus of training data is augmented with new key-value pairs or with existing entities used in a novel context (*i.e.,* requiring a new tag). A basic mechanism for obtaining such new key-value pairs or for benefiting from the use of existing entities in a novel context is to engage users of LUMI and entice their feedback in real-time, especially if this feedback is negative and points to missing or incorrect pieces of information in the existing training data. By enticing and incorporating such user-provided feedback as part of the Intent Confirmation stage (see Section 5 for more details), LUMI's design leverages readily available user expertise as a critical resource for constantly augmenting and updating its existing training data set with new and correctly-labeled training examples that are otherwise difficult to generate or obtain. After each new set of key-value pairs is obtained through user feedback, LUMI immediately augments the training corpus and retrains the NER model from scratch.

In light of the hierarchical structure of our LUMI-specific entities set, with user-provided feedback, we aim to discover and learn any newly-encountered common entities. Moreover, as the data set of common entities is augmented with new training instances, the accuracy of identifying composite entities also improves. With respect to the immutable entities, since the entity class @*operation* dictates what operations LUMI supports and understands, these operations cannot be learned from user-provided feedback. However, given the limited set of LUMI-supported operations, a relatively small training dataset should suffice to cover most natural language utterances that express these operations. As for the entity class @*entity*, being composed of a list of LUMI-supported common entities, it ensures that user-provided feedback can be correctly labeled.

3.3 Intent Assembly

Using a chatbot interface with NER capabilities as the front end of LUMI solves only part of the network intent refinement and deployment problem. For example, if a network operator asks a chatbot *"Please add a firewall for the backend."*, the extraction result could be the following entities: *{middleboxes: 'firewall'}*, *{target: 'backend'}*. Clearly, these two key-value pairs do not translate immediately to network configuration commands. Assembling the extracted entities into a structured and well-defined intent that can be interpreted and checked for correctness by the operator before being deployed in the network calls for introducing an abstraction layer between natural language intents and network configuration demands.

To achieve its intended purpose as part of LUMI, this abstraction layer has to satisfy three key requirements. First, the operations supported by the abstraction layer's grammar have to specify what entities to extract from natural language intents. Instead of trying to parse and piece together every possible network configuration from user utterances, we require that a predefined set of operations supported by the abstraction layer's grammar guide the information extraction process. As a result, when processing the input text corresponding to a network intent expressed by the operator in natural language, we only have to look for entities that have a matching operation in the abstraction layer's grammar, as those are the only ones that the system can translate into network configurations and subsequently act upon.

A second requirement for this abstraction layer is to allow for easy confirmation of the extracted intents by the operators by having a high level of legibility. We note that requiring confirmation does not burden the operators in the same manner that requiring them to express their network intents directly in the abstraction layer's language would. For instance, it is well-known that a person who can understand a written sentence cannot necessarily create it from scratch (*i.e.,* lacking knowledge of the necessary grammar).

Finally, the abstraction layer is also required to allow different network back-ends as targets given that a wide range of candidate network programming languages (FOSTER *et al.*, 2011; MONSANTO *et al.*, 2013; ANDERSON *et al.*, 2014; KIM *et al.*, 2015; BECKETT *et al.*, 2016; RYZHYK *et al.*, 2017; SOULÉ *et al.*, 2018) exist, and none of them seem to have been favored more by operators or industry yet. Using an abstraction layer on top of any existing languages allows us to decouple LUMI from the underlying technology, so we can easily change it if a better option arises in the future.

3.3.1 *Nile*: Network Intent Language

To satisfy the above requirements, in our design of the Intent Assembly module (*i.e.,* stage two of the LUMI pipeline), we rely on the Network Intent Language (*Nile*) to serve as our abstraction layer language. In a previous work (JACOBS *et al.*, 2018), we proposed an initial version of *Nile* that provided minimal operation support for intent definition and focused primarily on service-chaining capabilities. Here, we extend this original version to cover crucial features for network management in real-world environments (e.g., usage quotas and rate-limiting). Closely resembling natural language, the extended version of *Nile* has a high level of legibility, reduces the need for operators to learn new policy languages for different types of networks, and supports different configuration commands in heterogeneous network environments. Operationally, we designed this module to ingest as input the output of the information extraction module (*i.e.,* entities extracted from the operator's utterances), assemble this unstructured extracted information into syntactically-correct *Nile* intents, and then output them.

Table 3.2 – Overview of *Nile*-supported operations.

Operation	Function	Required	Policy Type
from/to	endpoint	Yes	All
for	group/endpoint/service/traffic	Yes	All
allow/block	traffic/service/protocol	No	ACL
set/unset	quota/bandwidth	No	QoS
add/remove	middlebox	No	Service Chaining
start/end	hour/date/datetime	No	Temporal

Table 3.2 shows the main operations supported by our extended version of *Nile*, and the full grammar of *Nile* is made available in LUMI's website (LUMI, 2020). Some of the operations have opposites (*e.g.,* allow/block) to undo previously deployed intents and enable capturing incremental behaviors stated by the operators. Some operations in a *Nile* intent are mandatory, such as **from/to** or **for**. More specifically, an operator cannot write an intent in *Nile* without stating a clear target (using **for**) or an origin and a destination (using **from/to**) when the direction of the specified traffic flow matters.

To enforce the correct syntax of the assembled intent, we leverage the *Nile* grammar to guarantee that the intent contains the required information for correct deployment. If the grammar enforcement fails due to the lack of information, the system prompts the operator via the chatbot interface to provide the missing information. Assume, for example, that the operator's original intent stated *"Please add a firewall."*, without providing

the target of the intent. Since specifying a target is required according to the *Nile* grammar, the module will not attempt to construct a Nile program but will instead interact with the operator to obtain the missing information.

3.3.2 *Nile* Intents: An Example

With *Nile*, we can express complex intents intuitively. For example, an input like "*Add firewall and intrusion detection from the gateway to the backend for client B with at least 100mbps of bandwidth, and allow HTTPS only*" is translated to the *Nile* intent shown in Listing 3.1. The **group** function is used as a high-level abstraction for clients, groups of IP addresses, VLANs, or any other aggregating capacity in low-level network configurations. Note that the IDs provided by the operator must be resolved during the compilation process, as they represent information specific to each network. This feature of the language enhances its flexibility for designing intents and serving as an abstraction layer. The example illustrates how *Nile* provides a high-level abstraction for structured intents and suggests that the grammar for *Nile* is expressive enough to represent many real-world network intents.

```
define intent qosIntent:
  from   endpoint('gateway')
  to     endpoint('database')
  for    group('B')
  add    middlebox('firewall'), middlebox('ids')
  set    bandwidth('min', '100', 'mbps')
  allow  traffic('https')
```

Listing 3.1 – *Nile* intent example.

For comparison purposes, the same intent expressed in *Merlin*, an existing policy abstraction language with similarities to *Nile* (SOULÉ *et al.*, 2018), can be found in Listing 3.2. Clearly, compared to *Merlin*, the more verbose nature of *Nile's* design makes it easier to read and understand, especially for users with a lower technical level. Also note that *Nile*'s abstraction level allows operators to reference specific endpoints without manually mapping IP addresses, and to reference many different VLANs as a single **group**. Finally, the condition to allow HTTPS cannot be directly expressed in *Merlin* and must be configured indirectly via a firewall middlebox.

```
[ x: (ipSrc = 10.1.2.1 and
      ipDst = 10.1.2.10 and
      (vlan = 2 or vlan = 3 or
      vlan = 4)) .* firewall .* ids .*
], min(x, 100MB/s)
```

Listing 3.2 – *Nile* intent example written as *Merlin* program.

3.4 Intent Confirmation (Feedback)

The major challenge with using natural language to operate networks (*e.g.,* as part of IBN) is that the method is inherently ambiguous. There are many different forms in which an operator can express the same intent in different network environments in natural language. Despite recent advances, natural language processing is prone to producing false positives or false negatives, resulting in incorrect entity tagging, leading to deploying incorrect network configuration commands. However, to be of practical use, network configurations are required to be unquestionably unambiguous and correct-by-construction.

One approach to address this challenge is to create an extensive dataset with training phrases and entities. However, it is unrealistic to expect that such a dataset will cover every possible English language (or any other language for that matter) example of phrase construction with domain-specific entities. Operators are free to use terms or expressions that the system has never encountered in the past. However, without proper safeguards, any such new phrase will likely result in misconfigurations of the network. An alternative approach to implementing a reliable intent deployment process is through "learning from the operator." Here, the basic idea is to leverage the operators' existing knowledge by requesting their feedback on the information extracted from natural language intents. Then, in the case of negative feedback, it is critical to engage with the operators to identify missing or incorrect pieces, include this such acquired new knowledge as additional phrases or entities in the original training dataset, and retrain the original learning model.

Our solution to deal with the ambiguity inherent in using natural language to express network intents follows the learning approach and leverages the chatbot interface implemented as part of our first module. In particular, in designing the intent confirmation module for realizing stage three of the four-stage LUMI pipeline, we require the output of the intent assembly module (*i.e.,* syntactically-correct *Nile* intents) to be confirmed by the operator. When presented with assembled intents that result in false positives or

negatives, the operator is asked to provide feedback that we subsequently use to augment the original training dataset with new labeled data; that is, LUMI is capable of learning new constructs over time, gradually reducing the chances of making mistakes. While this interaction may slow down initial intent deployments until LUMI adapts to the operator's usage domain, it is essential to guarantee reliable intent refinement and deployment.

Extracting pertinent information from user feedback also requires identifying specific entities in the user text, similarly to extracting entities from an input network intent. To this end, LUMI uses the same NER model for both tasks, relying primarily on the immutable *@entity* for extracting which entity class is being augmented and what value is being added. Relying on the same NER model also requires us to train the model to identify and extract entities in the received user feedback. However, since we limit LUMI's interactions with the user to answering simple questions, processing user feedback does not require a large set of training samples.

To reduce an operator's need for technical knowledge during this intent confirmation stage, we opted for supporting feedback interactions that induce the operator to provide the information that LUMI needs (*i.e.,* offering suggestions and asking complementary questions). Also, to provide operators with more flexibility and not insist that they have to use specific keywords, this module compares the entities and values provided by the operators with synonyms already in the training dataset. Figure 3.4 illustrates a case where, due to a lack of training, LUMI misses an entity in the input text, and the confirmation mechanism lets operators easily catch this mistake and provide the necessary feedback for LUMI to learn. Also, note that our design of this module is conservative in the sense that operator feedback is requested for each assembled intent, irrespective of the level of confidence that LUMI has concerning its accuracy.

3.5 Intent Deployment

The fourth and last stage of LUMI compiles the operator-confirmed *Nile* intents into code that can be deployed on the appropriate network devices and executes the original network intent expressed by the operator in natural language. Fortunately, the abstraction layer provided by *Nile* enables compilations to a number of different existing network configurations, including other policy abstractions languages such as *Merlin* (SOULÉ *et al.*, 2018), *OpenConfig* (OPENCONFIG, 2016), *Janus* (ABHASHKUMAR *et al.*, 2017), *PGA* (PRAKASH *et al.*, 2015), and *Kinetic* (KIM *et al.*, 2015).

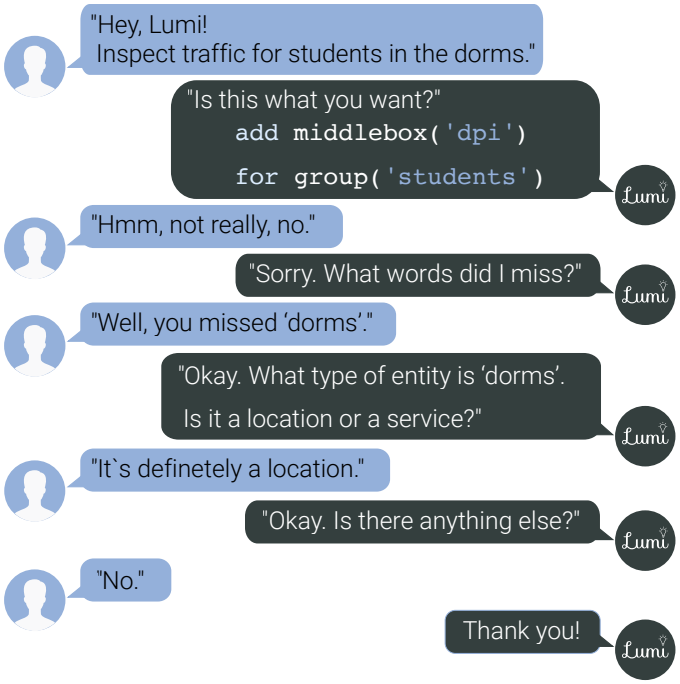


Figure 3.4 – LUMI's feedback mechanism in action.

For our design of LUMI's Intent Deployment module, we chose to compile structured *Nile* intents into *Merlin* programs. We picked *Merlin* over other alternative frameworks because of its good fit with *Nile*, the network features it supports, its performance, and the availability of source code. Also, given that none of the existing network programming languages natively supports all features proposed in *Nile*, we opted for the one that had the largest intersection of supported operations. In the process, we mapped each *Nile* feature to a corresponding *Merlin* feature.

Resolving logical handles. Logical handles in *Nile* intents are decoupled from low-level IP addresses, VLAN IDs and IP prefixes, which LUMI now resolves (*e.g.,* `dorm` →`10.1.2.0/24`) using information provided during the bootstrap process. ACLs rules are resolved similarly. Once LUMI produces *Merlin* programs with resolved identifiers (*i.e.,* VLAN IDs, IPs and prefixes), compilation to corresponding OpenFlow rules is handled by *Merlin*.

Temporal constraints and QoS. As *Merlin* does not support temporal policies, LUMI stores every confirmed *Nile* intent so that it can install or remove device configurations according to times and dates defined by the operator. We achieve quota restrictions (not natively supported by *Merlin*) by relaying all traffic marked with a quota requirement to a traffic-shaping middlebox, taking advantage of *Merlin*'s support for middlebox chaining. Other QoS policies, such as rate-limiting, are already supported in *Merlin*

Middlebox chaining. LUMI focuses on middlebox chaining, *i.e.,* correctly relay-

ing traffic specified in the intents through a specified middlebox. Since the actual configuration of each middlebox is currently done outside of LUMI, LUMI can handle chaining policies associated with *any* middlebox type, virtual or physical, compilation for which is straightforward since *Merlin* natively supports middlebox chaining.

3.6 Evaluation

In this section, we first evaluate the accuracy of our Information Extraction module to show that LUMI extracts entities with high precision and learns from operator-provided feedback. We then show that LUMI can quickly compile *Nile* intents into Merlin programs and deploy them. To assess the accuracy of the Information Extraction module, we use the standard metrics Precision, Recall, and F1-score. For the evaluation of the Intent Deployment stage, we measure the compilation time for translating *Nile* intents into Merlin statements and their deployment time.

3.6.1 Information Extraction

Evaluating systems like LUMI is challenging because of (i) a general lack of publicly available datasets that are suitable for this problem space (several operators we contacted in industry and academia gave proprietary reasons for not sharing such data), and (ii) difficulties in generating synthetic datasets that reflect the inherent ambiguities of real-world Natural Language Intents (NLIs).

To deal with this problem, we created two hand-annotated datasets for information extraction. The dataset *alpha* is "semi-realistic" in the sense that it is hand-crafted, consisting of 150 examples of network intents that we generated by emulating an actual operator giving commands to LUMI. In contrast, the *campi* dataset consists of real-world intents we obtained by crawling the websites of 50 US universities, manually parsing the publicly available documents that contained policies for operating their campus networks, and finally extracting one-phrase intents from the encountered public policies. From those 50 universities, we were able to extract a total of 50 different network intents. While some universities did not yield any intents, most universities published network policies related to usage quotas, rate-limiting, and ACL, and we were able to express all of them as *Nile* intents. We manually tagged the entities in each of these 200 intents to train and validate our information extraction model.

We used both datasets, separately and combined, to evaluate our NER model, with a 75%-25% training-testing random split. The small size of each dataset precludes us from performing conventional cross-validation. Table 3.3 shows the results for the *alpha* dataset, for the *campi* dataset, and for a combination of the two and illustrates the high accuracy of LUMI's information extraction module. Given the way we created the training examples for the *alpha* dataset, the excellent performance in terms of Precision, Recall, and F1-score is reassuring but not surprising. In creating the intent examples, we paid special attention to extracting all the entities defined in LUMI (see Section 3.2.1) and also creating multiple intents for each entity class.

Table 3.3 – Information extraction evaluation using the *alpha* and *campi* dataset.

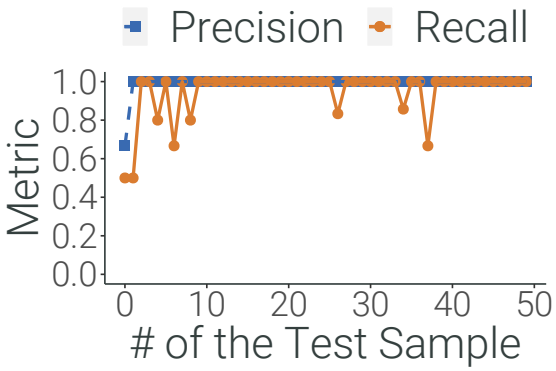
Dataset	# of Entries	Precision	Recall	F1
alpha	150	0.996	**0.987**	**0.991**
campi	50	**1**	0.979	0.989
alpha + *campi*	200	0.992	0.969	0.980

At the same time, despite the largely unstructured nature and smaller number of intent examples in the *campi* dataset, the results for that dataset confirm the above observation. Even though the example intents in this case were not designed with the NER model in mind, LUMI's performance remains excellent and is essentially insensitive to the differences in how the intent examples were generated. We attribute this success of LUMI at the information extraction stage to both continued advances in using machine learning for natural language processing and the fact that the complete set of LUMI-defined entities is relatively small and at the same time sufficiently expressive.

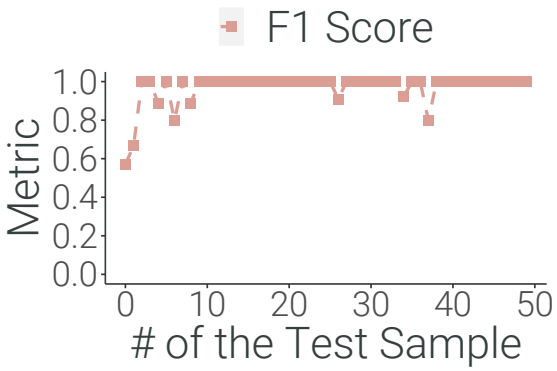
3.6.2 Intent Confirmation and Feedback

To evaluate the impact of operator-provided feedback on LUMI's ability to learn, we first trained our NER model using 75% of the combined *alpha* and *campi* datasets (*i.e.,* a total of 150 training examples) and then used the remaining 25% of examples (*i.e.,* a total of 50 test entries) as new intents that we presented LUMI in random order. We fed each of the 50 test intents into the NER model for information extraction and evaluated the Precision and Recall on a case-by-case basis. If a new intent generated False Positives or False Negatives, we inserted the intent into NER's existing training dataset, alongside the pre-tagged entities, mimicking the operator's feedback given during the Intent Confirmation stage.

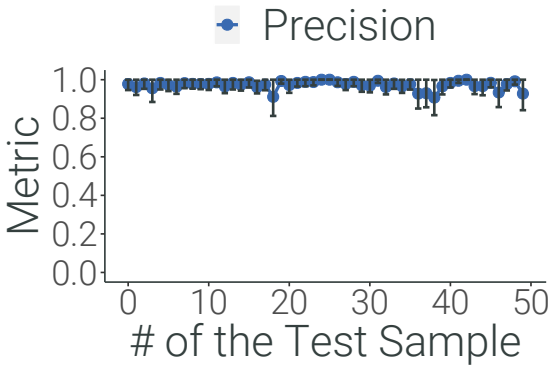
The results for this experiment (Precision, Recall and F1-score) are shown in Figures 3.5a and 3.5b. Since each point in the plots represents the Precision/Recall for one specific test sample rather than for the global model, the depicted values fluctuate as test samples are checked one after another. As can be seen, while Precision quickly reaches and then stays at 1.0, the Recall metric dips each time the model encounters an entity or construct it has not yet seen (*i.e.,* resulting in a False Negative). However, as more of these examples become part of NER's training data through the feedback mechanism (and because of re-training of the model after each new example is added), these dips become less frequent. Out of the 50 test intents, only eight resulted in a dip in Recall; that is, were used as feedback. Note, however, that the cases where the model does not identify an entity are precisely the situations where feedback is most informative and enables the model to learn for the benefit of the users.



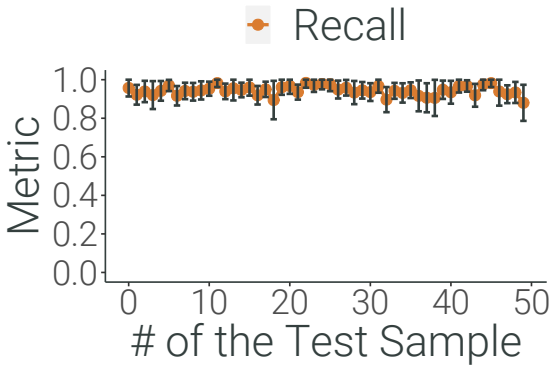
(a) Precision and Recall.



(b) F1 Score.



(c) Precision.



(d) Recall.

To assess how often a user has to rely on the feedback mechanism, we repeated our experiment 30 times, each time with a different 75-25 traininig-testing split. The resulting mean values for Precision and Recall are shown in Figures 3.5c and 3.5d, with corresponding 95% confidence intervals. As expected, over the 30 repetitions, both Pre-

cision and Recall remain close to 0.99, with very small fluctuations. And just as in the previous experiment, whenever there is a significant variation in Precision or Recall (*i.e.,* large confidence intervals), we added that particular intent example to NER's latest training dataset and retrained the model. We attribute the fact that about 20% of the test examples were used as feedback to the small size of our training dataset, but argue that having only this many feedbacks is a positive outcome.

3.6.3 Intent Deployment

To evaluate the deployment capabilities of LUMI, we compile and deploy five categories of *Nile* intents, with an increasing level of complexity: middlebox chaining, ACL, QoS, temporal, and intents with mixed operations. The last category of intents mixes *Nile* operations with distinct goals, which we use to evaluate the deployment of more complex intents. We generated a dataset with 30 intents per category, totaling 150 different intents, and measured the mean compilation and deployment time of each category. We ran this experiment on a generic campus network, with approximately 180 network elements. We relied on the Mininet (LANTZ; HELLER; MCKEOWN, 2010) emulator to perform the experiments. The results are given in Table 3.4. While deployment time necessarily depends on the network environment, in our setting, we consistently measured sub-second combined compilation and deployment times.

Table 3.4 – Compilation and deployment time for five categories of *Nile* intents.

Intent Type	Compilation Time (ms)	Deployment Time (ms)
Middlebox chaining	4.402	110
ACL	3.115	112
QoS	3.113	136
Temporal	4.504	111
Mixed	4.621	1030

3.7 User Study

To evaluate LUMI's ability to work in a real-world environment rather than with curated datasets of intents, we designed and carried out a small-scale user study. Specifically, we wanted to assess three critical aspects of our system: *(i)* How well does the information extraction process work with actual humans describing their intents in differ-

ent forms and phrasings? *(ii)* How often is it necessary for operators to provide feedback for LUMI while using the system? and *(iii)* Compared to existing alternative methods, what is the perceived value of a system like LUMI that leverages natural language for real-time network management?

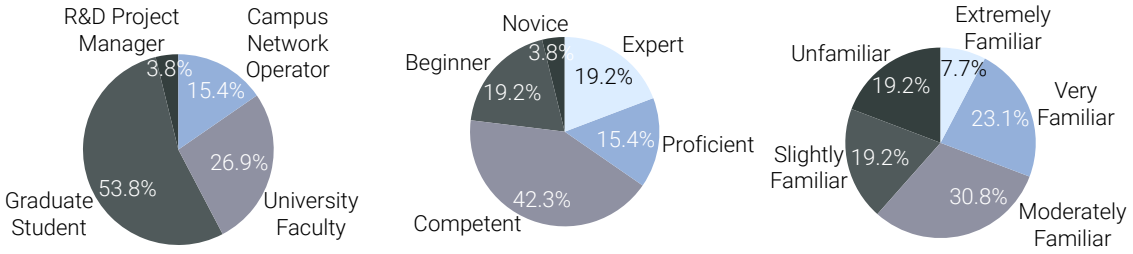
In this section, we describe the experiments we conducted, the participants' profiles, and the obtained results. We set up the user study as an online experiment that users could access and participate in anonymously. To select participants for the user study from different technical backgrounds and still keep their anonymity, we distributed a link to the online user study in mailing lists of both networking research groups and campus network operators. According to the guidelines of our affiliated institution, due to the fully anonymous nature of the experiment, no IRB approval was required to conduct this study, so this work does not raise any ethical issues.

3.7.1 Methodology

Participating users were asked to fill out a pre-questionnaire (*e.g.,* level of expertise, degree of familiarity with policy deployment, and use of chatbot-like interfaces) and then take on the role of a campus network operator by performing five specific network management tasks using LUMI. Based on the information these users provided in their pre-questionnaire, we had participants from three different continents: the Americas (88.5%), Europe (7.7%), and Asia (3.8%).

Each of the tasks required the user to enforce a specific type of network policy: *(i)* reroute traffic from the internet to an inspection middlebox; *(ii)* limit the bandwidth of guest users using torrent applications; *(iii)* establish a data usage quota for students in the university dorms; *(iv)* block a specific website for students in the labs, and *(v)* add a daily temporal bandwidth throttle to the server racks from 4pm to 7pm. Every interaction users had with LUMI was logged to a database for post-analysis.

After finishing the tasks, users were asked to complete a post-questionnaire (*e.g.,* number of completed tasks, the perceived value of LUMI, and comments on its usability). The complete set of management tasks presented to the users and all the results are available on the LUMI's website. Out of the 30 participants, four did not complete the online questionnaires and were excluded from the study, leaving a total of 26 subjects. Figure 3.6 shows a breakdown of the user profiles by type of job, level of experience with network management, and familiarity with chatbot-like interfaces.

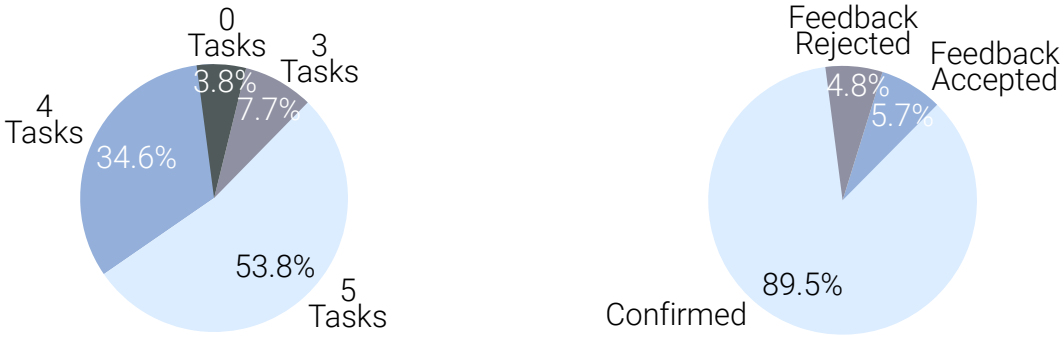


(a) Job. (b) Expertise. (c) Familiarity.

Figure 3.6 – Subjects profiles.

3.7.2 Information Extraction and Feedback

To assess the accuracy of our Information Extraction module, we use the number of tasks each participant concluded. For completing any given task, a specific set of labeled entities was required to build the correct *Nile* intent. Hence, each user's number of completed tasks reflects how accurately LUMI identified the entities in the input texts. The results in the left part of Figure 3.7 show that most users completed either 5/5 or 4/5 tasks. Some examples of successful intents for each task can be found on LUMI's website (LUMI, 2020). An analysis of the users' interactions with the system revealed that LUMI had trouble understanding temporal behavior (*e.g.,* "from 4pm to 7pm"), likely due to a lack of such training examples. This issue prevented some users from completing all five tasks. One user could not complete any task, reportedly because of an outage in the cloud provider infrastructure used to host LUMI.



(a) Tasks completed by users. (b) User Intents by status.

Figure 3.7 – LUMI information extraction and feedback.

To evaluate the value of LUMI's feedback mechanism as part of the Intent Confirmation module, we considered all intents that the 26 users generated and checked how many were confirmed and how many were rejected. If an intent was rejected, the user could provide feedback to correct it, thus improving LUMI. Such a corrected intent could then once again be accepted or rejected by the user. The right-hand side of Figure 3.7

gives the breakdown of the intents and shows that, most of the time, LUMI correctly interpreted the users' intents; in the few times feedback was needed, LUMI was able to correct and learn more often than not. This result is encouraging given the somewhat limited amount of data with which LUMI was trained.

On further analysis of the interactions that users had with LUMI, we observed that the feedback mechanism worked as expected in cases where the participants used a different or unusual phrasing of their intents. For instance, one user expressed the intent for task 3 in an unstructured manner, as "*student quota dorms 10GB week*", in which LUMI was not able to recognize the word "*dorms*" as a @*location* entity. However, the user then provided feedback that was successful in correcting the original *Nile* intent.

One concrete example where the feedback was unsuccessful happened in task 3, with a user that typed the intent "*Lumi, limit students to 10GB maximum download per week in dorms*". Lumi was only trained to recognize phrases of the form "10GB per week", and the additional text in between resulted in Lumi being unable to recognize the user's phrase. When asked what information LUMI had missed, the user provided feedback indicating that "10gb/wk" was an entity class and "Gb per week" was the value, instead of labeling "Gb per week" as a @*qos_unit* entity. We note that Lumi is an initial prototype, and such cases can be avoided in the future by improving the clarity of suggestions LUMI makes to the user, and by including sanity checks on user inputs.

3.7.3 Users Reactions and Usability

In the post-questionnaire, we asked the users to comment on LUMI's usability and overall merit by answering three questions: (i) How easy was it to use LUMI to configure the network? (ii) Compared to traditional policy configuration, how much better or worse was using LUMI instead? and (iii) Would they rather use LUMI to enforce policies or conventional network configuration commands. Figures 3.8, 3.9 and 3.10 summarize the users' responses, broken down by expertise in network management. The results show that the participants' overall reaction to LUMI was very positive, with most of them stating that they would either use LUMI exclusively or, depending on the tasks, in conjunction with configuration commands. Note that the expert users who identified themselves as campus network operators all had a positive reaction to LUMI. Overall, among all different levels of expertise, 88.5% of participants stated they would rather use LUMI exclusively or in conjunction with configuration commands.

We also asked participants to provide insights they had that could help us improve the implementation and design of LUMI. One important feedback we received was the lack of support for querying the network state. For example, one participant stated:

> "*Many network management tasks are about monitoring something or figuring out what's happening, not just installing a new policy. I didn't see any such tasks when using Lumi in this experiment. (e.g., 'Can these two endpoints communicate right now?', 'are there any packet drops between these two endpoints?').*"

While LUMI was not designed with this goal in mind, we do not foresee any major NLP challenge to incorporate such features into it, as the entity set could be extended to cover this use case. Acquiring the state information from the network requires further investigation, but Lumi's modular design makes it simpler to plug in a new module or an existing one (BIRKNER *et al.*, 2018) to query the network devices. This would enable LUMI to "understand" changes made through other tools, as LUMI will likely coexist with different management applications in real deployments. Overall, the feedback received from participants was positive and highlighted the promise and value of LUMI.

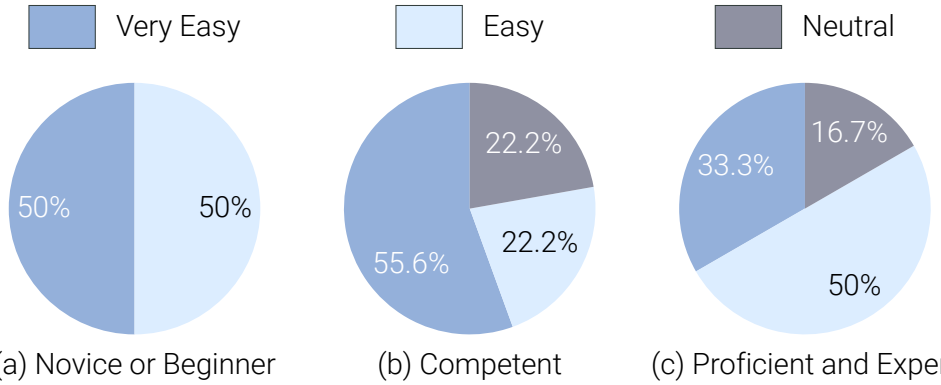


Figure 3.8 – User reaction to LUMI's usability, by expertise on network management.

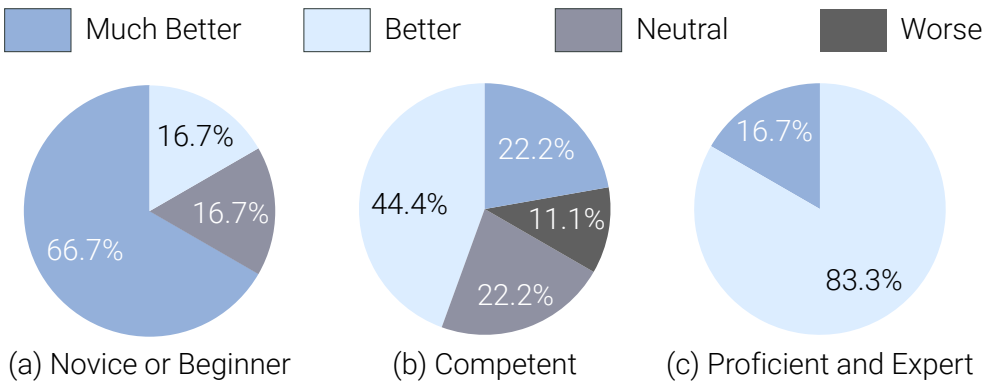


Figure 3.9 – User reaction to LUMI's when compared to traditional network configuration, by expertise on network management.

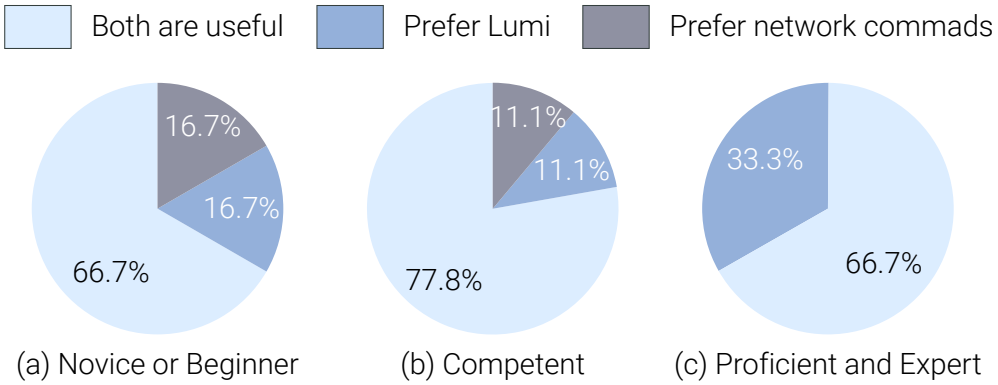


Figure 3.10 – User reaction to LUMI's when compared to traditional network configuration, by expertise on networking.

3.8 Chapter Summary and Remarks

In this chapter, we focus on answering the first research question, which concerns the level of trust operators can deposit in the ML models used in self-driving networks to break down high-level intents into network configurations automatically. To that end, we propose LUMI, a novel end-to-end intent refinement and deployment system that allows operators to express their intents in natural language and then check and confirm the intents before deploying them in the network. By demonstrating that LUMI can successfully deal with a wide range of network policies, this chapter represents a promising step towards realizing the vision of intent-based networking with natural language. Still, much work remains. For example, while our design choices for LUMI's different modules resulted in a working prototype, other features might be necessary for a production-ready system version. However, LUMI's modular design can readily accommodate such improvements. Also, since LUMI in its current form is mainly intended for use in campus networks, supporting other environments (*e.g.,* home or enterprise networks) will most likely require that the set of *Nile* operations and functions (and in turn the group of LUMI entities) be judiciously expanded.

4 MACHINE LEARNING FOR NETWORK SECURITY

Compared to computer vision or autonomous car technology, where AI and ML have been adopted early on and with enormous success, the networking area has been relatively late in joining the AI/ML fray. Network operators and security experts have remained doubtful about adopting AI/ML-based solutions and deploying them in their production networks, be it to help with network performance-related issues or address network security-specific problems. In particular, achieving the vision of a self-driving network, where the network itself is capable of learning and adapting according to usage patterns from traffic, makes AI/ML-based solutions a critical stepping stone and an integral part of the second management loop highlighted in Figure 4.1. It is crucial, then, for us to verify if the skepticism from operators and experts is unwarranted or not, which is the subject of the second research question.

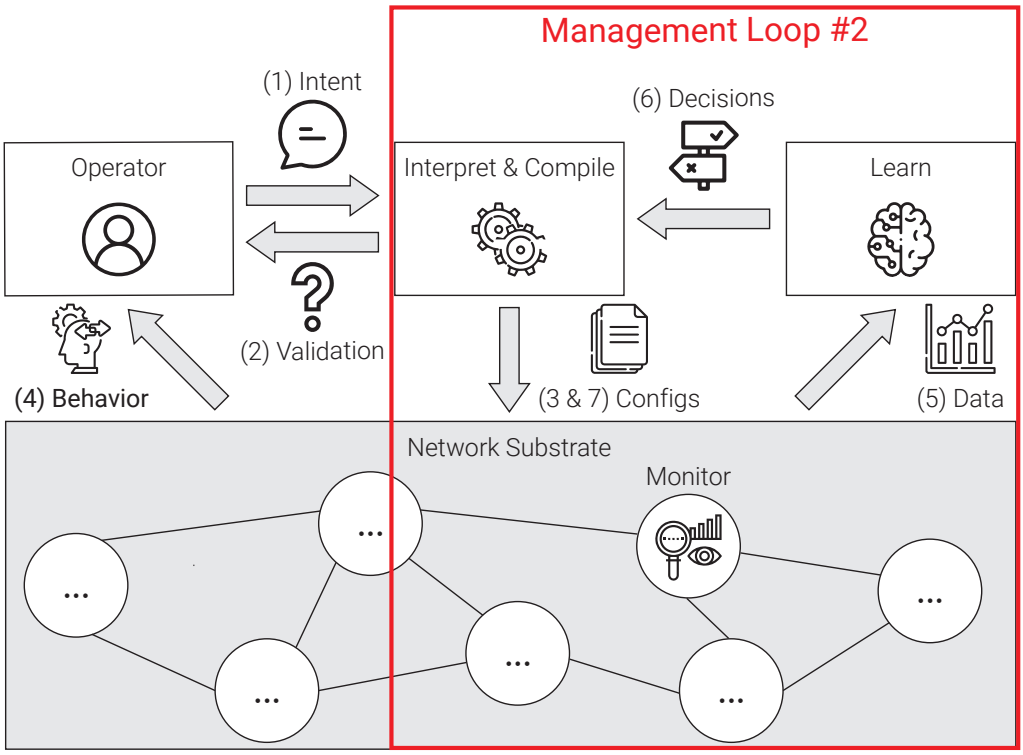


Figure 4.1 – Second management loop of a self-driving network.

In this chapter, we focus on the application of AI/ML to network security and the challenges that this application domain poses. For one, there exists an obvious mismatch between the black-box nature of some of the most commonly considered AI/ML models and what network practitioners expect from or look for in a new technology like AI/ML. While black-box learning models are inherently incapable of providing insights into their "inner workings" or underlying decision-making process, network operators and network

security experts are particularly keen on gaining a basic understanding of how these proposed models work in practice so they can be trusted in real-world production settings.

At the same time, AI/ML research has, in general, paid little to no attention to reproducibility (*i.e.,* the ability of third parties to independently assess and validate relevant research artifacts), contributing to and accelerating an arguable reproducibility crisis in science (GHOSH, 2019; MACHINERY, 2020; BAJPAI *et al.*, 2019). This lack of reproducibility has also played an essential role in the general distrust that network practitioners have on using AI/ML-based solutions in practice. Importantly, it prevents the basic evaluation and assessment of whether or not newly-developed AI/ML research artifacts generalize across different network environments or are only valid for the specific setting from which the underlying training data was obtained in the first place. At the same time, many reproducibility efforts suffer from a fundamental and widely-lamented lack of relevant data in general and labeled data in particular. This is especially true for an application domain like network security, where suitable training datasets often contain highly sensitive information that generally prevents third-party sharing of the raw data. While certain anonymized versions of the data may be suitable for sharing, most data anonymization efforts result in a loss of potentially valuable information, which often impedes subsequent reproducibility efforts.

We argue in this chapter that for AI/ML to be embraced by network practitioners, have an impact in practice, and realize the vision of a self-driving network, new research efforts are needed to re-invent how AI/ML should be used in an application domain like network security. In particular, we propose a two-pronged approach that focuses squarely on addressing the trust issue and comprises the following two critical efforts: *(i)* leveraging **explainable or interpretable AI/ML models** that can take the form of simple decision trees purposefully extracted from commonly-considered black-box models (ADADI; BERRADA, 2018); and *(ii)* requiring the **reproducibility of developed AI/ML research artifacts** to ensure safe third-party sharing of newly-designed learning models/algorithms and avoid the problem-ridden third-party sharing of sensitive data altogether (GUPTA; MAC-STOKER; WILLINGER, 2019).

To implement the first item of this proposed two-pronged approach in practice, we leverage techniques from the emerging XAI research field to scrutinize ML models. This approach answers the commonly-expressed need for gaining insight into the operations of newly-proposed black-box learning models intended to be used in practice as part of critical decision-making processes. We build on the standard AI/ML research pipeline

currently in use (*i.e.,* define model, train with a dataset, evaluate with train-test split) but extend it in significant ways by elevating extracted white-box learning models to the role of first-class citizens. Such a role is not only justified by the ability of explainable learning models to instill in network practitioners the required level of trust but also by the critical capacity that these types of models have for studying the well-documented problem of *underspecification* of developed black-box learning models (D'AMOUR *et al.*, 2020).

For a given black-box learning model, underspecification refers to the problem of determining whether the learning model's excellent performance (*e.g.,* high prediction accuracy) is indeed due to its ability to encode essential structure of the underlying system or simply the result of some inductive bias encoded by the trained model. We illustrate with examples of published black-box learning models developed for network security-related problems how extracted white-box learning models become potent tools for revealing the presence of such inductive biases in the underlying black-box model. In particular, we focus on biases that are the results of unintended learning strategies such as shortcut learning (also known as the "Clever Hans" effect (LAPUSCHKIN *et al.*, 2019; KAUFFMANN *et al.*, 2020)), spurious correlations, or overfitting (GEIRHOS *et al.*, 2020). In the process, we also demonstrate how suitably extracted white-box learning models can be used to determine whether the corresponding original black-box learning model is *credible*; that is, generalizes as expected in deployment scenarios and can hence be trusted to not only work in theory (*i.e.,* test set is independent and identically distributed – i.i.d. – with respect to the training set) but also in practice (*i.e.,* test set is out-of-distribution with regards to the training set).

Finally, while we are not the first to impress upon networking researchers the critical need to ensure that their published AI/ML research artifacts be reproducible by third parties (*e.g.,* see (GUPTA; MAC-STOKER; WILLINGER, 2019) and references therein), we are also not the first to comment on the unique challenges that the network security area poses as an application domain for AI/ML. In fact, not only do we fully agree with much of previously-expressed skepticism about the use of AI/ML for network security-related problems (*e.g.,* see (SOMMER; PAXSON, 2010; APRUZZESE *et al.*, 2018)), but we argue in this chapter more forcefully that when it comes to the application of AI/ML in its current form for network security, "the emperor has no clothes." More constructively, the purpose of this chapter is to "provide the emperor with at least some clothes," ordinary or unimpressive as they may be.

4.1 What AI/ML for Network Security?

In this section, we motivate our skepticism over existing works on AI/ML for network security and present a network performance use case that guides and motivates our efforts to scrutinize ML models for self-driving networks in general and network security in particular.

4.1.1 On the "Old" AI/ML for Network Security

To get a sense of the lay of the land, we performed a limited survey of the existing literature on applications of AI/ML to network security-related problems. We grouped the encountered efforts according to their ability to be reproducible and identified three different categories:

Learning model is published, and training data is publicly available. Only a minimal number of studies reported in the existing literature fall in this category, including (WANG *et al.*, 2016; SHARAFALDIN.; H. Lashkari.; GHORBANI., 2018) to mention a few. We commend these studies because third-party researchers can carefully scrutinize the research artifacts that make up these efforts. At the same time, we encountered hardly any studies that used the afforded reproducibility opportunities and carefully examined the published research artifacts (see Section 4).

Learning model is published, and training data is not publicly available. This category makes up the vast majority of studies reported in the existing literature. Unfortunately, for competitive, legal, or other reasons, the use of these efforts' training datasets is typically confined to the very researchers who developed the published learning models in the first place, which makes reproducibility of the described research artifacts by third-party researchers impossible (*e.g.,* see (FINAMORE *et al.*, 2010; SCHATZMANN *et al.*, 2010; JING *et al.*, 2011; ZHANGA *et al.*, 2013; SUN *et al.*, 2016)).

Learning model is not published, and training data is not publicly available. This category consists primarily of commercial solutions. These solutions are marketed by an ever-growing number of network security companies that claim to leverage AI/ML-based technologies in their products. The proprietary nature of these products rules out any scientifically rigorous evaluation by third parties and prevents even the most basic attempts at reproducibility (*e.g.,* see (DARKTRACE, 2021; MOLONY, 2020)).

The two main takeaways from this limited survey of the existing literature on the

application of AI/ML for network security are *(i)* reproducibility efforts are the exception rather than the rule, and *(ii)* the literature on AI/ML for network security is still short of efforts that apply explainable AI/ML, at least at a level comparable to the use case we describe in detail in Section 4.1.2 below. Notably, the general lack of reproducibility efforts makes it impossible to carefully scrutinize most published learning models developed with network security-specific tasks in mind. This observation highlights the current precarious state-of-the-art in AI/ML-based network security research, where the reported (typically superb) performance of learning models across all three categories cannot be taken at face value and where their ability to generalize as expected in deployment scenarios remains at best unknown and is at worse more than questionable.

4.1.2 An Illustrative Use Case

In the absence of a compelling published study that concerns an application of modern AI/ML to a network security problem and allows us to illustrate the issues we raise in this chapter, we find inspiration in a recent effort that concerns the use of modern AI/ML for a network performance problem.

The effort in question deals with a recently proposed AI/ML-based system for adaptive bitrate (ABR) video streaming and comprises three different published papers (MAO; NETRAVALI; ALIZADEH, 2017; AL., 2019; MENG *et al.*, 2020). The original paper (MAO; NETRAVALI; ALIZADEH, 2017) presents Pensieve, a new black-box learning model that combines (deep) neural network models with reinforcement learning (RL) to learn an optimized control policy for bitrate adaption in a data-driven and automatic manner. From our perspective, a relevant aspect of this work is that the authors of (MAO; NETRAVALI; ALIZADEH, 2017) open-sourced all Pensieve-related research artifacts, including data, learning model, and code, to encourage full reproducibility of their work. The second paper (AL., 2019) describes an initial attempt at "cracking open" Pensieve's black-box learning model and relies critically on the ability to reproduce the artifacts developed in (MAO; NETRAVALI; ALIZADEH, 2017). By applying a set of previously developed techniques that are commonly referred to as *local interpretability/-explainability* tools (*e.g.,* see (RIBEIRO; SINGH; GUESTRIN, 2016; MOLNAR *et al.*, 2020)), the authors of (AL., 2019) illustrate several findings that show the potential that local explainability tools have in increasing the network operators' trust in AI/ML-based systems like Pensieve.

Unfortunately, local explainability tools like the ones employed in (AL., 2019) are limited in their capabilities and fall short in explaining the behavior of a given black-box learning model as a whole (*i.e., global explainability*). The main contribution of the third paper (MENG *et al.*, 2020) is exploring the use of such global explainability tools to examine Pensieve's black-box learning models. To this end, the authors of (MENG *et al.*, 2020) build on recent advances in the area of explainable AI/ML (*e.g.,* see (BASTANI; KIM; BASTANI, 2017; LAKKARAJU *et al.*, 2017)) and extract a "white-box" learning model in the form of a decision tree from the black-box learning model used by Pensieve. A basic examination of the resulting highly structured and eminently interpretable decision tree demonstrates its ability to explain Pensieve's black-box learning model and the learned policies extracted by this method. Their examinations corroborate the findings of the second paper's local explainability analysis that Pensieve favors three out of the six different adaptive bitrates most of the time, relying mostly on the last selected bitrate and buffer size to choose.

In summary, the Pensieve example (MAO; NETRAVALI; ALIZADEH, 2017) convincingly demonstrates the merits of reproducible AI/ML-based networking research. The subsequent efforts, which first used local explainability tools to "look under the hood" of Pensieve's black-box learning model (AL., 2019), culminated in applying a global explainability method capable of distilling Pensieve's incomprehensible but high-performing learned policies into simple and highly interpretable decision tree policies (MENG *et al.*, 2020). These efforts could only succeed thanks to the open-source nature of the research artifacts that the authors of (MAO; NETRAVALI; ALIZADEH, 2017) released as part of their work. However, we show in the rest of the chapter that there is more to explainability than these Pensieve-specific efforts indicate when applying them to the network security problem domain.

4.2 A New AI/ML Pipeline

Motivated and inspired by the Pensieve's illustrative use case, we propose a new workflow for how modern AI/ML can be used in the application domain of network security. We describe the proposed workflow in terms of a new AI/ML pipeline that elevates suitably extracted white-box learning models to the role of first-class citizens, presented in Figure 4.2.

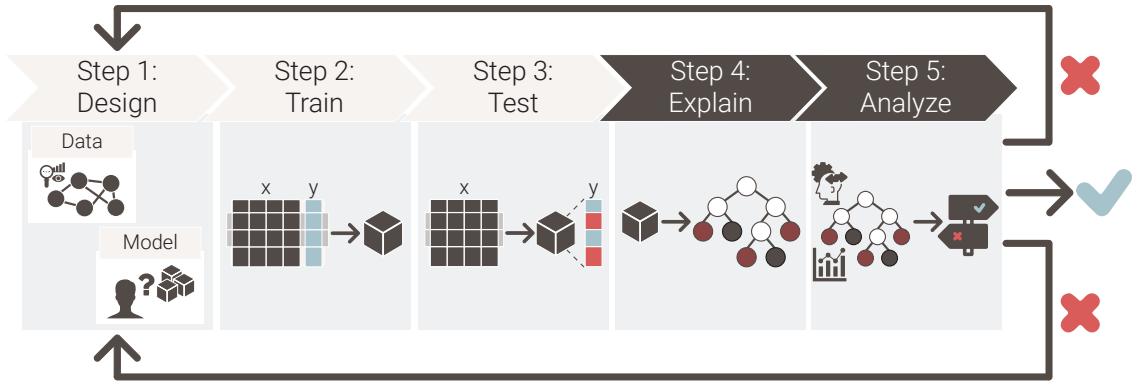


Figure 4.2 – New AI/ML pipeline.

- **Step 1**: Defining a specific training task and its description in terms of a chosen model specification or algorithm, and data collection and preparation to build a training dataset;
- **Step 2**: Using a provided training dataset and the selected algorithm to train a black-box learning model;
- **Step 3**: Evaluating the obtained learning model by traditional procedures that measure the model's performance using a dataset)*i.e.,* typically using a train-test split) that has the same distribution as the given training dataset;
- **Step 4**: Extracting a high-fidelity, post-hoc white-box learning model (*e.g.,* decision tree) from the black-box learning model obtained in Step 2 and evaluated in Step 3; and
- **Step 5**: Examining the white-box learning model obtained in Step 4 to determine whether or not the black-box learning model obtained in Step 2 and evaluated in Step 3 is credible and can be trusted.

The first three steps of this new AI/ML pipeline comprise the standard AI/ML pipeline currently in use, mainly consisting of traditional procedures such as model selection, data understanding and preparation, and model evaluation. In turn, the newly added Steps 4-5 capture the requirement of a growing number of AI/ML applications for real-world problems (*e.g.,* network security). This requirement demands to move beyond the single-issue objective of creating high-performing black-box learning models and strive instead for a more holistic assessment of them that focuses on aspects commonly associated with "trust", including interpretability, credibility, and effectiveness. To this end, Step 4 is intended to produce an appropriate post-hoc white-box learning model that, for all practical purposes, serves as a high-quality substitute for the underlying high-performing but incomprehensible black-box learning model. This explainable learning model is then used in Step 5 to examine important properties (*e.g.,* fairness, safety, robustness) of the obtained black-box learning model, its learned policies, or the underlying

training data. In effect, our newly proposed pipeline functions as a loop, as the problems uncovered by the added Steps 4-5 will likely trigger correcting actions in previous steps, most notably in data preparation and model design. In particular, we view this modified AI/ML pipeline as a means to provide transparency of and establish trust in black-box models to the point where it can become a primary vehicle for answering questions concerning the ethical quality of black-box based decision making (PIANO, 2020).

Notably, the steps in our proposed AI/ML pipeline share similarities with the steps in the Knowledge Discovery in Databases (KDD) (FAYYAD; PIATETSKY-SHAPIRO; SMYTH, 1996), a typical data mining process to identify patterns in data. While KDD can rely on AI/ML techniques to identify such patterns (*e.g.,* clustering and regression), the goal of KDD is usually to recognize trends and correlations in business-related ventures, such as sales and usage, and not classify or predict labels such as in networking applications. Still, given that KDD can use AI/ML to identify patterns, it could also benefit from the added interpretability and explainability introduced by the new steps in the AI/ML pipeline.

The proposed modification begs the question of why not train a white-box learning model in the first place (*e.g.,* see (RUDIN, 2019)). Some of the reasons discussed in the existing literature include: *(i)* an inherent tradeoff between accuracy and explainability (*i.e.,* complex black-box models are necessary to achieve top predictive performance) and *(ii)* important differences between problems with structured data and meaningful features vs. unstructured data and non-intuitive features (*i.e.,* designing ex-ante interpretable models for unstructured data and non-intuitive features with good performance is challenging). We add to this list our own reasoning; that is, in the network security area, even the top domain experts admit that their domain knowledge is bounded and that fully comprehending the ever-growing complexity of today's networks is no longer within their grasp. In particular, we argue that the network security domain is a prime example where domain knowledge has ballooned to a point where developing ex-ante white-box learning models with high performance is, in general, beyond the grasp of domain experts.

In designing our newly proposed AI/ML pipeline and having reviewed the existing literature and techniques in the XAI field in Section 2.4, we realized there was no method available that would allow us to fulfill the new Steps 4 and 5. This realization prompted us to design a novel method to extract global explanations from any given black-box model in the form of decision trees, which we called TRUSTEE. Alongside TRUSTEE, we also introduce a novel domain-targeted pruning method, which we call Top-k pruning, to keep

extracted decision tree explanations to a concise and manageable size while maintaining a good fidelity tradeoff, and a *trust report* that condenses and analyzes the most relevant aspects of the TRUSTEE's explanation to aid domain experts in spotting issues with the black-box model. These novel methods allowed us to accomplish our new AI/ML pipeline and continue to answer this thesis's second research question in the remainder of this chapter. We describe TRUSTEE and Top-k pruning in detail, alongside the answer to the third research question this thesis poses, in Chapter 5.

4.3 Use Cases

In this section, we apply TRUSTEE to scrutinize a few recently published black-box models that have been developed for network security-related problems and are accompanied by publicly available artifacts that are required for assessing whether the models are credible. By scrutinizing these few key works, we aim to provide an answer to answer to our second research question and determine whether operators are right to be skeptical of ML-based solutions in networking. All datasets, models, and results presented in this section are available at (JACOBS *et al.*, 2022b).

4.3.1 Summary

Table 4.1 summarizes the use cases we analyze. The first use case (§4.3.2) 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 (§4.3.3) analyzes a black-box model (*i.e.,* random forest) trained using the popular synthetic dataset CIC-IDS-2017 (SHARAFALDIN.; H. Lashkari.; GHORBANI., 2018) and shows that the developed model is vulnerable to o.o.d. samples. This use case cautions against an over-reliance on synthetic datasets that often include measurement artifacts that commonly-considered black-box models exploit to achieve high accuracy. The third use case (§4.3.4) analyzes a recent approach that advocates using bit-level feature representations of the input data instead of carefully engineered and semantically meaningful features (HOLLAND *et al.*, 2021). This use case warns against the indiscriminate use of the high-dimensional feature spaces that result from such representations because they

Table 4.1 – Case Studies.

Analyzed in	Problem	Dataset(s)	Model(s)	Trustee Fidelity	Type of inferred inductive bias
Section 4.3.2	Detect VPN traffic	ISCX VPN-nonVPN dataset (DRAPER-GIL. *et al.*, 2016)	1-D CNN (WANG *et al.*, 2017)	1.00	Shortcut learning
Section 4.3.3	Detect Heartbleed traffic	CIC-IDS-2017 (SHARAFALDIN.; H. Lashkari.; GHORBANI., 2018)	RF Classifier (SHARAFALDIN.; H. Lashkari.; GHORBANI., 2018)	0.99	Out-of-distribution samples
Section 4.3.4	Detect Malicious traffic (IDS)	CIC-IDS-2017 (SHARAFALDIN.; H. Lashkari.; GHORBANI., 2018), Campus dataset	nPrintML (HOLLAND *et al.*, 2021)	0.99	Spurious correlations
Section 4.3.5	Anomaly Detection	Mirai dataset (MIRSKY *et al.*, 2018)	Kitsune (MIRSKY *et al.*, 2018)	0.99	Out-of-distribution samples
Section 4.4	OS Fingerprinting	CIC-IDS-2017 (SHARAFALDIN.; H. Lashkari.; GHORBANI., 2018)	nPrintML (HOLLAND *et al.*, 2021)	0.99	Potential out-of-distribution samples
Section 4.4	IoT Device Fingerprinting	UNSW-IoT (SIVANATHAN *et al.*, 2019)	Iisy (XIONG; ZILBERMAN, 2019)	0.99	Likely shortcut learning
Section 4.4	Adaptive Bit-rate	HSDPA Norway (RIISER *et al.*, 2013)	Pensieve (MAO; NETRAVALI; ALIZADEH, 2017)	0.99	Potential out-of-distribution samples

allow black-box models to identify and exploit spurious correlations between features to achieve high accuracy. The fourth use case (§4.3.5) concerns the application of a complex ensemble of neural networks (MIRSKY *et al.*, 2018) to perform traffic anomaly detection (*e.g.,* Mirai attack). By showing that this model is also vulnerable to o.o.d. samples, we corroborate previously-reported criticism of this model (ARP *et al.*, 2022) and support it with further evidence. Finally, we briefly describe our results from applying TRUSTEE to a few other networking and network security use cases (§4.4).

4.3.2 Detecting VPN Traffic

Problem setup. We consider the paper (WANG *et al.*, 2017), which presents an AI/ML-based framework for encrypted traffic classification that integrates feature design, features extraction, and 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 (DRAPER-GIL. *et al.*, 2016), 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". (WANG *et al.*, 2017) reports outstanding performance (*i.e.,* 100% (99.9%) precision and 99.9% (100%) recall for Non-VPN (VPN) traffic). All AI/ML research artifacts (WANG *et al.*, 2017) and datasets (DRAPER-GIL. *et al.*, 2016) 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 (WANG *et al.*, 2017, Table VI) for classifying VPN vs. Non-VPN

traffic. Next, we used TRUSTEE to extract a decision tree from the black-box 1D-CNN model (Figure 4.3). 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 decision tree reproduces the black-box model, we used it to classify the test cases from (WANG *et al.*, 2017) 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.

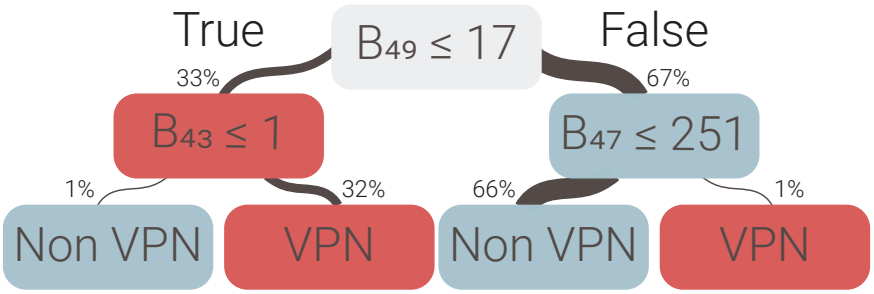


Figure 4.3 – Decision tree for black-box 1D-CNN model. The percentage of samples that follow each branch is presented above each node. Line widths are proportional to the percentage of samples.

Correctly interpreting this extracted decision tree requires understanding the structure of the input data. Because the decision tree 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 4.4 shows a schematic view of the first 80 bytes of actual input data used in (WANG *et al.*, 2017). 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.

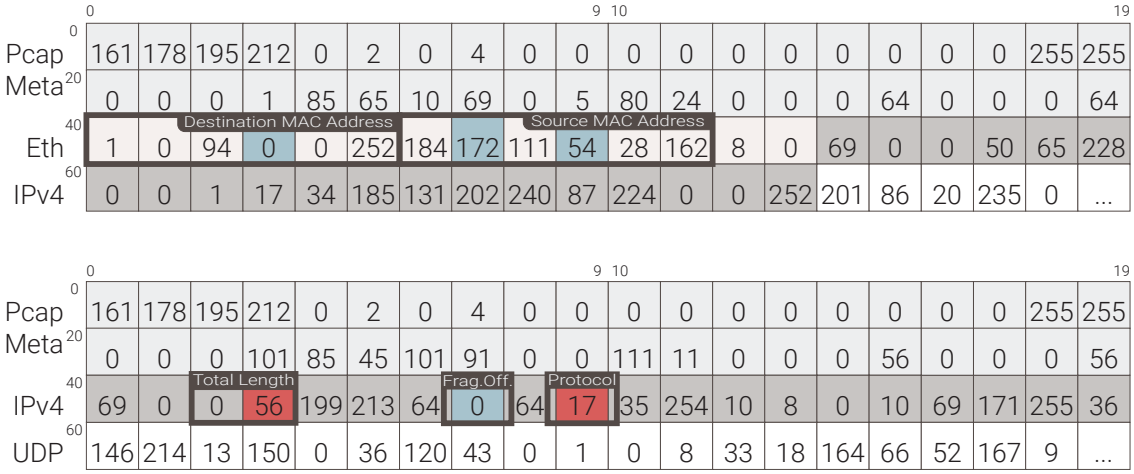


Figure 4.4 – First 80 bytes from the training dataset.

Upon further scrutiny of the public dataset (DRAPER-GIL. *et al.*, 2016), we noticed that Non-VPN traffic samples always contain Ethernet headers while roughly 90% of the VPN traffic samples do not

The left branch of the decision tree classifies most samples as VPNs. However, to weed out a few remaining samples of Non-VPN traffic, the decision tree 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} (Fragment Offset in Non-VPN vs. the second byte of Ethernet source address in VPN) to weed out the few VPN samples.

Validation. Even though the extracted decision tree 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 misclassify all VPN samples. The first two rows of Table 4.2 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.

Table 4.2 – 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 further demonstrate that the black-box model described in (WANG *et al.*, 2017) and claimed 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 indi-

vidual bytes. As Table 4.2 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.

4.3.3 Detecting Heartbleed Traffic

Problem Setup. We consider the paper (SHARAFALDIN.; H. Lashkari.; GHORBANI., 2018), 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 (DORIGUZZI-CORIN *et al.*, 2020; BUSSE-GRAWITZ *et al.*, 2019; DWIVEDI; VARDHAN; TRIPATHI, 2020; SHARAFALDIN.; H. Lashkari.; GHORBANI., 2018; ZHANG *et al.*, 2020).

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 (LEMAîTRE; NOGUEIRA; ARIDAS, 2017; KAUR; PANNU; MALHI, 2019) to produce a balanced training dataset to retrain the Random Forest Classifier and then used TRUSTEE to extract a decision tree explanation. Without applying our Top-k pruning method, the high-fidelity decision tree extracted by TRUSTEE from the classifier contained 899 nodes, making it unfeasible to understand the decision-making process of the black-box model. However, when running the full-fledged version of TRUSTEE (*i.e.,* with the Top-k Pruning method with $k = 3$), we obtain the small-sized and therefore inherently manageable decision tree shown in Figure 4.5.

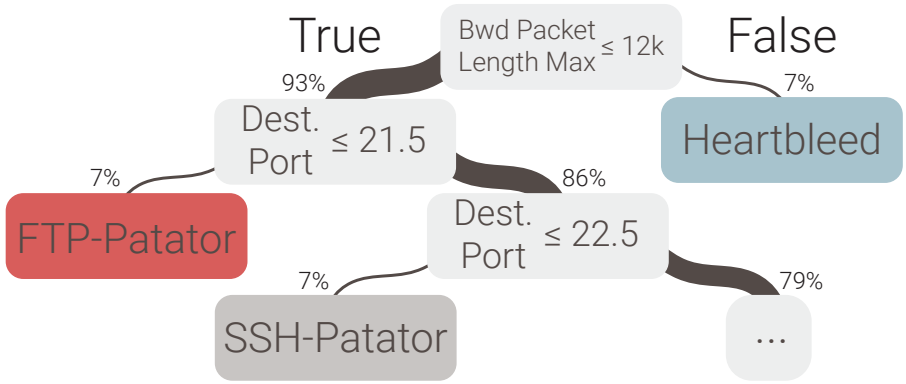


Figure 4.5 – Decision tree for Random Forest Classifier. The percentage of samples that follow each branch is presented above each node. Line widths are proportional to the percentage of samples.

Despite the likely shortcut the model takes by using TCP ports to classify SSH, and FTP-Patator attacks, the root node of Figure 4.5 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 (DURUMERIC *et al.*, 2014). Prompted by this observation, we further inspect the decision tree to identify other most dominant features after removing 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 4.6) 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".

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 (SHARAFALDIN.; H. Lashkari.; GHORBANI., 2018), 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

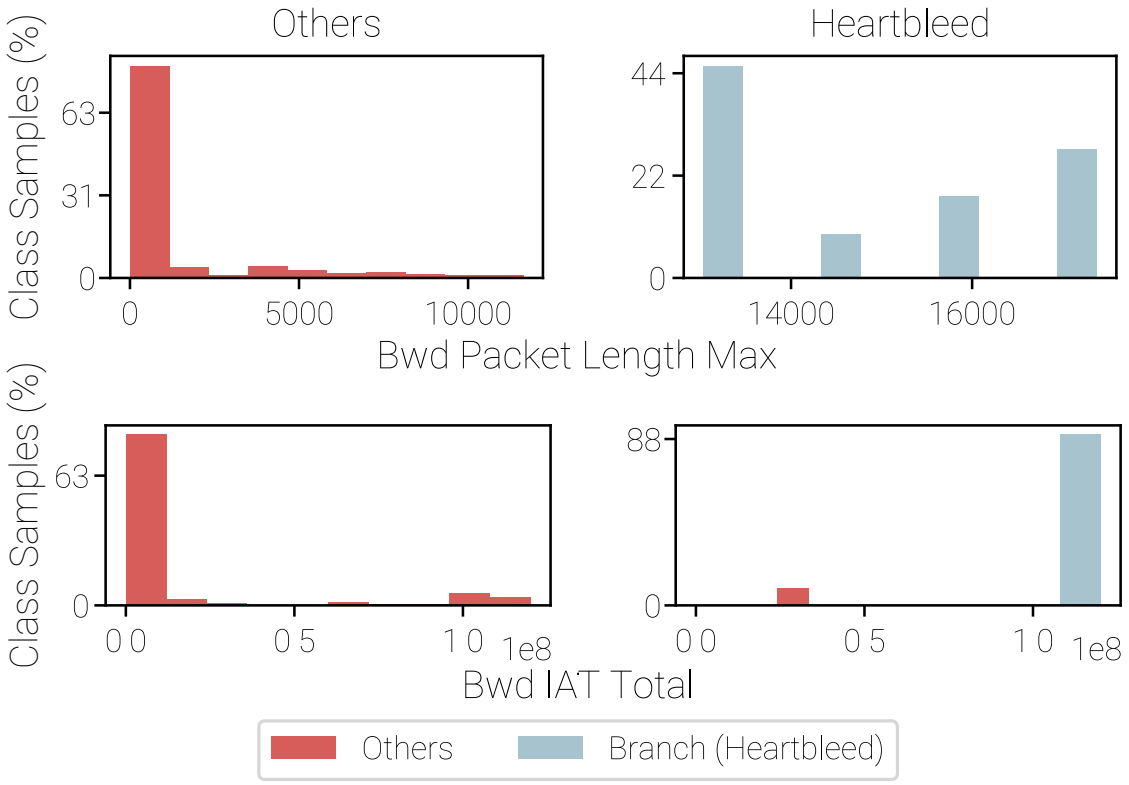


Figure 4.6 – Data distribution of feature "Bwd Packet Length Max" (top) and "Bwd IAT Total" (bottom) comparing values in the Heartbleed class to all Others.

the exact packet sizes for the original trace. We then evaluated the Random Forest Classifier using the newly generated Heartbleed flows as test data. Table 4.3 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 reflects what readily available domain knowledge tells us about Heartbleed attacks.

Table 4.3 – Black-box classifier's accuracy.

Class	Precision	Recall	F1
Heartbleed (i.i.d.)	1.000	1.000	1.000
Heartbleed (o.o.d.)	0	0	0

4.3.4 Inferring Malicious Traffic for IDS

Problem setup. We consider the paper (HOLLAND *et al.*, 2021), which proposes nPrint and the stable bit-level representation of network packets for automatically training learning models using AutoML (ERICKSON *et al.*, 2020). 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 and payloads. The value -1 represents bits 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 (SHARAFALDIN.; H. Lashkari.; GHORBANI., 2018)

Explanation. We successfully reproduced the reported results using the same configurations as those used in (HOLLAND *et al.*, 2021), obtaining a model with a 0.999 F1 score. To investigate this high-performance model, we generated a high-fidelity (0.999) decision tree with TRUSTEE (including running the Top-k pruning method with $k = 4$) and show the top-4 branches in Figure 4.7. 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. The description of the setup used to generate the CIC-IDS-2017 dataset reveals why. All attacks were generated using hosts outside of the network in which the dataset was collected, while the benign traffic was from hosts inside the network, creating a strong correlation between the TTL value and a traffic type. Also, most attacks were generated by a host with Kali Linux, which sets the initial value for TTL to 64 (*i.e.,* 00**1**00000), while the DDoS attack was generated using a host with 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 easier for the model to separate all DDoS attacks using only the second and third most significant bits of the TTL field.

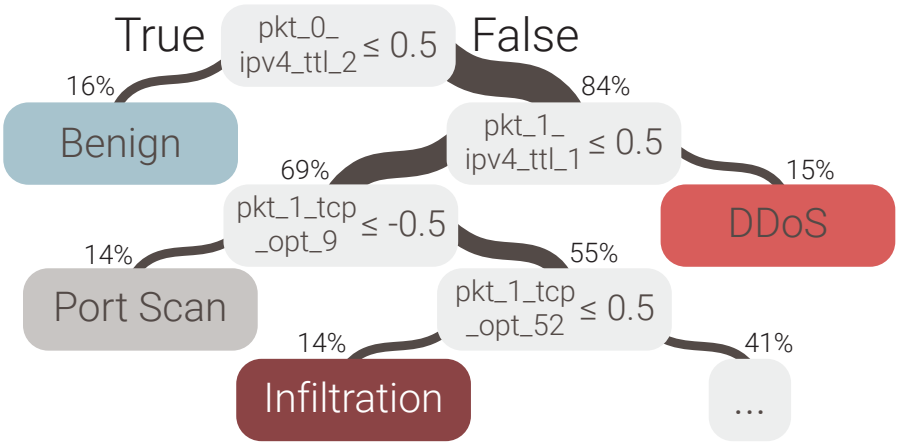


Figure 4.7 – Decision tree for nPrintML IDS model. The percentage of samples that follow each branch is presented above each node. Line widths are proportional to the percentage of samples.

We used the extracted decision tree to investigate the model's behavior further. 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 almost perfectly separates the attacks in the CIC-IDS-2017 dataset, reaching a 0.990 F1 score. The decision tree explanations produced by TRUSTEE showed that the model still used single bits or packets to divide the traffic perfectly. That indicates that the model overfits to spurious correlations of the dataset, finding shortcuts due to the vast feature vector where each bit is a feature. This issue is also known as the "curse of dimensionality" (RUSSELL; NORVIG, 2020), which happens when a model faces a high-dimensional feature space (such as the 4,480 features per sample in the nPrintML IDS) and not a diverse enough data distribution to account for the multiple shortcuts.

Validation. To examine the ability of the nPrintML IDS model to generalize to other networks, we deployed the Suricata Intrusion Detection System (FOUNDATION., 2018) in the campus network of the University of California - Santa Barbara (UCSB) 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 Appendix A 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 did not trigger any alert, which we labeled benign traffic. Finally, we used nPrint to create a test dataset from that traffic and validate the trained model of (HOLLAND *et al.*, 2021). Table 4.4 shows the classification results of the model for the trace of our campus network.

Table 4.4 – Accuracy for black-box model trained in (HOLLAND *et al.*, 2021) 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 noticed 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 attacks, we were intrigued that it correctly classified some port scans. Inspecting the decision presented in Figure 4.7, 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 (HOLLAND *et al.*, 2021) 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 (SHARAFALDIN.; H. Lashkari.; GHORBANI., 2018)). Thus the simple rule that the second packet of a flow is missing would be enough to find all port scans. However, in the trace of our campus network, 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.

4.3.5 Anomaly Detection for Mirai Attacks

Problem setup. We analyzed the paper (MIRSKY *et al.*, 2018), where the authors present Kitsune, an unsupervised ML classifier for anomaly detection. Kitsune comprises an ensemble of auto-encoders and neural networks and solves a regression problem in practice. It receives a set of 115 statistical features (*e.g.,* mean and standard deviation), calculated incrementally for a stream of packets for different levels of aggregation (*e.g.,* by source MAC and IP addresses). It outputs the Root Mean Squared Error (RMSE) as an anomaly score by reconstructing the input features from the ensemble output. Kitsune is trained for some time under normal traffic conditions before moving to an execution phase to detect anomalies. The larger the RMSE, the bigger the anomaly detected by Kitsune. Hence, the authors propose that operators use a threshold-based approach calculated on the training data to detect an anomaly.

Kitsune relies on dampened incremental statistics over time windows, where all features are calculated based on *weights*. The *weight* feature corresponds to the current packet count multiplied by a decay factor so that the weight of older features decreases over time, akin to a sliding window. Kitsune uses a set with five different time windows (100ms, 500ms, 1.5sec, 10sec, and 1min, represented by a variable λ = 5, 3, 1, 0.1, 0.01, respectively) for which the same 23 features are calculated for each time window, resulting in 115 features. While the work described in (MIRSKY *et al.*, 2018) applies Kitsune to several anomaly detection use cases, a recent study (ARP *et al.*, 2022) pointed

out potential problems with one of these use cases (*i.e.,* the Mirai attack) and prompted us to use TRUSTEE to scrutinize Kitsune's proposed ML model for the Mirai attack.

Explanation. We first executed the Mirai attack-specific experiments described in (MIRSKY *et al.*, 2018) and were able to reproduce the results reported (MIRSKY *et al.*, 2018). The Mirai trace that Kitsune uses for training and evaluation consists of 120 minutes (≈760k packets) of a synthetically generated attack in a network with nine IoT devices, in which the first 70 minutes (≈120k packets) consist of benign traffic and the remaining 50 minutes (≈640k packets) have anomalous traffic. Kitsune is trained on the first 50 minutes of the trace and evaluated on the remainder of traffic. For benign traffic, the largest RMSE computed by Kitsune was approximately 6.9, but for anomalous traffic, this value went up to 14 RMSE. We generated a balanced subset of 300k packets, split between benign and anomalous packets, and used TRUSTEE to extract a high-fidelity (0.99) DT from Kitsune. As Kitsune works as a regression problem, we measure fidelity for this use case as the R-squared value between Kitsune's predictions and those obtained by the DT explanation. Using TRUSTEE with its built-in Top-k Pruning method and setting $k = 3$ results in the small DT explanation that is shown in Figure 4.8 and achieves 0.94 fidelity compared to Kitsune.

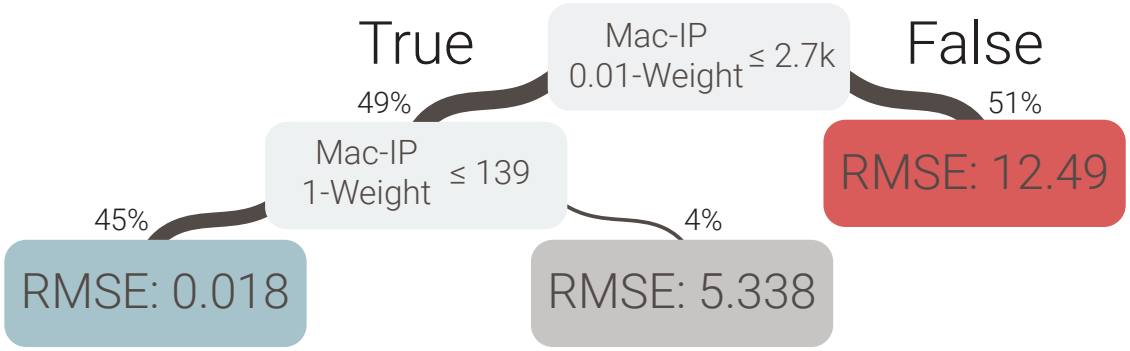


Figure 4.8 – Decision tree for Kitsune Mirai model.

The resulting DT explanation shows that the most prominent features Kitsune uses to determine an anomaly are the *weights*, aggregated by source MAC and IP addresses and associated with two different time windows: 0.01 (1min) and 1 (1.5sec). That is, Kitsune relies mainly on the volume of packets per time frame to determine if an attack is underway. An infected device suffering from a Mirai attack (GALLOPENI *et al.*, 2020) exhibits three main traffic behaviors: *(i)* scanning the network for other vulnerable IoT devices; *(ii)* communicating with the Command and Control (C&C) server, and *(iii)* launching a volumetric DDoS attack from the IoT devices to a target server (usually outside of the infected network). However, the Mirai attack in the synthetic trace used in Kitsune mixes

two of these behaviors: a volumetric scan of the infected IoT network with a flood of ARP requests (about 6x times the amount of packets per second compared to the benign traffic, as shown by top-left plot in Figure 4.9) and a DDoS attack to the target server. This pronounced difference in volume between benign and attack traffic makes it easy for Kitsune to detect anomalous behavior based on traffic volume alone and corroborates the findings in (ARP *et al.*, 2022) where a simple Boxplot method is shown to achieve a performance very similar to that of the complex Kitsune. However, as pointed out (ARP *et al.*, 2022), this difference between malicious and benign traffic for this portion of the Mirai attack is unlikely to be this pronounced in traces collected from real-world networks.

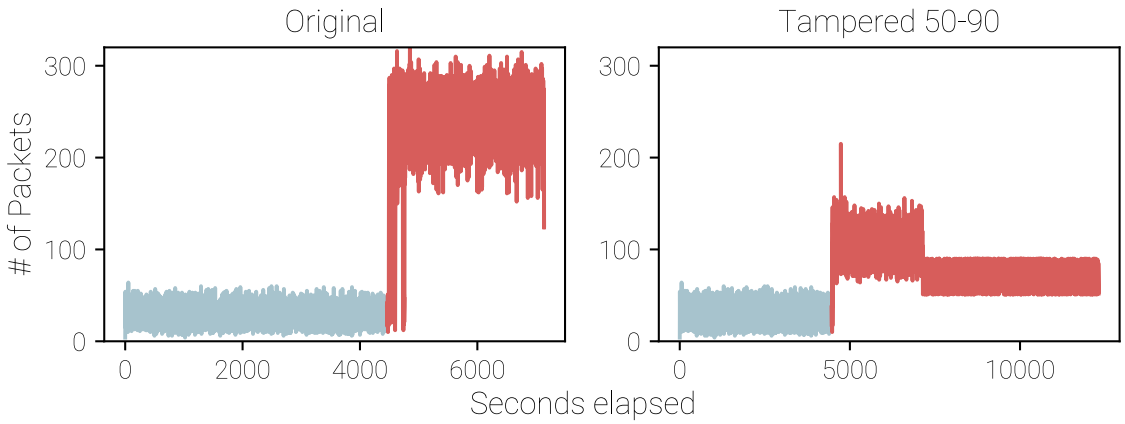


Figure 4.9 – Packets per second for original Mirai trace from Kitsune and tampered trace. Blue segments represent benign traffic and red segments represent traffic with malicious activities (*i.e.,* benign plus attack).

Validation. To validate the DT explanation that TRUSTEE generated, we tampered with the original Mirai trace used in (MIRSKY *et al.*, 2018). We modified the attack portion of the original trace by spacing out ARP requests from the Mirai-infected devices so that the number of packets per second (pps) would not cross a random threshold from a given range of specified limits. In particular, by considering the ranges *(i)* from 10 to 50 pps; *(ii)* from 30 to 70 pps and *(iii)* from 50 to 90 pps, we obtained three distinct tampered traces with different volumes of attack traffic (Figure 4.9 shows the original and one tampered trace). We changed neither the order in which the packets appeared nor the timestamps of ARP responses to avoid interfering with established RTTs. We ran Kitsune for each of these traces, using the same amount of training samples. Figure 4.10 (left) shows the results for each of the traces' first 200k packets in the execution phase of Kitsune. On the right side of Figure 4.10, we also compare the expected RMSE (produced by Kitsune in the original trace) and the predicted RMSE for each tampered trace. The diagonal line (in red) represents the optimal outcome between expected and predicted

RMSE. Hence, the more dots are closer to the line, the less impact our tampering had on the predicted outcome.

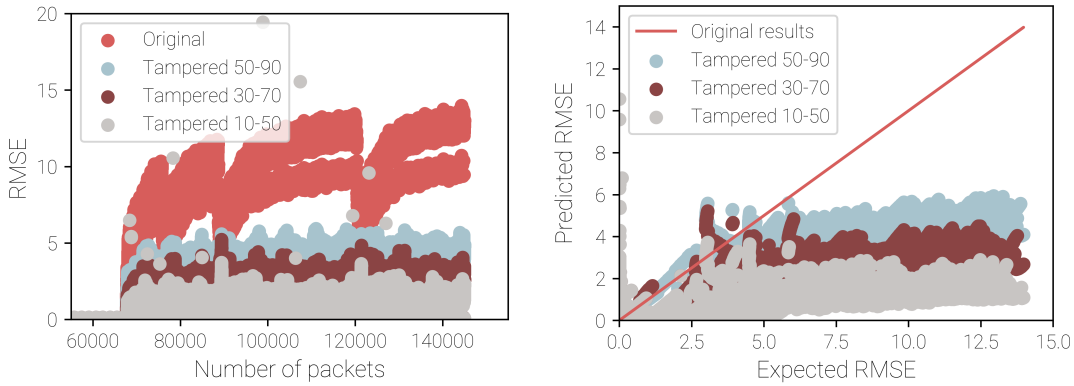


Figure 4.10 – Kitsune execution-phase predicted RMSE results for first 200k packets from original and tampered traces.

The results clearly show that the RMSE values produced by Kitsune depend highly on the volume of the attack traffic encountered, diminishing as the volume decreases, all the way within the values generated for the benign traffic. However, we did notice that our tampering with the original traces made Kitsune produce outliers of RMSE for otherwise benign traffic. While we cannot be sure of the reason for these outliers, since all features calculated by Kitsune depend on the *weight* for each time window, we believe that the changes we made to the attack traffic affected the feature values for the underlying benign traffic. This experiment demonstrates that the Mirai use case from Kitsune is vulnerable to o.o.d. samples, similar to the Heartbleed use case (Section 4.3.3). A simple but realistic change to the Mirai attack pattern made it impossible for Kitsune to accurately detect anomalous behavior. Finally, while our observations point to problems with Kitsune's ability to detect Mirai attacks, they do not imply that Kitsune is unable or unfit to detect other attacks and problems if it uses training data of representative real-world scenarios.

4.4 Other Use Cases

We also used TRUSTEE to analyze several other ML-based models for networking and network security related problems in the literature. For brevity, we describe general outcomes of each of them below. These use cases are also available in our supplemental material (JACOBS *et al.*, 2022b).

OS Fingerprinting Use Case. Aside from the IDS use case presented in the nPrint paper (HOLLAND *et al.*, 2021), we also applied TRUSTEE to scrutinize their OS

fingerprinting application of nPrintML. The authors of (HOLLAND *et al.*, 2021) relied on the same CIC-IDS-2017 published PCAPs and associated OSes from each host in the dataset to build an OS fingerprinting dataset to train nPrintML. The outcome explanation TRUSTEE provided, in general, corroborates the results and claims presented by nPrintML that the model relies on TTL and window size header values to distinguish between Windows, Linux and MacOS OSes. However, we also noted that the black-box model relied on the second most important bit of the TTL header to distinguish Kali Linux OS host machines from regular Linux 3.11 machines. While both OSes have the initial TTL value set to 64, the Kali Linux hosts were positioned outside the network as attackers, and therefore had their TTL value decreased by 2. In the original paper, the authors discard Kali Linux samples as they are an underrepresented class.

IoT Use Case. We considered the paper (XIONG; ZILBERMAN, 2019), where the authors present a method to deploy in-network ML models using P4. In particular, the authors train a Random Forest Classifier to distinguish between different classes of Internet-of-Things devices (*i.e.,* video, audio, etc), using the UNSW-IoT dataset (SIVANATHAN *et al.*, 2019). From the dataset PCAPs, the authors extract 11 features to train the black-box model, including source and destination TCP and UDP ports but no other identifying headers such as IP and MAC addresses. Applying TRUSTEE to the presented black-box model showed that port numbers were the most prominent features to distinguish between all classes, which obviously makes the model very susceptible to mistakes as port numbers change regularly from device to device, and from manufacturer to manufacturer. Iteratively removing TCP and UDP ports from the data showed a decrease in F1 score from 0.99 to 0.63, with frame length taking place as the most relevant feature then. We do note, however, that the author's goal was not to design the best model possible, but to enable its deployment in-network with P4.

Pensieve Use Case. While other works (AL., 2019; MENG *et al.*, 2020) have thoroughly inspected and scrutinized the behavior from the popular Pensive (MAO; NETRAVALI; ALIZADEH, 2017) model to predict adaptive bit-rate, for comparative purposes, we also applied TRUSTEE to produce an explanation decision tree for Pensieve. Our finding mostly corroborate what previous works (AL., 2019; MENG *et al.*, 2020) have noted: *(i)* Pensieve relies heavily on the previous bit-rate choice and the previous buffer size to choose the best adaptive bit-rate; and *(ii)* Pensive rarely picks the 1200kbps or goes higher than 1850kbps bit rates. However, note that the decision tree explanation produced by TRUSTEE achieved a much higher fidelity (*i.e.,* F1 score = 0.90) than

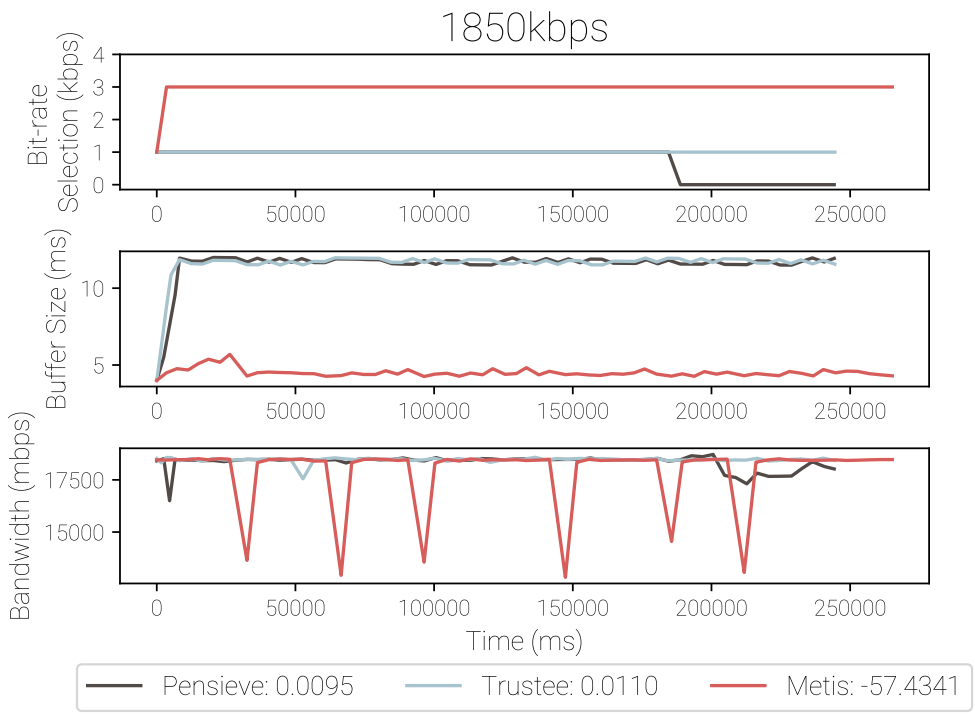


Figure 4.11 – Comparative bit-rate selection results from Pensive, a Metis-generated decision tree and a TRUSTEE generated decision, over a 1850kbps link. The legend shows the overall reward for each method. Results obtained through Pensieve's and Metis' reproduction artifacts.

the reward optimized approach produced by Metis (MENG *et al.*, 2020) (*i.e.,* F1 score = 0.60), at similar explanation sizes. In an experimental deployment of the decision tree to predict bit-rate over different link capacities, we observed that the decision tree produced by TRUSTEE was able to match Pensieve's performance much more faithfully than the Metis-generated decision tree, in particular for link capacities equal and lower than 1850kbps, as illustrated by Figure 4.11.

4.5 Chapter Summary and Remarks

In this chapter, we address the second research question, which concerns the amount of trust operators can deposit in the ML models used in self-driving networks to configure and optimize network devices based on monitored data automatically. In particular, we criticize the current state-of-the-art in AI/ML in AI/ML-based network security research and identify some significant deterrents to applying AI/ML-based network security in practice. We scrutinize three key use-case research efforts representative of the literature by elevating high-fidelity white-box learning models extracted from an underlying black-box learning model to the role of a first-class citizen. As such, the main objective is no longer the development of a high-performance black-box learning model

but a careful examination of whether or not a learned model's typically excellent performance (*e.g.,* high prediction accuracy) is indeed due to the model's ability to encode essential structure of the input data or simply the result of some inductive bias encoded by the trained model. It is only by addressing this so-called problem of underspecification that we can advance AI/ML-based network security research to the point where the credibility and trust of proposed black-box learning models can be rigorously established, and their "inner workings" can be explained to network practitioners.

5 IMPROVING TRUST IN MACHINE LEARNING

In the last few years, we have witnessed a growing tension in the networking community. Recent research has demonstrated the superiority of AI and ML models over simpler rule-based heuristics in identifying complex network traffic patterns for a wide range of network problems (see recent survey articles such as (BOUTABA *et al.*, 2018; XIN *et al.*, 2018; SHAUKAT *et al.*, 2020; National Academies of Sciences Engineering and Medicine, 2019)). At the same time, we have seen reluctance among network operators when it comes to adopting these ML-based research artifacts in production settings (*e.g.,* see (SOMMER; PAXSON, 2010; ARP *et al.*, 2022; APRUZZESE; PAJOLA; CONTI, 2022)). The black-box nature of most of these proposed solutions is the primary reason for this lack of enthusiasm. 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 networking problems but applies more generally to any learning models, especially when their decision-making can have severe societal implications (*e.g.,* healthcare, credit rating, job applications, the criminal justice system, etc.). At the same time, this fundamental 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" (RUDIN, 2019), "explainable AI (XAI)" (TUREK, 2016), and "trustworthy AI" (BRUNDAGE *et al.*, 2020)). 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) (LIPTON, 2018) and also demands solving several fundamental research problems in these new areas of AI/ML.

In this chapter, we address this thesis's third and last research question, which inquires whether or not there is a feasible way to increase the credibility and trust on ML models used in self-driving networks. Based on the insights we gathered from addressing the previous research questions, we focus on analyzing how trust can be improved in the ML models used in the second management loop, continuing our network security use case. Given the results obtained in previous research questions, we argue that the safeguards of human verification and feedback in the first management loop provide operators with an inherently reliable option to check if the ML models are trustworthy and correct them if necessary. Similar to the computer vision research field, the human

presence in manually verifying classifications prevents faulty ML models from deploying system-breaking configurations in the network.

For this thesis, and to guide the development of this chapter, we equate *"an end user having trust in an AI/ML model"* with *"an end user being comfortable with relinquishing control to the model"* (LIPTON, 2018). Given this specific definition of what it means for an AI/ML model to engender trust, we next address fundamental research challenges related to 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 chapter is on determining whether or not a given AI/ML model suffers from the *problem of underspecification* (D'AMOUR *et al.*, 2020).

As we briefly discuss in Chapter 4, 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 an inherent inability for out-of-distribution (o.o.d.) generalizations (*i.e.,* test data distribution is unknown and different from the training data distribution) which, in turn, often reveals itself in the form of specious learning strategies (*e.g.,* shortcut learning (GEIRHOS *et al.*, 2020) or spurious correlations (ARJOVSKY *et al.*, 2020)). Such inductive biases imply that their presence in trained AI/ML models prevents these models from being credible or trustworthy; that is, they generalize as expected in deployment scenarios. Thus, for establishing the specific type of trust in an ML model considered in this chapter, it is critical to identify these inductive biases, and this chapter takes a first step towards achieving this ambitious goal.

To detect underspecification problems in learning models for network security problems, we develop TRUSTEE (TRUSt-oriented decision TreE Extraction) (JACOBS *et al.*, 2022a), which allows us to effectively implement our proposed AI/ML pipeline from the previous chapter. This framework carefully inspects black-box learning models for the presence of inductive biases. Figure 5.1 shows how TRUSTEE augments the traditional ML pipeline to examine the trustworthiness of a given ML model, by implementing Steps 4 (Explain) and 5 (Analyze) of our new AI/ML pipeline presented in the previous section (Figure 4.2). Specifically developed with the application domain of network security in mind, TRUSTEE takes a given black-box model, and the dataset used to train that model as input and outputs "white-box" models in the form of decision trees.

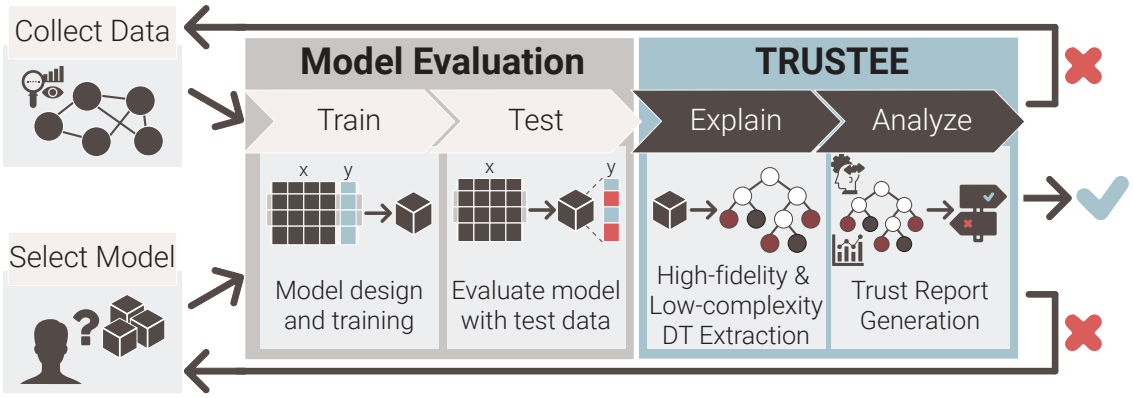


Figure 5.1 – TRUSTEE overview.

Importantly, in synthesizing these decision trees, TRUSTEE's focus is first and foremost on ensuring their practical use, which, in turn, requires leveraging domain-specific observations to strike a balance between *model fidelity* (*i.e.,* accuracy of the decision tree) and *model complexity*. Here, complexity refers to the decision tree's size and aspects of the tree's branches. In particular, when viewing the tree's branches as decision rules, we are concerned with their explicitness/intelligibility and coverage; that is, we require these rules to be readily recognizable by domain experts, be largely in agreement with the experts' domain knowledge, and describe how the given black-box model makes a significant number of its decisions.

TRUSTEE also outputs a trust report associated with the decision tree explanation, which operators can use 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, model selection) that need to be modified 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. First, given the inherent complexity of learning problems for networking, the existing approaches of replacing black-box models with "white-box" models such as decision trees are generally impractical. For example, among existing methods that are concerned with *local interpretability* (*i.e.,* can at best explain the decisions of a trained AI/ML model in a local region near a particular data point; *e.g.,* see LIME (RIBEIRO; SINGH; GUESTRIN, 2016), SHAP (SHAPLEY, 2016), LEMNA (GUO *et al.*, 2018), etc.), are not suitable for examining the various instances of the underspecification problem. Second, though our

effort takes inspiration from prior works that focus on *global interpretability* (*i.e.,* explains how a trained ML model makes decisions as a whole; *e.g.,* see (BASTANI; KIM; BASTANI, 2017; BASTANI; PU; SOLAR-LEZAMA, 2018; LAKKARAJU; BASTANI, 2020)), they are also not suited for the problem at hand. These methods are either only applicable to a specific class of learning models (*e.g.,* reinforcement learning) or suffer from poor fidelity. In their current form, these existing methods are all insufficient for providing the level of model explainability that we demand so that network operators can decide if they are comfortable with relinquishing control to a given black-box model.

Through the case studies presented in the previous chapter, which allowed us to answer our second research question, we already illustrated how operators could use TRUSTEE's decision trees and associated trust reports to detect the presence of existing inductive biases. Hence, our focus in this chapter lies in providing a more in-depth analysis of the TRUSTEE framework itself, showing how it fared when compared to existing global and local explanation methods, and analyzing the stability of the explanations produced by TRUSTEE. More specifically, we revisit two of the presented use cases to show that explanations produced by TRUSTEE achieve a better fidelity-complexity tradeoff than existing methods in the literature. We also present an illustrative use case to show how TRUSTEE's global explanations can avoid common pitfalls of local explanations methods, represented by SHAP in this analysis.

5.1 Problem Statement

This chapter focuses on post-hoc global model interpretability for the application domain of network security problems. Although the idea of using decision trees for investigating global model interpretability for a given black-box learning model has been explored, the set of requirements that we impose on the decision tree explanations we desire is non-standard, making this a challenging problem and motivating us to develop TRUSTEE. For one, we require that our new decision tree extraction method be model-agnostic; that is, applicable to any given black-box model. Second, we also demand that the method produces high-fidelity decision tree explanations; that is, decision trees that accurately describe how the black-box model makes most of its decisions. In addition, we insist that selected parts of the decision trees extracted by our new method are intelligible or comprehensible; that is, easy to understand by domain experts.

Given an extracted decision tree that 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 be used by domain experts to determine whether the given black-box model suffers from the problem of underspecification and can therefore not be trusted. To achieve this goal, we look for ways to exploit the extracted decision tree explanations to enable the end users to investigate the black-box model for signs that indicate the presence of inductive biases. In particular, in this chapter, 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. On the one hand, the presence of any of these inductive biases proves that the given black-box model suffers from the underspecification problem and cannot be trusted. On the other hand, the absence of these instances does not mean that the black-box model can be trusted.

While proving for an arbitrary black-box model that it does not suffer from the underspecification problem is challenging 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 inductive bias, and our work is an initial effort that simplifies demonstrating that certain biases are present in the model. To help in this effort, we consider a simple post-processing step for the extracted decision tree explanation that consists of *(i)* determining the Top-k branches of the tree, *(ii)* presenting them in the form of decision rules that allow for easy inspection, and *(iii)* quantifying the number of decisions of the black-box model that are accurately and intelligibly described by these Top-k branches. This pruning effort is dictated by the very nature of the problem we intend to address, and its results are summarized in a trust report that end users can consult to determine whether or not they can trust the given black-box model.

5.2 Extracting Decision Trees

The first step to realizing the agenda detailed in our overview in Section 5.1 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 present an original and practical method that domain experts can apply to extract high-fidelity decision tree explanations from an arbitrary black-box learning model.

5.2.1 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 (CRAVEN; SHAVLIK, 1995; BASTANI; KIM; BASTANI, 2017; BASTANI; PU; SOLAR-LEZAMA, 2018; MENG *et al.*, 2020) have proposed different approaches to extracting decision tree explanation from a black-box model, but they all lack one or more features that we require when using their extracted decision trees for our purpose (*i.e.,* enabling network security expert to gauge their level of trust in a given black-box model). We list the prior efforts and their pertinent features in Table 5.1. Note that some of these existing methods (BASTANI; PU; SOLAR-LEZAMA, 2018; MENG *et al.*, 2020) are not model-agnostic but have been designed for specific learning paradigms and models, such as Reinforcement Learning (RL). As such, they typically rely on assumptions specific to the learning paradigm or model their designs focus on. For example, in the case of a large class of RL models, it is assumed that the learning problem can be formulated as a Markov Decision Process (MDP) (WHITE; WHITE, 1989) and the reward function intrinsic to RL is then often used as the function that the learning model seeks to optimize. On the other hand, prior efforts that do propose model-agnostic approaches (CRAVEN; SHAVLIK, 1995; BASTANI; KIM; BASTANI, 2017) tend to produce decision tree explanations that do not satisfy the fidelity requirement that we demand for realizing our objective (see Section 5.4 for empirical evidence).

Table 5.1 – Existing approaches to extract decision trees.

Method	Optimization Objective	Stopping Criterion	Model Agnostic	High Fidelity	Domain-specific Pruning
Trepan (CRAVEN; SHAVLIK, 1995)	Fidelity	Max Nodes	✓	-	-
dtextract (BASTANI; KIM; BASTANI, 2017)	Accuracy	Max Nodes	✓	-	-
VIPER (BASTANI; PU; SOLAR-LEZAMA, 2018)	RL Reward	Max Iterations	-	-	-
Metis (MENG *et al.*, 2020)	RL Reward	Max Iterations	-	-	-
TRUSTEE	Fidelity	Max Iterations	✓	✓	✓

Another essential aspect of many of these existing efforts is their stopping criterion to obtain their extracted decision tree explanations. For example, prior efforts such as (CRAVEN; SHAVLIK, 1995; BASTANI; KIM; BASTANI, 2017) require specifying the maximum size (*i.e.,* number of nodes) that the extracted DT can have and use this input parameter as a stopping criterion. Such approaches are convenient for obtaining explanations guaranteed to be of a specific size, but this convenience typically comes at the cost of low fidelity, implying that critical decision-making rules may be missing from

the resulting decision tree. Other methods such as (BASTANI; PU; SOLAR-LEZAMA, 2018) and (MENG *et al.*, 2020) extract decision tree explanations iteratively, require specifying the maximum number of iterations, and use this user-specified input parameter as a stopping criterion. Even though these methods do not explicitly optimize for fidelity, they typically produce high-fidelity explanations 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 (MENG *et al.*, 2020) rely on a commonly-used technique called Cost-Complexity Pruning (CPP) (FRIEDMAN, 2001). Similar to other pruning methods (ESPOSITO *et al.*, 1997), CPP succeeds in striking a balance between the overall fidelity of the extracted decision trees and their size. However, from an interpretability perspective, CPP tends to be oblivious to what roles decision-making rules play as part of the resulting decision tree 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.

5.2.2 Model-Agnostic Decision Tree Extraction

Given the absence of readily available model-agnostic methods for extracting high-fidelity decision tree explanations from black-box learning models, we present in the following TRUSTEE. TRUSTEE takes as input a given black-box model together with the dataset that was used to train the black-box and relies on a teacher-student dynamic derived from imitation learning (HUSSEIN *et al.*, 2017) to extract "white-box" models in the form of decision tree explanations by iteratively maximizing their fidelity. In effect, TRUSTEE uses the given black-box model as an oracle to guide the training of multiple surrogate white-box models (*i.e.,* decision trees) that imitate the black-box's decisions and then selects the explanation with the highest overall fidelity as the "best" white-box counterpart for the given black-box model.

Algorithm 1 describes the different steps that TRUSTEE performs to achieve its objective. The algorithm requires as primary input a black-box model π^* that we desire to explain, the original dataset $\mathcal{D}_0$ that was used to train π^*, the number of samples M to use when training the surrogate decision trees, and the maximum number of iterations N to use when trying to maximize the fidelity of the considered surrogate decision trees. The

complete algorithm requires one additional input parameter (Line 12), but we discuss the utility of this parameter in more detail in Section 5. The algorithm starts by initializing the training dataset $\mathcal{D}$ (Line 2) using the given black-box π^* to predict the expected outcome from the given input data $\mathcal{D}_0$.

During the i-th ($1 \leq i \leq N$) iteration (Lines 3-8), the algorithm: *(i)* uniformly chooses M training samples from the optimal prediction dataset $\mathcal{D}$ (Line 4) to initialize a training dataset $\mathcal{D}'$; *(ii)* splits the subsampled dataset $\mathcal{D}'$ for training and testing (Line 5); *(iii)* trains a decision tree student $\hat{\pi}_i$ on $\mathcal{D}'_{train}$ (Line 6) using the well-known Classification and Regression Tree (CART) (BREIMAN *et al.*, 1984) method; *(iv)* tests the decision tree explanation using $\mathcal{D}'_{test}$, selecting the samples that the decision tree classifier wrongfully classified as $\mathcal{D}'_e$ (Line 6), and queries the black-box model for the expected results for $\mathcal{D}'_e$ (Line 7), producing a correction dataset $\mathcal{D}_i$ (Line 8); and *(v)* finally augments the optimal dataset $\mathcal{D}'$ with this correction dataset (Line 9) to reinforce the correct decisions during the subsequent iterations. The algorithm selects the student model with the highest fidelity after N iterations (Line 11) and applies a post-processing step (described in Section 5.2) before returning the final explanation (Line 12). In the following, we give a more detailed description of the main design choices we made for TRUSTEE.

Algorithm 1 Model agnostic decision tree explanation extraction.

1: **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 the algorithm,
   k: Number of branches to select from),
2: Initialize dataset using black-box $\mathcal{D} \leftarrow \pi^*(\forall x \in \mathcal{D}_0)$
3: **for** $i \leftarrow 1 \dots N$ **do**
4: Sample M training instances uniformly $\mathcal{D}' \leftarrow \{(x, y) \overset{\text{i.i.d.}}{\sim} U(\mathcal{D})\}$
5: Split subsampled dataset for training and testing
   $\mathcal{D}'_{train}, \mathcal{D}'_{test} \leftarrow$ TRAINTESTSPLIT($\mathcal{D}'$)
6: Train DT $\hat{\pi}_i \leftarrow$ TRAINDECISIONTREE($\mathcal{D}'_{train}$)
7: Test and get samples DT misclassifies
   $\mathcal{D}'_e \leftarrow \{\forall (x, y) \in \mathcal{D}'_{test} \mid \hat{\pi}_i(x) \neq \pi^*(x)\}$
8: Get correct outcome from black-box $\mathcal{D}_i \leftarrow \pi^*(\forall x \in \mathcal{D}'_e)$
9: Augment dataset $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_i$
10: **end for**
11: Select tree with highest fidelity $\hat{\pi}_{max} \leftarrow \hat{\pi} \in \{\hat{\pi}_1, ..., \hat{\pi}_N\}$
12: **return** TOPKPRUNE($\hat{\pi}_{max}$, k)
13: **end procedure**

Multiple iterations and uniform sampling. The CART algorithm traditionally used to train decision tree models relies on a greedy approach to find the best splits in the training dataset. This search is performed in three stages: *(i)* for each feature, the best split is determined by minimizing a given loss function; *(ii)* among all these best feature splits, the best split is selected, and *(iii)* repeat until either all input samples have been classified or a stopping criterion has been reached. This greedy approach ensures that the resulting decision tree will be largely insensitive to the order in which the input samples are processed for a given training dataset. At the same time, in the absence of any explicit means to prevent overfitting, this greedy approach is prone to result in an overfitted decision tree (BRAMER, 2007). While using this approach without further constraints (*e.g.,* stopping criterion) to train a decision tree results in perfect fidelity, being overfitted makes the resulting decision tree ill-suited for providing an intelligible explanation for how the given black-box model makes its decisions. Instead, the resulting explanation is largely an artifact of the method used to generate the explanation.

To overcome this problem, we first note that it is unreasonable to expect that a single decision tree is sufficient to capture the complexity of a black-box model. Indeed, ignoring this issue may result in a small set of features producing misleading global explanations in some cases (LAKKARAJU; BASTANI, 2020). TRUSTEE tackles this problem by employing an iterative approach to train multiple student models on the expert model predictions. To this end, we select a subset of the input samples of size M at each iteration by sampling uniformly from the original training dataset (Lines 3-9). While some existing efforts (BASTANI; PU; SOLAR-LEZAMA, 2018) also use an iterative approach to producing different student models, our approach differs in what training dataset the individual student models have at their disposal. In particular, by requiring the uniform sub-sampling step at each iteration, we ensure that each decision tree explanation will have a limited view of the entire data, akin to a k-fold cross validation (RUSSELL; NORVIG, 2020). 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, sampling uniformly at random from the original training dataset assumes each sample has the same probability of being selected. For imbalanced training datasets, uniform sampling means that classes with fewer samples will appear less frequently in the final decision tree explanation, thus negatively impacting TRUSTEE's ability to explain the black-box model's decision-making process for those "underrepre-

sented" classes. However, it is well known that using imbalanced datasets to train ML models leads to biases towards the majority classes, and the existing ML literature provides several approaches that resolve this problem through proper pre-processing of the original training data (KAUR; PANNU; MALHI, 2019).

Dataset augmentation. An important design choice for TRUSTEE involves a dataset augmentation step (Line 9), where in each iteration, 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 decision tree model makes wrong decisions. Leveraging results from the existing literature on imitation learning (ROSS; GORDON; BAGNELL, 2011; BASTANI; PU; SOLAR-LEZAMA, 2018), performing this step can increase the trained student's overall accuracy and reduce the overall number of leaf nodes in the resulting tree. For example, in the case of RL models, this dataset augmentation step helps to course-correct the wrong trajectories of student models in case they face an unknown (*e.g.,* o.o.d.) state. In practice, we observe that for complex use cases where running our algorithm without this dataset augmentation step results in large decision tree explanations, incorporating this step into our algorithm not only improves the fidelity of an otherwise already high-fidelity explanation but also, and more importantly, significantly (some 20% on average) reduces the explanation's size as measured by the number of nodes of the tree.

Fidelity as objective function. When selecting from among the different student models (one per iteration) that TRUSTEE extracts from a given black-box model, it uses model fidelity as an objective function and picks the student model with the highest fidelity (Line 11). This design choice implies that while the final decision tree explanation produced by TRUSTEE is not necessarily the most *accurate* decision tree for the given black-box model, it is the decision tree that is most *faithful* with respect to explaining how the black-box model makes its decisions. Our working hypothesis is that only by insisting on this high-fidelity aspect of TRUSTEE's output we are able to post-process the resulting decision tree explanation in ways that will help security experts with varying degrees of domain knowledge to gauge their trust in the given black-box model. Below, we provide evidence in support of this working hypothesis and describe the type of post-processing that we perform on the output from TRUSTEE so it can serve as an inherently practical means for faithfully explaining most of the given black-box model's decisions.

5.3 Processing Decision Trees

When synthesizing a high-fidelity decision tree explanation for a given black-box model, realizing the agenda outlined in Section 3 requires including an additional step in Algorithm 1 (Line 12). This step consists of purposefully post-processing the generated decision tree that the algorithm has selected at the end of its iterative process (Line 11) and requires supplying Algorithm 1 with an additional input parameter k (we discuss k in more detail below). In particular, the purpose of this post-processing step is to obtain a final decision tree explanation for the given black-box model that is inherently practical in the sense of having small complexity and, at the same time, having sufficiently high fidelity. Here, when referring to the complexity of a decision tree explanation, we mean small trees but also, and more importantly, trees whose top branches (*i.e.,* decision rules) explicitly, intelligently, and accurately describe how the black-box model makes most of its decision.

Effectively, when producing the final decision tree explanation as the output of this post-processing step, we tolerate some loss of fidelity in return for improved complexity. In the following, we examine different aspects of this fidelity-complexity tradeoff and introduce a novel tree pruning method that we call Top-k Pruning. We design this method for the explicit purpose of ensuring that the final decision tree explanation (*i.e.,* the output that TRUSTEE produces after applying the post-processing step) can be readily processed and understood by human domain experts.

5.3.1 Decision Tree Pruning: Tradeoffs

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 decision trees that can have thousands of nodes (ESPOSITO *et al.*, 1997). Clearly, such large trees are detrimental to our ultimate goal; that is, presenting human domain experts with inherently practical explanations that they can readily inspect and understand with their available domain knowledge. In designing TRUSTEE, we similarly focused first on obtaining a largely unconstrained decision tree explanation with the best possible fidelity (Lines 1-11). However, our reasoning for doing so is that we explicitly require that the explanation that TRUSTEE outputs undergoes a post-processing phase for the purpose of assessing and improving the explanation's complexity. Moreover, our intuition behind obtaining a

high-fidelity decision tree explanation first is that manipulating a high-fidelity explanation with an eye toward 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 decision trees into trees of smaller sizes is pruning, and the existing literature describes several pruning methods for decision trees (ESPOSITO *et al.*, 1997; FRIEDMAN, 2001), many of which are highly effective in obtaining decision trees that have small complexity, at least as far as the size of the trees is 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).

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 decision tree. 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 tradeoff between fidelity and size. However, the better tradeoff comes at the cost of reduced interpretability, mainly because this pruning approach relies on a parameter that is indirectly responsible for how many nodes of a tree get pruned. The lack of a more direct control makes it difficult to decide which decision branches to include in the pruned tree and which to exclude.

5.3.2 Top-k Pruning Method

We present in the following our Top-k pruning method, which requires a parameter k that specifies the number of Top-k branches to keep when pruning a given decision tree. In designing the Top-k pruning method, we note that each branch in the decision tree that TRUSTEE synthesizes by means of its iterative process (Lines 1-11) maps to a classification "rule"; that is, a combination of individual decisions on features that result in labeling the input data as part of a specific class (*e.g.,* malware vs benign). Moreover, each of these "rules" accounts for a certain percentage of all samples in the input dataset, contributing more or less to the overall model fidelity. Also, as the complexity of the decision tree grows (*i.e.,* number of branches), so does the fidelity. Based on this empirical

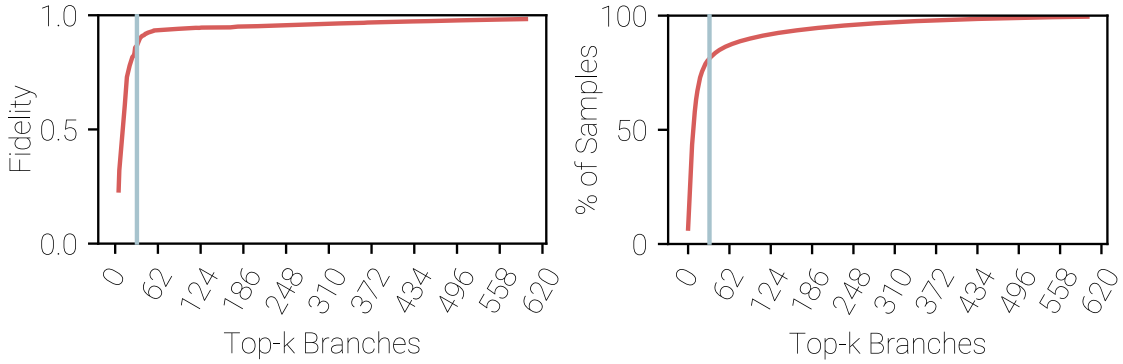


Figure 5.2 – Fidelity and fraction of samples classified by Top-k branches.

observation that we illustrate for an actual use case in Figure 5.2, the idea behind our Top-k Pruning method is simple: 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 decision tree, especially in cases where they 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 decision tree (*i.e.,* branches that are not in the Top-k for large values of k) reflect 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.

Since TRUSTEE proceeds by first producing a high-fidelity decision tree in an unconstrained manner, we have a guarantee that the individual branches of this synthesized tree accurately reflect decisions made by the black-box. Hence, at this point, we are able to relax our requirements on the overall fidelity of the decision tree explanation so as to obtain trees that have lower complexity (*i.e.,* are more manageable). At the same time, we also want end users to be able to carefully control and fine-tune how many of the most important branches they desire to interpret at a time. Our Top-k pruning method achieves this goal via the user-specified parameter k that determines the complexity of the final decision tree that TRUSTEE presents to the end user. If at any point a user wants to inspect more branches of the tree, they can choose a larger k and re-run our algorithm. Note, however, that due to its probabilistic nature, re-running our algorithm may result in applying the Top-k pruning method to an extracted decision tree explanation that is different 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.

5.3.3 Generating Trust Reports

We use the pruned decision tree that forms the output of TRUSTEE as the 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 particular, we intend this trust report to help end users readily spot instances of inductive biases in the given black-box model, which in turn would be proof for the end users that they cannot trust the given model. To this end, we leverage the fact that by virtue of applying our Top-k pruning technique before outputting its solution, the final decision tree explanation synthesized by TRUSTEE is a small tree comprised of k branches. We present the details of this small decision tree to the end users so they can examine it with an eye towards three common ways the underspecification problem can manifest in a given black-box model. More precisely, we are interested in helping end users quickly spot inductive biases that reveal 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. In the following, we briefly describe how the generated trust report helps with detecting each of these three inductive biases.

Shortcut learning. Presenting the small decision tree generated by TRUSTEE and annotating them with pertinent information (*e.g.,* features used, splitting conditions or clauses present at the different nodes, 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 the input data (or a specific class of input samples) is a strong indication of shortcut learning that can, in general, quickly be confirmed with a minimal amount of domain knowledge. Note, however, that a small number of features in the output explanation may also indicate that the classification 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 somewhat more involved investigation of the annotated small decision tree shown as part of the trust report concerns studying the impact of removing the identified most important feature from the provided dataset. We can then retrain the black-box model using this altered dataset, proceed to use TRUSTEE to extract a new decision tree 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 classification accuracy of the black-box model decreases. However, it is often the case,

especially when the data include a very large number of features, that the back-box model is able to find alternative features so that removing important features leaves the overall accuracy essentially unchanged. We take this as a strong indication of spurious correlations in the data that can subsequently be easily confirmed via a simple analysis of the original feature set.

Out-of-distribution samples. The annotated version of the decision tree that is the output of TRUSTEE and is shown as part of the trust report can also be utilized to consider the individual features in each tree branch and plot the distribution of the values that each feature can take in the provided dataset (and include the different distribution plots in the trust report). 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, 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.

5.4 TRUSTEE Evaluation

This section presents a qualitative and quantitative evaluation of TRUSTEE, comparing existing approaches and, arguably, the most popular local explanation method, highlighting the benefits of global explanations produced by our algorithm.

5.4.1 Stability

Given that TRUSTEE only analyzes a subset of the input data to generate an explanation decision tree, it is expected that the resulting explanations will diverge from iteration to iteration. However, if a domain expert is to trust the outcomes of TRUSTEE, it is expected that the same underlying decisions appear in the explanation and that they are not simply artifacts of the data subset. To ensure that, in this section, we evaluate the stability of the decision trees outputted by TRUSTEE, using the same Random Forest Classifier trained on the CIC-IDS-2017 dataset from the Heartbleed use case (Section 4.3.3).

We also conducted this experiment for other use cases and achieved similar results.

Through manual inspection of TRUSTEE generated decision trees, we noticed that, while the underlying decisions of the generated decision trees were stable, the actual rules (from root to leaf) of the tree were not, due to small fluctuations in the tree structure, use of correlated features for similar decisions and changes in the root node. We tried different approaches to quantify this perceived stability and found out that traditional tree stability metrics (such as tree edit distance (ZHANG; SHASHA, 1989) and tree isomorphism (JOVANOVIĆ; DANILOVIĆ, 1984)) were not a good fit since they failed to capture the semantic meaning of decision trees. Hence, we relied on the notion of agreement between decision trees, *i.e.,* comparing if the outcome decision is the same for the same data across multiple iterations of TRUSTEE. Using a 70%-30% train-test split, we ran TRUSTEE using the same 70% training data, from which TRUSTEE sampled a different 30% split of data at every iteration, to generate 50 different Top-10 pruned decision tree explanations from the Random Forest Classifier. We then tested the pair-wise agreement of those 50 decision trees by measuring the F1-score between their classifications on the remaining 30% of the data.

Figure 5.3 presents a heat-map with the pair-wise agreement of each iteration of TRUSTEE with the others. Due to space constraints, we could display 30 out of the 50 different iterations. In addition, Figure 5.4 presents the mean agreement of each of the 50 Top-10 pruned decision trees generated by TRUSTEE on each iteration, *i.e.,* the mean between the F-1 scores in each line of Figure 5.3. Our results showed high stability of decisions made by explanations produced by TRUSTEE, where Top-k pruned trees produced the same outcome for the same samples over 90% of the time, on average. However, on a few iterations (3-4%), the mean agreement of some trees would go as low as 81%. To avoid producing misleading explanations based on those lower-agreement decision trees, one could add an outer loop to TRUSTEE 's algorithm that allows us to produce multiple explanations and filter out the few with a lower mean agreement than the rest. That way, the domain expert would be presented with the decision tree explanation with the highest mean agreement. While this refinement step does not completely prevent TRUSTEE from generating misleading explanations, which remains an open problem, it gives domain experts more confidence in the decision presented in the resulting explanation.

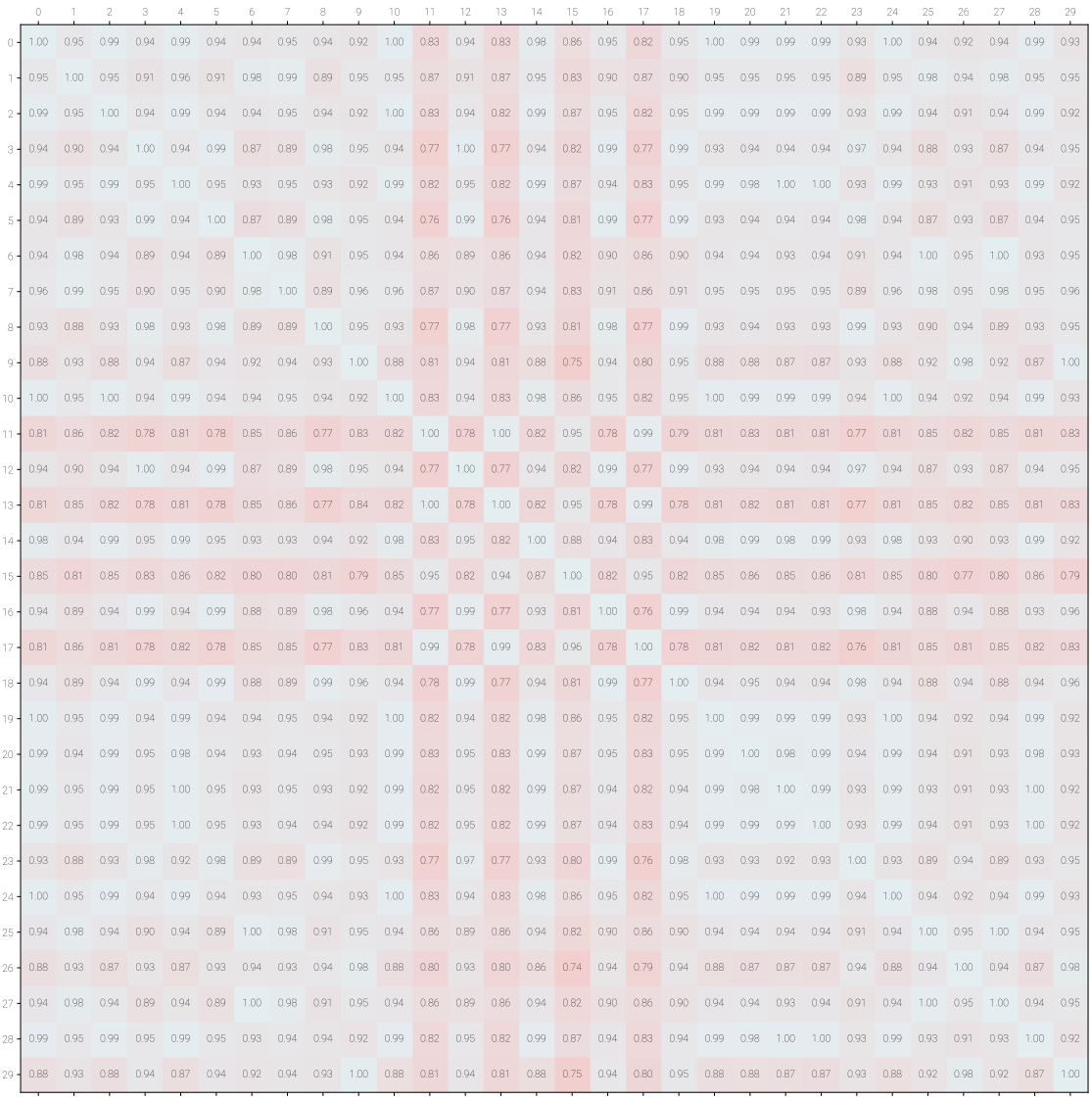


Figure 5.3 – Pair-wise agreement of decision trees generated by 30 out of 50 iteration of TRUSTEE.

5.4.2 TRUSTEE vs. SHAP

To motivate the need for a new global explanation method, we examine an example of shortcut learning presented in (GEIRHOS *et al.*, 2020). In this toy example (Figure 5.5), a deep neural network with three fully connected layers classifies images as either moons or stars. The neural network is trained with a biased dataset of images (in yellow in Figure 5.5) in which the stars are always in the upper-right or lower-left quadrants, while the moons are always in the upper-left or lower-right quadrants. When evaluating the classifier with i.i.d. test samples (in blue in Figure 5.5), it achieves a perfect F1 score, which suggests the neural network learns to distinguish between the shapes of moons and stars.

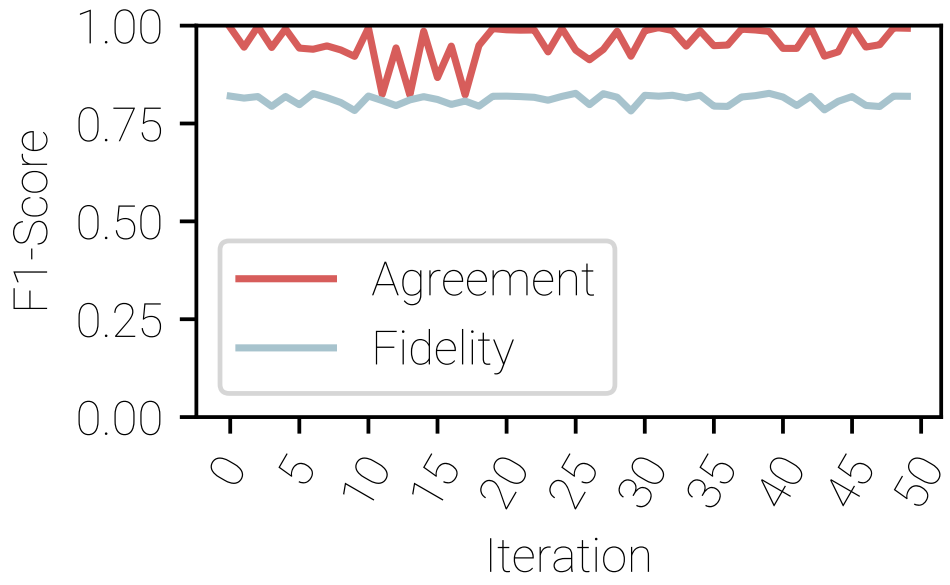


Figure 5.4 – Mean agreement of decision trees generated by 50 iteration of TRUSTEE.

5.4.2.1 SHAP for Moons and Stars Use Case

To verify the credibility of the classifier, we first use SHAP (LUNDBERG; LEE, 2017) to explain five random i.i.d. test samples. The left plot of Figure 5.6 shows the SHAP results for each sample: on the left, the original test sample; in the middle, the pixels contributing for or against a 'star' classification; and on the right, the pixels contributing for or against a 'moon' classification. Blue pixels represent pixels contributing to classifying a class, while red pixels represent pixels contributing against that classification. With this explanation, a domain expert would believe the neural network learned each object's correct shape, as all the 'right' pixels of a moon or a star are in blue, and the 'wrong' pixels are in red.

When we test the classifier with o.o.d. samples (in red in Figure 5.5), generated with stars and moons in random positions, the classifier's accuracy drops to almost the probability of a random coin toss (*i.e.,* F1 score close to 0.5). From the o.o.d. evaluation results, it becomes clear that the classifier did not learn the shape of the objects but rather their positions. By running SHAP on o.o.d. test samples (right plot of Figure 5.6), a domain expert could detect that the model suffers from underspecification, as only the pixels in the 'correct' positions for star and moon are in blue, while stars in the wrong position get highlighted in red. However, o.o.d. are notoriously hard to come by, and relying on such cases to identify underspecification issues is detrimental to the purpose of

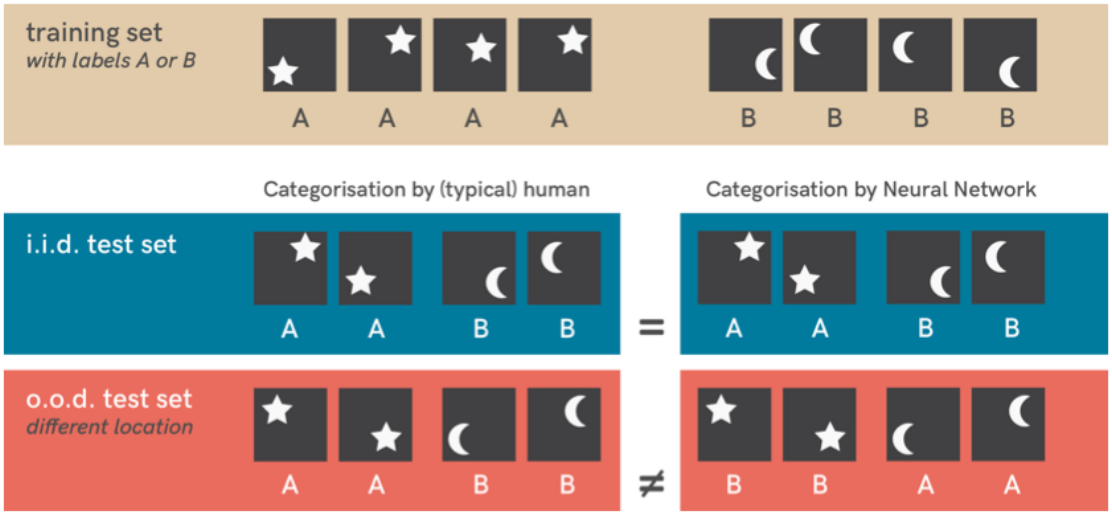


Figure 5.5 – Shortcut learning toy example, from (GEIRHOS *et al.*, 2020), used with permission.

model explainability.

This simple example illustrates the limitations of local explainability tools like SHAP. When presented with i.i.d. test samples, which is usually the case when testing a classifier, local explanations can easily misguide model developers into believing a model is trustworthy and ready for production. In practice, local explainability tools work better for post-hoc analysis, while a global explanation can give a more 'complete' picture of how the model behaves, allowing a developer to detect such issues before deploying the model.

5.4.2.2 TRUSTEE *for Moons and Stars Use Case*

We start by applying TRUSTEE to extract a decision tree explanation from the trained neural network, shown in Figure 5.7. Immediately by looking at the decision tree, it is clear that there is some underspecification problem inflicting the trained black-box. The generate decision tree explanation only has five nodes (including leaves) and three branches. With only two decision splits, we can achieve a perfect fidelity to the black-box decisions (when testing with i.i.d. samples).

By analyzing the decisions made by the generated explanation, we notice that the pixel numbers indicated in the top nodes correspond to pixels in the upper-left quadrant (X_{9233}) and the lower-left quadrant (X_{30129}). So, this explanation tells us that only by looking at two single pixels, which obviously correspond to their positions in the images, the black-box is capable of perfectly distinguishing between moon and star shapes. This toy example perfectly synthesizes how powerful a global explanation tool can be in identifying underspecification problems in black-box models. However, as discussed in Section

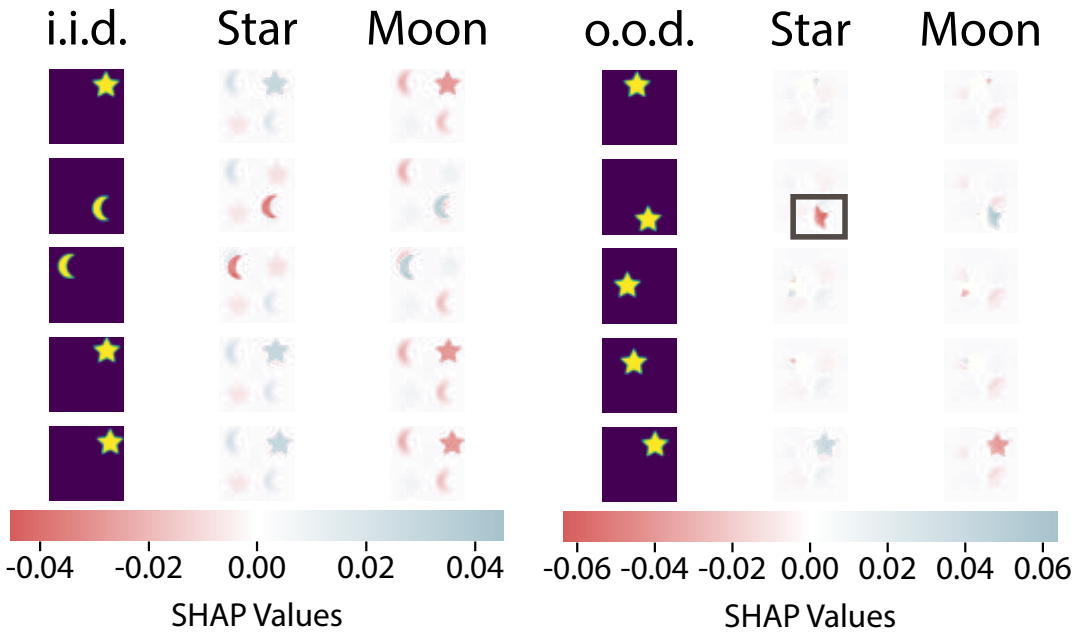


Figure 5.6 – Results from running SHAP on i.i.d. (left) and o.o.d. (right) test samples from shortcut learning toy example. Square on the right mark the cases in which SHAP values show that only position of elements matter, rather than shape.

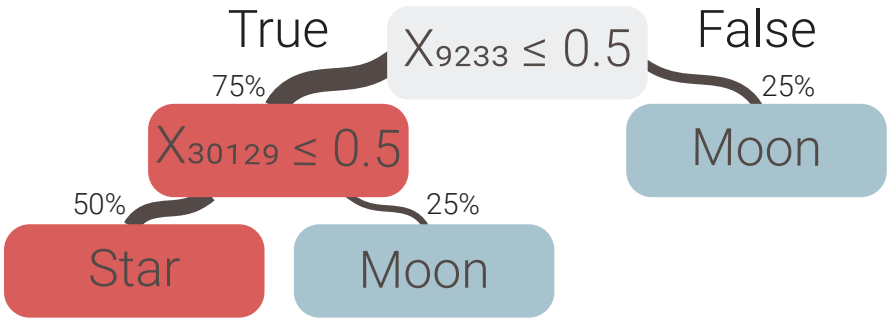


Figure 5.7 – Decision tree explanation extracted from the blackbox neural network model, with fidelity (*i.e.,* F-1 score) = 1. Each node depicts a decision based on the pixel of the images used as input for the black-box model.

5.2, our explanation method leverages random sampling to drive explanations to a higher fidelity. This means that, by repeatedly executing TRUSTEE on this black-box model, one would get hundreds of different decision tree explanations with perfect fidelity, as this toy example has numerous intrinsic shortcuts. That is, if we removed the given pixels X_{9233} and X_{30129} from the data samples, the black-box would still find other pixels to perfectly match the desired position of moons and stars, and that would be reflected in the generated decision tree explanations.

5.4.3 TRUSTEE vs. Related Work

To complement the qualitative comparison we provided in Section 5.2, we present below a quantitative comparison of fidelity results from existing approaches in the literature to extract decision tree explanation from black-box models. We evaluate all competing approaches using two of the use cases analyzed in Section 4.3: the VPN-vs-NonVPN Use Case (WANG *et al.*, 2017), where the black-box model is a 1D-CNN, and the Random Forest Classifier from the CIC-IDS-2017 Use Case (SHARAFALDIN.; H. Lashkari.; GHORBANI., 2018). All explanation methods were given the same 200 node limit. Table 5.2 presents the fidelity and size of the resulting explanation from each method.

Since the two analyzed use cases do not rely on RL models, neither VIPER (BASTANI; PU; SOLAR-LEZAMA, 2018) nor Metis (MENG *et al.*, 2020) could be applied to extract an explanation. In both use cases, TRUSTEE was able to achieve a near-perfect fidelity even with the given 200 maximum nodes constraint. While Trepan performed well for the simpler VPN use case, it produces subpar results for the more complex CIC-IDS-2017 trained model. Lastly, *dtextract* had the worst resulting explanation of all methods, producing very low fidelity decision trees.

Note that *dtextract*'s only stopping criterion is the number of nodes. Hence, until the given number of nodes is reached, it will continue adding new nodes to the decision tree, even if unnecessary, as illustrated by the VPN use case. This behavior also makes the resulting explanation a bad vehicle for identifying the clear shortcut we describe in Section 4.3 in that specific use case, which Trepan successfully picked up on. In addition, neither Trepan nor *dtextract* could uncover the Heartbleed o.o.d. underspecification problem from the CIC-IDS-2017 use case presented in the previous chapter.

Table 5.2 – Size and fidelity of decision tree explanations produce by existing methods to interpret black-box models for different use cases. Size is measured as the number of nodes in the resulting decision tree. Fidelity is measured as the F1-score between black-box predictions and decision tree predictions.

	VPN vs NonVPN			CIC-IDS-2017		
Method	Max Nodes	Size	Fidelity	Max Nodes	Size	Fidelity
Trepan (CRAVEN; SHAVLIK, 1995)	200	9	0.99	200	200	0.75
dtextract (BASTANI; KIM; BASTANI, 2017)	200	200	0.55	200	200	0.24
VIPER (BASTANI; PU; SOLAR-LEZAMA, 2018)	-	-	-	-	-	-
Metis (MENG *et al.*, 2020)	-	-	-	-	-	-
TRUSTEE	200	7	**1**	200	200	**0.99**

5.5 Discussion

While using TRUSTEE to scrutinize several use cases of published ML models for network security problems and uncover a number of common pitfalls in those models that make them ill-suited for deployment in production networks. For one, model developers need to be aware that using features in the form of bit- or byte-level representations of semantically rich raw networking data in the hope that a model will capture meaningful patterns in the underlying data is problematic. While this practice is standard for ML-based image and text processing and classification, the distributions of the feature values (*i.e.,* pixels or words) for these applications are generally available via community crowd-sourcing datasets. We argue that ML-based network security applications are better off relying on carefully engineered and semantically meaningful features that are less prone to measurement artifacts that are commonly present in synthetic datasets such as the ones we described in Section 4.3 and result in trained models that cannot be trusted.

Second, we emphasize that the analyses in this and previous chapters were made possible because a few other researchers made their artifacts publicly available. In recent years, the network and security communities have pushed for more reproducibility, and we second this effort. However, network security researchers interested in using AI/ML should accept the lack of open-source datasets and a general reluctance for widespread data sharing as *faits accomplis*. Instead, they should embrace sharing AI/ML research artifacts such as learning algorithms or trained learning models and take charge of the data problem by collecting the necessary data themselves, in close collaboration with their universities' IT organizations.

Last, because of a lack of open-source artifacts, we could not find any networking-related ML model that we deemed trustworthy when analyzing decisions uncovered by TRUSTEE. Our intuition is that an explanation for a trustworthy model would consist of decisions a domain expert would agree on and would show strong robustness or stability properties with respect to the Top-k (for a range of k-values) of the decision trees that result from multiple executions of TRUSTEE. Yet, determining whether a model can be trusted ultimately depends on the help of human experts who can assert if the model makes decisions in accordance with existing domain knowledge or is even capable of teaching the domain experts new patterns. To realize this goal, we need to overcome the challenge of involving networking and network security experts in carefully designed user studies for quantitatively assessing their level of trust in a given black-box ML model. Otherwise,

we as a community will have to settle for ML solutions that provide recommendations only, rather than automatic classification, which would require further human interactions. While using ML for recommendations can help guide human interactions, it would still prevent us from reaching the vision of a genuinely self-driving network.

5.6 Chapter Summary and Remarks

In this chapter, we address this thesis's third and last research question, which drove us to find a practical way to increase the trust of operators in the ML models used in self-driving networks. In particular, we equate *"an end user having trust in an AI/ML model"* with *"an end user being comfortable with relinquishing control to the model"* and propose a novel framework called TRUSTEE that allowed us to put in practice the new AI/ML pipeline to scrutinize decision made by black-box ML models in networking problems. Our focus in designing TRUSTEE was on determining whether or not a given AI/ML model suffers from any underspecification, such as shortcut learning or spurious correlations. As the use cases presented in Chapter 4 already illustrated how operators could use TRUSTEE's decision trees and associated trust reports to detect the presence of underspecification issues, we focused in this chapter on showing how the explanations produced by TRUSTEE fare compared to existing global and local explanation methods, in terms of fidelity and complexity. In practical terms, we showed that TRUSTEE, the Top-k pruning and the proposed *trust reports* enables operators to carefully analyze and determine whether ML models can be trusted to relinquish control.

6 CONCLUSIONS

This thesis has investigated the use of ML techniques in the general design of a self-driving network. Despite not having a consensus among researchers, all designs of a self-driving network found in the literature rely heavily on ML classifiers. In particular, the IBN efforts embodied by self-driving networks require extensive use of ML to extract information from natural language intents that dictate the network behavior. In addition, the grand vision of a self-driving network hinges on using ML models to learn and self-configure network devices according to traffic and user behaviors. However, the indiscriminate use of ML to solve network management tasks raises an inherent lack of trust in the black-box classifiers among network practitioners. This lack of trust drove us to establish the following problem statement for developing this thesis.

Problem Statement: *The present thesis aims to enable self-driving networks with ML, tackling the problem of the inherent lack of trust in ML models that empower it by assessing their decision-making process.*

To tackle this problem, this thesis analyzed and evaluated the decision-making process of ML-based classifiers that compose a self-driving network. We also propose a novel method to uncover the "inner-workings" of any given black-box ML model, which operators and domain experts can leverage to gain trust in decisions made by such ML models. Our results indicate that ML models employed to extract information from natural language intents can be trusted, with minimal safeguarding and human-in-the-loop verification. However, ML models that automatically solve network security and performance problems have not been put under proper scrutiny and can easily break when put under stress, failing to fulfill their given tasks properly.

6.1 Research Questions

The results presented in this thesis allow us to answer the three key research questions that guided our efforts, as detailed below.

RQ-1: *In a self-driving network, can operators trust decisions and classifications made by current Machine Learning models used to configure the network based on high-level intents?*

Answer: A self-driving network design requires minimal human interaction from operators when expressing network intents. This makes it possible to rely on humans to verify classifications from ML models applied to extracting relevant information from natural language intents and correct them when necessary by providing feedback. This feedback loop enables operators to verify and, therefore, trust decisions and classifications made by such models.

In this thesis, we introduced LUMI, a conversational assistant (*i.e.,* chatbot) that allows operators to use natural language to describe high-level network intents, filling the existing gap in IBN literature. LUMI relies on highly accurate ML techniques from NLP to extract information from input intents and uses its conversational interface to confirm with operators if the parsed information is correct. More importantly, LUMI can collect feedback from operators on its classifications and uses that feedback to retrain and improve the underlying ML model accuracy over time. We relied on LUMI to answer the first research question, and our results showed that LUMI is capable of accurately extracting information from both synthetic and real-world input intents. We also conducted a small-scale user study with network practitioners from different backgrounds and expertise to assess LUMI's usability, where 88.5% of participants stated they would instead use LUMI exclusively or in conjunction with configuration commands.

RQ-2: *In a self-driving network, can operators trust decisions and classifications made by current Machine Learning models used to self-configure the network based on monitored data analysis?*

Answer: ML models widely applied to solve networking problems fail to fulfill their tasks when put under minimal stress. Since it is unfeasible for an operator to verify every decision in a zero-touch management loop manually, our results indicate that operators are, in fact, correct not to trust ML models to configure the network based solely on monitored data automatically.

In this thesis, we focused on analyzing the application of ML techniques in the network security problem space as a use case for the second management loop in a self-driving network. Based on a motivating use case from the network performance domain, we proposed a revision to the traditional AI/ML pipeline. Our new AI/ML pipeline relies on XAI techniques to extract decision-tree explanations from black-box ML models, elevating them to the role of first-class citizens, becoming the primary vehicle for scrutinizing decisions from ML models. We selected a few key works from the existing literature to put under scrutiny and reproduce their results. Our analysis showed that while

most works in the literature report outstanding classification accuracy in their evaluation scenarios, the classifications made are not credible enough to be put under unforeseen real-world traffic and could lead to system-breaking configurations to be applied.

RQ-3: *Is there a feasible way to increase operator trust in the decision-making process of Machine Learning models that configure a self-driving network?*

Answer: By employing a new research pipeline for AI/ML in networking and network security, one can expose decisions made by black-box ML classifiers and therefore increase trust in those decisions. Given this knowledge of the "inner-workings" of ML models, operators can choose to relinquish control to the ML model if they agree with the decision made by the model.

In this thesis, we introduce TRUSTEE, a novel model-agnostic method for extracting global explanations from black-box ML models in the form of decision trees. To ensure the extracted explanations are manageable in size while maintaining high fidelity to the black-box model's decisions, we introduce the Top-k pruning method. Finally, to help operators spot the presence of inductive biases in the learning strategies employed by the ML models, we summarize and condense a series of analyses based on the extracted explanation and underlying data into a *trust report*. This set of novel models allowed us to implement our proposed AI/ML pipeline in practice and improve trust in ML-based methods of self-driving networks. In addition, our results and experiments show that TRUSTEE is capable of producing stable global explanations, with a better fidelity-complexity trade-off than existing methods.

6.2 Future Work and Open Problems

Finally, while the work presented in this thesis advances the state-of-the-art on multiple fronts, from IBN to XAI, with promising and necessary steps towards fully realizing self-driving networks, several challenges and open problems remain to be tackled. We list some of these research opportunities for future work in this section.

6.2.1 Ambiguities in natural language policies

NLP policies have the potential for ambiguities. In exploring this issue further, we extracted pairs of intents from our *campi* dataset and generated 213 pairs in all.

Furthermore, we adapted existing NLP efforts on contradiction detection in general text (MACCARTNEY *et al.*, 2006; MARNEFFE; RAFFERTY; MANNING, 2008; TAWFIK; SPRUIT, 2018; LINGAM *et al.*, 2018; RITTER *et al.*, 2008; HARABAGIU; HICKL; LACATUSU, 2006; SARAFRAZ, 2012) by developing a Random Forest Classifier trained to classify pairs of network intents based on contradiction indicators (features) found between the two input intents. The classifier flagged 9 cases with potential contradictions.

Manual inspection of all 213 intent pairs indicated that most of these cases were benign. They typically corresponded to cases where universities expressed policies that depended on a user's total traffic usage over different periods (*e.g.,* a 10 GB weekly limit vs. a 5 GB daily limit). The two policies could be applied in any order with no negative consequences in these cases. One interesting case, however, was a campus that expressed two different policies at different locations on their website. The first policy indicated H323 video conferencing was allowed by the University firewall, while the second indicated MSN audio and video communications were not allowed. This is a case where the relative precedence between the two policies impacts their combined effect.

A promising research opportunity would be to combine LUMI with formal methods to help detect ambiguities that are potentially of concern. Specifically, translating NLP policies into LUMI intents enables the use of automated methods that can check whether the impact of applying two policies is sensitive to their relative ordering but also provides opportunities to use methods for detecting policy conflicts (PRAKASH *et al.*, 2015; ABHASHKUMAR *et al.*, 2017).

6.2.2 Deploying Lumi in a production network

The initial design of LUMI was aimed at solving management problems that arise in a Campus network environment. We have engaged in discussions with operators of our campus network regarding validating LUMI in production. Below, we discuss some of the issues raised by the operators and outline challenges and potential solutions.

Co-existing with current technologies. Most campus networks consist of legacy equipment from multiple vendors with vendor-specific configuration interfaces. Getting LUMI "production-ready" requires developing a *Nile* compiler that can accommodate this diversity in legacy devices. Since the *Nile* abstraction offers isolation from the system's interface and minimizes the need for changes in the early stages of the LUMI pipeline, it is well-suited for OpenConfig (OPENCONFIG, 2016), a vendor-neutral model for network management that is supported by an increasing number of devices.

Extending LUMI **for other use-cases**. While we have focused on Campus networks, deploying LUMI in other environments may require extending its feature set. For example, unlike the Campus networks we have access to, multi-tenant data-center networks may use VXLANs or NVGRE as solutions to scalably share network infrastructure between tenants. However, LUMI is easily extended to support such features, and we illustrate the four generic steps required for such extension with the VXLAN example. In a first step, one must decide the abstraction level in which LUMI should handle natural language text. Consistent with LUMI's design philosophy, for VXLAN support, a high-level intent could be *"Block incoming traffic for tenant A"*, where *"tenant A"* refers to a specific VXLAN Network Identifier (VNI). Next, LUMI's training dataset has to be augmented appropriately, the *Nile* language must be extended with new keywords (*e.g.,* a `tenant('A')` operator for the VXLAN example), and the *Nile* compiler has to be instrumented to handle the new set of configurations. In the case of VXLAN, similarly to VLANs, tenants' names must be mapped to VNIs. Lastly, since OpenFlow switches support VXLANs, all that is needed is to extend *Merlin* to allow matching traffic based on VNI for each tenant.

6.2.3 How to verify if Lumi is correct?

We discuss potential sources of errors in each stage of the LUMI pipeline and possible solution approaches.

Translating human language to *Nile* intents. Information extraction from natural language is inherently prone to errors. LUMI alleviates this problem by asking operators for feedback and using their responses to check if the extracted information was correct. Over the longer term, we plan to leverage ongoing ML research efforts that focus on making ML models more robust and secure so they can be deployed in security- and safety-critical settings such as production network environments (HUANG *et al.*, 2017; BASTANI *et al.*, 2016).

Compiling *Nile* intents. Recent works on formal network verification (BECKETT *et al.*, 2017; ABHASHKUMAR; GEMBER-JACOBSON; AKELLA, 2020; PRABHU *et al.*, 2020) provide sub-second verification of waypointing, reachability, and isolation properties and are well suited for verifying configurations generated by LUMI-compiled intents. LUMI supports time-constrained intent deployment and QoS features. Despite recent work on verifying such properties (*e.g.,* (SUNG *et al.*, 2009; KIM *et al.*, 2015)),

more advances may be needed in the area, including the possible adaption of existing verification techniques to a LUMI-specific setting.

Post-deployment behavior monitoring. Ultimately, we envision the LUMI pipeline shown in Figure 3.2 to include a monitoring module for verifying that the deployed configurations respect the intents produced by the refinement process and achieve the objective(s) that the operator expressed (in natural language) in the first place. By monitoring both the traffic and configurations of specific devices affected by a deployed intent, such a module would allow operators to query at any time if the deployed intent produced the desired network behavior (FOSTER *et al.*, 2020), thereby improving the operators' trust in relying on LUMI. However, deciding which traffic, devices, and properties to monitor will require instrumenting networks with an unprecedented level of control that is currently only possible by leveraging the latest programmable data plane technologies (GUPTA *et al.*, 2018; BASAT *et al.*, 2020). At the same time, developing such a module can be viewed as a first step toward realizing the vision of self-driving networks (MOGUL, 2018; FEAMSTER; REXFORD, 2018; BEHRINGER *et al.*, 2015).

6.2.4 Distilling stable explanations from TRUSTEE

We show in Chapter 5 that TRUSTEE is capable of generating multiple different explanations for the same black-box ML model, as it relies on a fraction of the training data to produce decision trees. While our results demonstrate that these explanations are mostly in agreement, we believe that there is an opportunity to reconcile those multiple explanations into a single stable decision tree. One possible alternative to achieve that is to rely on methods for model distillation (HINTON; VINYALS; DEAN, 2015; BUCILUA; CARUANA; NICULESCU-MIZIL, 2006), which aim to encode large and complex models into smaller and simpler light-weight models. As such, one could envision a method that compresses different classification rules from multiple decision trees generated by TRUSTEE into a single decision tree. This would further reduce the possibility of generating misleading explanations from the underlying data.

REFERENCES

ABHASHKUMAR, A.; GEMBER-JACOBSON, A.; AKELLA, A. Tiramisu: Fast and General Network Verification. In: **17th USENIX Symposium on Networked Systems Design and Implementation**. Santa Clara, CA: USENIX Association, 2020. (NSDI '20). Available from Internet: <https://www.usenix.org/conference/nsdi20/presentation/abhashkumar>.

ABHASHKUMAR, A. *et al.* Supporting Diverse Dynamic Intent-based Policies Using Janus. In: **Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies**. New York, NY, USA: ACM, 2017. (CoNEXT '17), p. 296–309. ISBN 978-1-4503-5422-6. Available from Internet: <http://doi.acm.org/10.1145/3143361.3143380>.

ADADI, A.; BERRADA, M. Peeking Inside the Black-Box: A Survey on Explainable Artificial Intelligence (XAI). **IEEE Access**, v. 6, p. 52138–52160, 2018.

AL., A. D. et. 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**. New York, NY, USA: Association for Computing Machinery, 2019. (NetAI'19), p. 29–36. ISBN 9781450368728. Available from Internet: <https://doi.org/10.1145/3341216.3342210>.

ALSUDAIS, A.; KELLER, E. Hey network, can you understand me? In: **2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)**. [S.l.: s.n.], 2017. p. 193–198.

AMOKRANE, A. *et al.* Flow-Based Management For Energy Efficient Campus Networks. **IEEE Transactions on Network and Service Management**, v. 12, n. 4, p. 565–579, dec. 2015. ISSN 1932-4537.

ANDERSON, C. J. *et al.* NetKAT: Semantic Foundations for Networks. **SIGPLAN Not.**, ACM, New York, NY, USA, v. 49, n. 1, p. 113–126, jan. 2014. ISSN 0362-1340. Available from Internet: <http://doi.acm.org/10.1145/2578855.2535862>.

APLEY, D. W.; ZHU, J. Visualizing the effects of predictor variables in black box supervised learning models. **Journal of the Royal Statistical Society: Series B (Statistical Methodology)**, v. 82, n. 4, p. 1059–1086, 2020. Available from Internet: <https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/rssb.12377>.

APOSTOLOPOULOS, J. **Improving Networks with Artificial Intelligence**. Cisco, 2020. <https://blogs.cisco.com/networking/improving-networks-with-ai>. Available from Internet: <https://blogs.cisco.com/networking/improving-networks-with-ai>.

APRUZZESE, G. *et al.* On the effectiveness of machine and deep learning for cyber security. In: **2018 10th International Conference on Cyber Conflict (CyCon)**. [S.l.: s.n.], 2018. p. 371–390.

APRUZZESE, G.; PAJOLA, L.; CONTI, M. The Cross-evaluation of Machine Learning-based Network Intrusion Detection Systems. **IEEE Transactions on Network and Service Management**, p. 1–1, 2022.

ARJOVSKY, M. *et al.* **Invariant Risk Minimization**. 2020.

ARP, D. *et al.* Dos and Dont's of Machine Learning in Computer Security. In: **31st USENIX Security Symposium (USENIX Security 22)**. Boston, MA: USENIX Association, 2022. Available from Internet: <https://www.usenix.org/conference/usenixsecurity22/presentation/arp>.

BAJPAI, V. *et al.* The Dagstuhl Beginners Guide to Reproducibility for Experimental Networking Research. **SIGCOMM Comput. Commun. Rev.**, Association for Computing Machinery, New York, NY, USA, v. 49, n. 1, p. 24–30, feb. 2019. ISSN 0146-4833. Available from Internet: <https://doi.org/10.1145/3314212.3314217>.

BASAT, R. B. *et al.* PINT: Probabilistic In-Band Network Telemetry. In: **Proceedings of the Annual Conference of the ACM SIGCOMM**. New York, NY, USA: ACM, 2020. (SIGCOMM '20), p. 662–680. ISBN 9781450379557. Available from Internet: <https://doi.org/10.1145/3387514.3405894>.

BASTANI, O. *et al.* Measuring Neural Net Robustness with Constraints. In: **Advances in Neural Information Processing Systems**. Curran Associates, Inc., 2016. v. 29. Available from Internet: <https://proceedings.neurips.cc/paper/2016/file/980ecd059122ce2e50136bda65c25e07-Paper.pdf>.

BASTANI, O.; KIM, C.; BASTANI, H. Interpreting Blackbox Models via Model Extraction. **CoRR**, abs/1705.08504, 2017. Available from Internet: <http://arxiv.org/abs/1705.08504>.

BASTANI, O.; PU, Y.; SOLAR-LEZAMA, A. Verifiable Reinforcement Learning via Policy Extraction. In: **Proceedings of the 32nd International Conference on Neural Information Processing Systems**. Red Hook, NY, USA: Curran Associates Inc., 2018. (NIPS'18), p. 2499–2509.

BECKETT, R. *et al.* A General Approach to Network Configuration Verification. In: **Proceedings of the Annual Conference of the ACM SIGCOMM**. New York, NY, USA: ACM, 2017. (SIGCOMM '17), p. 155–168. ISBN 978-1-4503-4653-5. Available from Internet: <http://doi.acm.org/10.1145/3098822.3098834>.

BECKETT, R. *et al.* Don't Mind the Gap: Bridging Network-wide Objectives and Device-level Configurations. In: **Proceedings of the Annual Conference of the ACM SIGCOMM**. New York, NY, USA: ACM, 2016. (SIGCOMM '16), p. 328–341. ISBN 978-1-4503-4193-6. Available from Internet: <http://doi.acm.org/10.1145/2934872.2934909>.

BEHRINGER, M. *et al.* **Autonomic Networking: Definitions and Design Goals**. [S.l.], 2015.

BIRKNER, R. *et al.* Net2Text: Query-Guided Summarization of Network Forwarding Behaviors. In: **15th USENIX Symposium on Networked Systems Design and Implementation**. Renton, WA: USENIX Association, 2018. (NSDI '18), p. 609–623. ISBN 978-1-931971-43-0. Available from Internet: <https://www.usenix.org/conference/nsdi18/presentation/birkner>.

BOSSHART, P. *et al.* P4: Programming Protocol-Independent Packet Processors. **SIGCOMM Comput. Commun. Rev.**, Association for Computing Machinery, New York, NY, USA, v. 44, n. 3, p. 87–95, jul 2014. ISSN 0146-4833. Available from Internet: <https://doi.org/10.1145/2656877.2656890>.

BOUABENE, G. *et al.* The autonomic network architecture (ANA). **IEEE Journal on Selected Areas in Communications**, v. 28, n. 1, p. 4–14, 2010.

BOUTABA, R. *et al.* A comprehensive survey on machine learning for networking: evolution, applications and research opportunities. **Journal of Internet Services and Applications**, v. 9, n. 1, p. 16, 2018. Available from Internet: <https://doi.org/10.1186/s13174-018-0087-2>.

BRAMER, M. Avoiding Overfitting of Decision Trees. In: ____. **Principles of Data Mining**. London: Springer London, 2007. p. 119–134. ISBN 978-1-84628-766-4. Available from Internet: <https://doi.org/10.1007/978-1-84628-766-4_8>.

BREIMAN, L. *et al.* **Classification and Regression Trees**. Monterey, CA: Wadsworth and Brooks, 1984.

BRUNDAGE, M. *et al.* **Toward Trustworthy AI Development: Mechanisms for Supporting Verifiable Claims**. arXiv, 2020. Available from Internet: <https://arxiv.org/abs/2004.07213>.

BUCILUA, C.; CARUANA, R.; NICULESCU-MIZIL, A. Model Compression. In: **Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining**. New York, NY, USA: Association for Computing Machinery, 2006. (KDD '06), p. 535–541. ISBN 1595933395. Available from Internet: <https://doi.org/10.1145/1150402.1150464>.

BUSSE-GRAWITZ, C. *et al.* pForest: In-Network Inference with Random Forests. **CoRR**, abs/1909.05680, 2019. Available from Internet: <http://arxiv.org/abs/1909.05680>.

CAESAR, H. *et al.* nuScenes: A multimodal dataset for autonomous driving. **arXiv preprint arXiv:1903.11027**, 2019. Available at <https://www.nuscenes.org>.

CHANG, M. *et al.* Argoverse: 3D Tracking and Forecasting With Rich Maps. In: **Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)**. [S.l.: s.n.], 2019. Available at <https://www.argoverse.org/>.

CHEMINOD, M. *et al.* A comprehensive approach to the automatic refinement and verification of access control policies. **Computers & Security**, v. 80, p. 186–199, 2019. ISSN 0167-4048. Available from Internet: <http://www.sciencedirect.com/science/article/pii/S0167404818303870>.

CHITICARIU, L. *et al.* SystemT: Declarative Text Understanding for Enterprise. In: **Proceedings of the 2018 Conference of the North American Chapter of the ACL: Human Language Technologies, Volume 3**. ACL, 2018. p. 76–83. Available from Internet: <http://aclweb.org/anthology/N18-3010>.

CHIU, J.; NICHOLS, E. Named Entity Recognition with Bidirectional LSTM-CNNs. **Transactions of the ACL**, v. 4, p. 357–370, 2016. Available from Internet: <http://aclweb.org/anthology/Q16-1026>.

CLEMM, A. *et al.* **Intent-Based Networking - Concepts and Definitions**. [S.l.], 2019. Work in Progress. Available from Internet: <https://datatracker.ietf.org/doc/html/draft-irtf-nmrg-ibn-concepts-definitions-00>.

CRAVEN, M. W.; SHAVLIK, J. W. Extracting Tree-Structured Representations of Trained Networks. In: **Proceedings of the 8th International Conference on Neural Information Processing Systems**. Cambridge, MA, USA: MIT Press, 1995. (NIPS'95), p. 24–30.

D'AMOUR, A. *et al.* **Underspecification Presents Challenges for Credibility in Modern Machine Learning**. 2020.

DARKTRACE. **Darktrace Cyber AI Analyst: Autonomous Investigations**. [S.l.], 2021. Available at <https://www.darktrace.com/en/resources/wp-cyber-ai-analyst.pdf>. Available from Internet: <https://www.darktrace.com/en/resources/wp-cyber-ai-analyst.pdf>.

DENG, J. *et al.* ImageNet: A large-scale hierarchical image database. In: **2009 IEEE Conference on Computer Vision and Pattern Recognition**. [S.l.: s.n.], 2009. p. 248–255.

DORIGUZZI-CORIN, R. *et al.* Lucid: A Practical, Lightweight Deep Learning Solution for DDoS Attack Detection. **IEEE Transactions on Network and Service Management**, v. 17, n. 2, p. 876–889, 2020.

DRAPER-GIL., G. *et al.* Characterization of Encrypted and VPN Traffic using Time-related Features. In: INSTICC. **Proceedings of the 2nd International Conference on Information Systems Security and Privacy - ICISSP,**. [S.l.]: SciTePress, 2016. p. 407–414. ISBN 978-989-758-167-0. ISSN 2184-4356.

DURUMERIC, Z. *et al.* The Matter of Heartbleed. In: **Proceedings of the 2014 Conference on Internet Measurement Conference**. [S.l.]: Association for Computing Machinery, 2014. (IMC '14), p. 475–488.

DWIVEDI, S.; VARDHAN, M.; TRIPATHI, S. An effect of chaos grasshopper optimization algorithm for protection of network infrastructure. **Computer Networks**, v. 176, p. 107251, 2020. ISSN 1389-1286. Available from Internet: <https://www.sciencedirect.com/science/article/pii/S1389128619310175>.

ERICKSON, N. *et al.* **AutoGluon-Tabular: Robust and Accurate AutoML for Structured Data**. arXiv, 2020. Available from Internet: <https://arxiv.org/abs/2003.06505>.

ESPOSITO, F. *et al.* A comparative analysis of methods for pruning decision trees. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 19, n. 5, p. 476–491, 1997.

ETSI. **Autonomous Networks, supporting tomorrow's ICT business**. [S.l.], 2020. Available from Internet: <https://www.etsi.org/images/files/ETSIWhitePapers/etsi-wp-40-Autonomous-networks.pdf>.

FAYYAD, U.; PIATETSKY-SHAPIRO, G.; SMYTH, P. From Data Mining to Knowledge Discovery in Databases. **AI Magazine**, v. 17, n. 3, p. 37, Mar. 1996. Available from Internet: <https://ojs.aaai.org/index.php/aimagazine/article/view/1230>.

FEAMSTER, N.; REXFORD, J. Why (and How) Networks Should Run Themselves. In: **Proceedings of the Applied Networking Research Workshop**. New York, NY, USA: ACM, 2018. (ANRW '18), p. 20. ISBN 9781450355858. Available from Internet: <https://doi.org/10.1145/3232755.3234555>.

FINAMORE, A. *et al.* KISS: Stochastic Packet Inspection Classifier for UDP Traffic. **IEEE/ACM Transactions on Networking**, v. 18, n. 5, p. 1505–1515, 2010.

FINKEL, J. R.; GRENAGER, T.; MANNING, C. Incorporating Non-local Information into Information Extraction Systems by Gibbs Sampling. In: **Proceedings of the 43rd Annual Meeting on ACL**. Stroudsburg, PA, USA: ACL, 2005. (Acl '05), p. 363–370. Available from Internet: <https://doi.org/10.3115/1219840.1219885>.

FOSTER, N. *et al.* Frenetic: A Network Programming Language. **SIGPLAN Not.**, ACM, New York, NY, USA, v. 46, n. 9, p. 279–291, sep. 2011. ISSN 0362-1340. Available from Internet: <http://doi.acm.org/10.1145/2034574.2034812>.

FOSTER, N. *et al.* Using Deep Programmability to Put Network Owners in Control. **SIGCOMM CCR**, ACM, New York, NY, USA, v. 50, n. 4, p. 82–88, oct. 2020. ISSN 0146-4833. Available from Internet: <https://doi.org/10.1145/3431832.3431842>.

FOUNDATION., T. O. I. S. **Project Clearwater**. 2018. Available from Internet: <https://suricata-ids.org/>.

FRIEDMAN, J. H. Greedy function approximation: A gradient boosting machine. **The Annals of Statistics**, Institute of Mathematical Statistics, v. 29, n. 5, p. 1189 – 1232, 2001. Available from Internet: <https://doi.org/10.1214/aos/1013203451>.

GALLOPENI, G. *et al.* A Practical Analysis on Mirai Botnet Traffic. In: **2020 IFIP Networking Conference (Networking)**. [S.l.: s.n.], 2020. p. 667–668.

GEIRHOS, R. *et al.* Shortcut learning in deep neural networks. **Nature Machine Intelligence**, v. 2, n. 11, p. 665–673, Nov 2020. ISSN 2522-5839. Available from Internet: <https://doi.org/10.1038/s42256-020-00257-z>.

GHORBANI, A.; ABID, A.; ZOU, J. Interpretation of Neural Networks Is Fragile. **Proceedings of the AAAI Conference on Artificial Intelligence**, v. 33, n. 01, p. 3681–3688, Jul. 2019. Available from Internet: <https://ojs.aaai.org/index.php/AAAI/article/view/4252>.

GHOSH, P. **AAAS: Machine learning 'causing science crisis'**. BBC, 2019. Available at <https://www.bbc.com/news/science-environment-47267081>. Available from Internet: <https://www.bbc.com/news/science-environment-47267081>.

GUO, W. *et al.* LEMNA: Explaining Deep Learning Based Security Applications. In: **Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security**. New York, NY, USA: Association for Computing Machinery, 2018. (CCS '18), p. 364–379. ISBN 9781450356930. Available from Internet: <https://doi.org/10.1145/3243734.3243792>.

GUPTA, A. *et al.* Sonata: Query-Driven Streaming Network Telemetry. In: **Proceedings of the Annual Conference of the ACM SIGCOMM**. New York, NY, USA: ACM, 2018. (SIGCOMM '18), p. 357–371. ISBN 9781450355674. Available from Internet: <https://doi.org/10.1145/3230543.3230555>.

GUPTA, A.; MAC-STOKER, C.; WILLINGER, W. An Effort to Democratize Networking Research in the Era of AI/ML. In: **Proceedings of the 18th ACM Workshop on Hot Topics in Networks**. New York, NY, USA: Association for Computing Machinery, 2019. (HotNets '19), p. 93–100. ISBN 9781450370202. Available from Internet: <https://doi.org/10.1145/3365609.3365857>.

HALEPLIDIS, E. *et al.* **Software-Defined Networking (SDN): Layers and Architecture Terminology**. RFC Editor, 2015. RFC 7426. (Request for Comments, 7426). Available from Internet: <https://rfc-editor.org/rfc/rfc7426.txt>.

HARABAGIU, S. M.; HICKL, A.; LACATUSU, V. F. Negation, Contrast and Contradiction in Text Processing. In: **AAAI**. [S.l.: s.n.], 2006.

HINTON, G.; VINYALS, O.; DEAN, J. **Distilling the Knowledge in a Neural Network**. arXiv, 2015. Available from Internet: <https://arxiv.org/abs/1503.02531>.

HOCHREITER, S.; SCHMIDHUBER, J. Long Short-Term Memory. **Neural Comput.**, MIT Press, Cambridge, MA, USA, v. 9, n. 8, p. 1735–1780, nov. 1997. ISSN 0899-7667. Available from Internet: <http://dx.doi.org/10.1162/neco.1997.9.8.1735>.

HOLLAND, J. *et al.* New Directions in Automated Traffic Analysis. In: **Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security**. New York, NY, USA: Association for Computing Machinery, 2021. (CCS '21), p. 3366–3383. ISBN 9781450384544. Available from Internet: <https://doi.org/10.1145/3460120.3484758>.

HORN, P. **Autonomic Computing: IBM's Perspective on the State of Information Technology**. [S.l.], 2001.

HUANG, X. *et al.* Safety Verification of Deep Neural Networks. In: MAJUMDAR, R.; KUNČAK, V. (Ed.). **Computer Aided Verification**. Cham: Springer International Publishing, 2017. p. 3–29. ISBN 978-3-319-63387-9.

HUAWEI. **Huawei Core Network Autonomous Driving Network White Paper**. 2019. Accessed at 2022-07-06. Available from Internet: <https://carrier.huawei.com/~/media/CNBGV2/download/adn/Huawei-Core-Network-Autonomous-Driving-Network-White-Paper.pdf>.

HUSSEIN, A. *et al.* Imitation Learning: A Survey of Learning Methods. **ACM Comput. Surv.**, Association for Computing Machinery, New York, NY, USA, v. 50, n. 2, abr. 2017. ISSN 0360-0300. Available from Internet: <https://doi.org/10.1145/3054912>.

INC., G. **Dialogflow**. 2018. <https://dialogflow.com/>. Available from Internet: <https://dialogflow.com/>.

JACOB, B. *et al.* **A Practical Guide to the IBM Autonomic Computing Toolkit**. IBM, International Support Organization, 2004. (IBM redbooks). ISBN 9780738498058. Available from Internet: <https://books.google.com.br/books?id=XHeoSgAACAAJ>.

JACOBS, A. S. *et al.* AI/ML for Network Security: The Emperor Has No Clothes. In: **Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security**. New York, NY, USA: Association for Computing Machinery, 2022. (CCS '22), p. 1537–1551. ISBN 9781450394505. Available from Internet: <https://doi.org/10.1145/3548606.3560609>.

JACOBS, A. S. *et al.* **Supplemental material**. 2022. Reproducibility artifacts available at <https://github.com/asjacobs92/emperor>. The TRUSTEE method was implemented in a Python library we called *scikit-explain* for ease of use, available at <https://github.com/asjacobs92/scikit-explain>. Available from Internet: <https://github.com/asjacobs92/emperor>.

JACOBS, A. S. *et al.* Affinity measurement for NFV-enabled networks: A criteria-based approach. In: **2017 IFIP/IEEE Symposium on Integrated Network and Service Management**. [S.l.: s.n.], 2017. (IM '17), p. 125–133.

JACOBS, A. S. *et al.* Refining Network Intents for Self-Driving Networks. In: **Proceedings of the Annual Conference of the ACM SIGCOMM Afternoon Workshop on Self-Driving Networks**. New York, NY, USA: ACM, 2018. (SelfDN 2018), p. 15–21. ISBN 978-1-4503-5914-6. Available from Internet: <http://doi.acm.org/10.1145/3229584.3229590>.

JACOBS, A. S. *et al.* Hey, Lumi! Using Natural Language for Intent-Based Network Management. In: **2021 USENIX Annual Technical Conference**. USENIX Association, 2021. (USENIX ATC 21)), p. 625–639. ISBN 978-1-939133-23-6. Available from Internet: <https://www.usenix.org/conference/atc21/presentation/jacobs>.

JACOBS, A. S. *et al.* Artificial neural network model to predict affinity for virtual network functions. In: **2018 IEEE/IFIP Network Operations and Management Symposium**. [S.l.: s.n.], 2018. (NOMS '18), p. 1–9. ISSN 2374-9709.

JING, N. *et al.* An efficient SVM-based method for multi-class network traffic classification. In: **30th IEEE International Performance Computing and Communications Conference**. [S.l.: s.n.], 2011. p. 1–8.

JOVANOVIĆ, A.; DANILOVIĆ, D. A new algorithm for solving the tree isomorphism problem. **Computing**, v. 32, n. 3, p. 187–198, 1984. Available from Internet: <https://doi.org/10.1007/BF02243572>.

JUNIPER. **The Self-Driving Network: The Future State of Operations for Cloud-Grade Networking**. [S.l.], 2017.

JURAFSKY, D.; MARTIN, J. H. **Speech and Language Processing**. 3rd. ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2019. ISBN 130950696. Available from Internet: <https://web.stanford.edu/~jurafsky/slp3/>.

KAUFFMANN, J. *et al.* **The Clever Hans Effect in Anomaly Detection**. 2020.

KAUR, H.; PANNU, H. S.; MALHI, A. K. A Systematic Review on Imbalanced Data Challenges in Machine Learning: Applications and Solutions. **ACM Comput. Surv.**, Association for Computing Machinery, New York, NY, USA, v. 52, n. 4, aug 2019. ISSN 0360-0300. Available from Internet: <https://doi.org/10.1145/3343440>.

KIM, H.; GUPTA, A. ONTAS: Flexible and scalable online network traffic anonymization system. In: **ACM SIGCOMM Workshop on Network Meets AI & ML**. [S.l.: s.n.], 2019. p. 15–21.

KIM, H. *et al.* Kinetic: Verifiable Dynamic Network Control. In: **12th USENIX Symposium on Networked Systems Design and Implementation**. Berkeley, CA, USA: USENIX Association, 2015. (NSDI '15), p. 59–72. ISBN 978-1-931971-218. Available from Internet: <http://dl.acm.org/citation.cfm?id=2789770.2789775>.

KOLEY, B. **The Zero Touch Network**. 2016. (CNSM '16). Keynote Speech at the 12th International Conference on Network and Service Management. <https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/45687.pdf>.

LAFFERTY., J. D.; MCCALLUM, A.; PEREIRA, F. C. N. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In: **Proceedings of the 18th International Conference on Machine Learning**. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001. (ICML '01), p. 282–289. ISBN 1-55860-778-1. Available from Internet: <http://dl.acm.org/citation.cfm?id=645530.655813>.

LAKKARAJU, H.; BASTANI, O. "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**. New York, NY, USA: Association for Computing Machinery, 2020. (AIES '20), p. 79–85. ISBN 9781450371100. Available from Internet: <https://doi.org/10.1145/3375627.3375833>.

LAKKARAJU, H. *et al.* **Interpretable & Explorable Approximations of Black Box Models**. 2017.

LAMPLE, G. *et al.* Neural Architectures for Named Entity Recognition. In: **Proceedings of the 2016 Conference of the North American Chapter of the ACL: Human Language Technologies**. San Diego, California: ACL, 2016. p. 260–270. Available from Internet: <http://www.aclweb.org/anthology/N16-1030>.

LANTZ, B.; HELLER, B.; MCKEOWN, N. A Network in a Laptop: Rapid Prototyping for Software-defined Networks. In: **Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks**. New York, NY, USA: ACM, 2010. (Hotnets-IX), p. 19:1–19:6. ISBN 978-1-4503-0409-2. Available from Internet: <http://doi.acm.org/10.1145/1868447.1868466>.

LAPUSCHKIN, S. *et al.* Unmasking Clever Hans predictors and assessing what machines really learn. **Nature Communications**, v. 10, n. 1, p. 1096, Mar 2019. ISSN 2041-1723. Available from Internet: <https://doi.org/10.1038/s41467-019-08987-4>.

LEMAîTRE, G.; NOGUEIRA, F.; ARIDAS, C. K. Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning. **Journal of Machine Learning Research**, v. 18, n. 17, p. 1–5, 2017. Available from Internet: <http://jmlr.org/papers/v18/16-365.html>.

LIMSOPATHAM, N.; COLLIER, N. Bidirectional LSTM for Named Entity Recognition in Twitter Messages. In: **Proceedings of the 2nd Workshop on Noisy User-generated Text, NUTCOLING 2016, Osaka, Japan, December 11, 2016**. [s.n.], 2016. p. 145–152. Available from Internet: <https://aclanthology.info/papers/W16-3920/w16-3920>.

LINGAM, V. *et al.* Deep learning for conflicting statements detection in text. **PeerJ Preprints**, v. 6, p. e26589v1, mar. 2018. ISSN 2167-9843. Available from Internet: <https://doi.org/10.7287/peerj.preprints.26589v1>.

LIPTON, Z. C. The Mythos of Model Interpretability: In Machine Learning, the Concept of Interpretability is Both Important and Slippery. **Queue**, Association for Computing Machinery, New York, NY, USA, v. 16, n. 3, p. 31–57, jun. 2018. ISSN 1542-7730. Available from Internet: <https://doi.org/10.1145/3236386.3241340>.

LIU, H. H. **The Practice of Network Verification in Alibaba's Global WAN**. 2021. <https://netverify.fun/the-practice-of-network-verification\-in-alibaba-global-wan/>. Available from Internet: <https://netverify.fun/the-practice-of-network-verification-in-alibaba-global-wan/>.

LUMI. **Lumi supplemental material**. 2020. <https://lumichatbot.github.io/>. Available from Internet: <https://lumichatbot.github.io/>.

LUNDBERG, S. M.; LEE, S. A Unified Approach to Interpreting Model Predictions. In: GUYON, I. *et al.* (Ed.). **Advances in Neural Information Processing Systems 30**. Curran Associates, Inc., 2017. p. 4765–4774. Available from Internet: <http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf>.

MACCARTNEY, B. *et al.* Learning to Recognize Features of Valid Textual Entailments. In: **Proceedings of the Main Conference of the North American Chapter of the ACL**. Stroudsburg, PA, USA: ACL, 2006. (NAACL '06), p. 41–48. Available from Internet: <https://doi.org/10.3115/1220835.1220841>.

MACHINERY, A. for C. **Artifact Review and Badging Version 1.1**. 2020. Available at <https://www.acm.org/publications/policies/artifact-review-and-badging-current>. Accessed in May 25th, 2021. Available from Internet: <https://www.acm.org/publications/policies/artifact-review-and-badging-current>.

MAO, H.; NETRAVALI, R.; ALIZADEH, M. Neural Adaptive Video Streaming with Pensieve. In: **Proceedings of the Conference of the ACM Special Interest Group on Data Communication**. New York, NY, USA: Association for Computing Machinery, 2017. (SIGCOMM '17), p. 197–210. ISBN 9781450346535. Code available at GitHub repository at <https://github.com/hongzimao/pensieve.> Available from Internet: <https://doi.org/10.1145/3098822.3098843>.

MARNEFFE, M. C. D.; RAFFERTY, A. N.; MANNING, C. D. Finding contradictions in text. In: **Proceedings of the Conference of the ACL**. [s.n.], 2008. (ACL '08), p. 1039–1047. Cited By 108. Available from Internet: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-84859888583&partnerID=40&md5=624f6d85fc589442d7af96db6236f74c>.

MARQUEZAN, C. C.; Z.GRANVILLE, L. **On the investigation of the joint use of self-* properties and peer-to-peer for network management.** [s.n.], 2010. Available from Internet: <https://search.ebscohost.com/login.aspx?direct=true&AuthType=shib&db=cat07377a&AN=sabi.000761898&lang=pt-br&scope=site&authtype=guest,shib&custid=s5837110&groupid=main&profile=eds>.

MARRERO, M. *et al.* Named Entity Recognition: Fallacies, Challenges and Opportunities. **Computer Standards & Interfaces**, v. 35, n. 5, p. 482–489, 2013. ISSN 0920-5489. Available from Internet: <http://www.sciencedirect.com/science/article/pii/S0920548912001080>.

MENG, Z. *et al.* Interpreting Deep Learning-Based Networking Systems. In: **Proceedings of the Annual Conference of the ACM SIGCOMM**. New York, NY, USA: Association for Computing Machinery, 2020. (SIGCOMM '20), p. 154–171. ISBN 9781450379557. Available from Internet: <https://doi.org/10.1145/3387514.3405859>.

MIKOLOV, T. *et al.* Efficient Estimation of Word Representations in Vector Space. **CoRR**, abs/1301.3781, 2013. Available from Internet: <http://dblp.uni-trier.de/db/journals/corr/corr1301.html\#abs-1301-3781>.

MIRSKY, Y. *et al.* Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection. In: **Network and Distributed System Security Symposium 2018**. [s.n.], 2018. (NDSS'18). Available from Internet: <https://arxiv.org/abs/1802.09089>.

MOGUL, J. Unsafe at Any Speed? Self-Driving Networks without Self-Crashing Networks. In: **Keynote Speech at the ACM SIGCOMM Afternoon Workshop on Self-Driving Networks**. New York, NY, USA: ACM, 2018. (SelfDN 2018).

MOLNAR, C. *et al.* **Pitfalls to Avoid when Interpreting Machine Learning Models**. 2020.

MOLONY, R. **How We Trained Overfit Models to Identify Malicious Activity**. 2020. Available at <https://www.crowdstrike.com/blog/how-we-trained-overfit-models-to-identify-malicious-activity/>. Available from Internet: <https://www.crowdstrike.com/blog/how-we-trained-overfit-models-to-identify-malicious-activity/>.

MONSANTO, C. *et al.* Composing Software Defined Networks. In: **10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)**. Lombard, IL: USENIX Association, 2013. p. 1–13. ISBN 978-1-931971-00-3. Available from Internet: <https://www.usenix.org/conference/nsdi13/technical-sessions/presentation/monsanto>.

NARCISI, G. **Juniper Networks CEO: 'The goal now is a self-driving network'**. 2020. Accessed at 2022-07-06. Available from Internet: <https://www.crn.com/news/networking/juniper-networks-ceo-the-goal-now-is-a-self-driving-network->.

National Academies of Sciences Engineering and Medicine. **Implications of Artificial Intelligence for Cybersecuriy: Proceedings of a Workshop**. Washington, DC: The National Academies Press, 2019. ISBN 978-0-309-49450-2. Available from Internet: <https://www.nap.edu/catalog/25488/implications-of-artificial-intelligence-for-cybersecurity-proceedings-of-a-workshop>.

NETWORKS, J. **What Is Intent-Based Networking?** <https://www.juniper.net/us/en/products-services/what-is/intent-based-networking/>. Available from Internet: <https://www.juniper.net/us/en/products-services/what-is/intent-based-networking/>.

NETWORKS, J. **Self-Driving Networks for Service Delivery**. 2018. Available from Internet: <https://www.slideshare.net/apnic/selfdriving-networks-for-service-delivery>.

OPENCONFIG. **OpenConfig**. 2016. <http://www.openconfig.net/>. Available from Internet: <http://www.openconfig.net/>.

PEDREGOSA, F. *et al.* Scikit-learn: Machine Learning in Python. **Journal of Machine Learning Research**, v. 12, p. 2825–2830, 2011. <http://scikit-learn.org>. Available from Internet: <http://scikit-learn.org>.

PIANO, S. L. Ethical principles in machine learning and artificial intelligence: cases from the field and possible ways forward. **Humanities and Social Sciences Communications**, v. 7, n. 1, p. 9, Jun 2020. ISSN 2662-9992. Available from Internet: <https://doi.org/10.1057/s41599-020-0501-9>.

PRABHU, S. *et al.* Plankton: Scalable network configuration verification through model checking. In: **17th USENIX Symposium on Networked Systems Design and Implementation**. Santa Clara, CA: USENIX Association, 2020. (NSDI '20). Available from Internet: <https://www.usenix.org/conference/nsdi20/presentation/prabhu>.

PRAKASH, C. *et al.* PGA: Using Graphs to Express and Automatically Reconcile Network Policies. In: **Proceedings of the Annual Conference of the ACM SIGCOMM**. New York, NY, USA: ACM, 2015. (SIGCOMM '15), p. 29–42. ISBN 978-1-4503-3542-3. Available from Internet: <http://doi.acm.org/10.1145/2785956.2787506>.

RESEARCH, M. **Hyperscale cloud reliability and the art of organic collaboration**. 2018. <https://www.microsoft.com/en-us/research/blog/hyperscale-cloud-reliability-and-the\-art-of-organic-collaboration>. Available from Internet: <https://www.microsoft.com/en-us/research/blog/hyperscale-cloud-reliability-and-the-art-of-organic-collaboration/>.

RIBEIRO, M.; SINGH, S.; GUESTRIN, C. "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: Association for Computational Linguistics, 2016. p. 97–101. Available from Internet: <https://aclanthology.org/N16-3020>.

RIISER, H. *et al.* Commute path bandwidth traces from 3G networks: analysis and applications. In: **Proceedings of the 4th ACM Multimedia Systems Conference**. [S.l.: s.n.], 2013. p. 114–118.

RITTER, A. *et al.* It's a Contradiction–no, It's Not: A Case Study Using Functional Relations. In: **Proceedings of the Conference on Empirical Methods in Natural Language Processing**. Stroudsburg, PA, USA: ACL, 2008. (Emnlp '08), p. 11–20. Available from Internet: <http://dl.acm.org/citation.cfm?id=1613715.1613718>.

ROSS, S.; GORDON, G. J.; BAGNELL, J. A. **A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning**. 2011.

RUDIN, C. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. **Nature Machine Intelligence**, v. 1, n. 5, p. 206–215, May 2019. ISSN 2522-5839. Available from Internet: <https://doi.org/10.1038/s42256-019-0048-x>.

RUSSELL, S.; NORVIG, P. **Artificial Intelligence: A Modern Approach**. 4rd. ed. USA: Pearson, 2020. ISBN 9780134610993.

RYZHYK, L. *et al.* Correct by Construction Networks Using Stepwise Refinement. In: **14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)**. Boston, MA: USENIX Association, 2017. p. 683–698. ISBN 978-1-931971-37-9. Available from Internet: <https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/ryzhyk>.

SARAFRAZ, F. **Finding conflicting statements in the biomedical literature**. Thesis (PhD) — University of Manchester, UK, 2012. Available from Internet: <http://www.manchester.ac.uk/escholar/uk-ac-man-scw:157382>.

SCHATZMANN, D. *et al.* Digging into HTTPS: Flow-Based Classification of Webmail Traffic. In: **Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement**. New York, NY, USA: Association for Computing Machinery, 2010. (IMC '10), p. 322–327. ISBN 9781450304832. Available from Internet: <https://doi.org/10.1145/1879141.1879184>.

SCHEID, E. J. *et al.* INSpIRE: Integrated NFV-based Intent Refinement Environment. In: **2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)**. [S.l.: s.n.], 2017. p. 186–194.

SHAPLEY, L. S. 17. A Value for n-Person Games. In: ____. **Contributions to the Theory of Games (AM-28), Volume II**. Princeton University Press, 2016. p. 307–318. Available from Internet: <https://doi.org/10.1515/9781400881970-018>.

SHARAFALDIN., I.; H. Lashkari., A.; GHORBANI., A. A. Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. In: INSTICC. **Proceedings of the 4th International Conference on Information Systems Security and Privacy - ICISSP,**. [S.l.]: SciTePress, 2018. p. 108–116. ISBN 978-989-758-282-0. ISSN 2184-4356.

SHAUKAT, K. *et al.* A Survey on Machine Learning Techniques for Cyber Security in the Last Decade. **IEEE Access**, v. 8, p. 222310–222354, 2020.

SIVANATHAN, A. *et al.* Classifying IoT Devices in Smart Environments Using Network Traffic Characteristics. **IEEE Transactions on Mobile Computing**, v. 18, n. 8, p. 1745–1759, 2019.

SOMMER, R.; PAXSON, V. Outside the Closed World: On Using Machine Learning for Network Intrusion Detection. In: **2010 IEEE Symposium on Security and Privacy**. [S.l.: s.n.], 2010. p. 305–316.

SOULÉ, R. *et al.* Merlin: A Language for Managing Network Resources. **IEEE/ACM Transactions on Networking**, v. 26, n. 5, p. 2188–2201, oct. 2018. ISSN 1063-6692.

STRASSNER, J. C.; AGOULMINE, N.; LEHTIHET, E. A. FOCALE: A Novel Autonomic Networking Architecture. In: **Latin American Autonomic Computing Symposium (LAACS)**. [S.l.: s.n.], 2006.

SUN, P. *et al.* **Scalability in Perception for Autonomous Driving: Waymo Open Dataset**. 2020. Available at <https://waymo.com/open/>.

SUN, Y. *et al.* CS2P: Improving Video Bitrate Selection and Adaptation with Data-Driven Throughput Prediction. In: **Proceedings of the 2016 ACM SIGCOMM Conference**. New York, NY, USA: Association for Computing Machinery, 2016. (SIGCOMM '16), p. 272–285. ISBN 9781450341936. Available from Internet: <https://doi.org/10.1145/2934872.2934898>.

SUNG, Y. E. *et al.* Modeling and Understanding End-to-End Class of Service Policies in Operational Networks. In: **Proceedings of the Annual Conference of the ACM SIGCOMM**. New York, NY, USA: ACM, 2009. (SIGCOMM '09), p. 219–230. ISBN 9781605585949. Available from Internet: <https://doi.org/10.1145/1592568.1592595>.

SUNG, Y. E. *et al.* Robotron: Top-down Network Management at Facebook Scale. In: **Proceedings of the Annual Conference of the ACM SIGCOMM**. New York, NY, USA: ACM, 2016. (SIGCOMM '16), p. 426–439. ISBN 978-1-4503-4193-6. Available from Internet: <http://doi.acm.org/10.1145/2934872.2934874>.

TAWFIK, N. S.; SPRUIT, M. R. Automated Contradiction Detection in Biomedical Literature. In: PERNER, P. (Ed.). **Machine Learning and Data Mining in Pattern Recognition**. Cham: Springer International Publishing, 2018. p. 138–148. ISBN 978-3-319-96136-1.

TOUTANOVA, K. *et al.* Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network. In: **Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the ACL**. [s.n.], 2003. p. 252–259. Available from Internet: <https://www.aclweb.org/anthology/N03-1033>.

TUREK, M. DARPA RSS, 2016. Available at <https://www.darpa.mil/program/explainable-artificial-intelligence>. Accessed on May 25th, 2021. Available from Internet: <https://www.darpa.mil/program/explainable-artificial-intelligence>.

VMWARE. **Intent-based Networking**. 2021. <https://www.vmware.com/topics/glossary/content/intent-based-networking>. Available from Internet: <https://www.vmware.com/topics/glossary/content/intent-based-networking>.

WANG, L. *et al.* Joint Optimization of Service Function Chaining and Resource Allocation in Network Function Virtualization. **IEEE Access**, v. 4, p. 8084–8094, 2016. ISSN 2169-3536.

WANG, W. *et al.* End-to-end encrypted traffic classification with one-dimensional convolution neural networks. In: **2017 IEEE International Conference on Intelligence and Security Informatics (ISI)**. [S.l.: s.n.], 2017. p. 43–48. Code available at GitHub repository <https://github.com/echowei/DeepTraffic.>

WATTS, S. **An Introduction to Self-Driving Networks**. 2020. Available from Internet: <https://www.bmc.com/blogs/self-driving-networks/#>.

WHITE, C. C.; WHITE, D. J. Markov decision processes. **European Journal of Operational Research**, v. 39, n. 1, p. 1–16, 1989. ISSN 0377-2217. Available from Internet: <https://www.sciencedirect.com/science/article/pii/0377221789903482>.

XIN, Y. *et al.* Machine Learning and Deep Learning Methods for Cybersecurity. **IEEE Access**, v. 6, p. 35365–35381, 2018.

XIONG, Z.; ZILBERMAN, N. Do Switches Dream of Machine Learning? Toward In-Network Classification. In: **Proceedings of the 18th ACM Workshop on Hot Topics in Networks**. New York, NY, USA: Association for Computing Machinery, 2019. (HotNets '19), p. 25–33. ISBN 9781450370202. Available from Internet: <https://doi.org/10.1145/3365609.3365864>.

YADAV, V.; BETHAR, S. A Survey on Recent Advances in Named Entity Recognition from Deep Learning model. In: **Proceedings of the 27th International Conference on Computational Linguistic**. ACL, 2018. p. 2145–215. Available from Internet: <http://aclweb.org/anthology/C18-118>.

ZHANG, H. *et al.* An effective convolutional neural network based on SMOTE and Gaussian mixture model for intrusion detection in imbalanced dataset. **Computer Networks**, v. 177, p. 107315, 2020. ISSN 1389-1286. Available from Internet: <https://www.sciencedirect.com/science/article/pii/S1389128620300712>.

ZHANG, K.; SHASHA, D. Simple Fast Algorithms for the Editing Distance between Trees and Related Problems. **SIAM J. Comput.**, Society for Industrial and Applied Mathematics, USA, v. 18, n. 6, p. 1245–1262, dec 1989. ISSN 0097-5397. Available from Internet: <https://doi.org/10.1137/0218082>.

ZHANGA, J. *et al.* Internet Traffic Classification by Aggregating Correlated Naive Bayes Predictions. **IEEE Transactions on Information Forensics and Security**, v. 8, n. 1, p. 5–15, 2013.

APPENDIX A — ETHICAL CONCERNS

In Chapter 4, we describe a dataset collected and curated in the campus network from the University of California Santa Barbara (UCSB). While collecting that dataset does not have any activity that directly involves human subjects, it does involve packet traces and associated IDS alerts from the campus network, which may contain personally identifiable information, or PII. Thus, we performed a proper anonymization process for this dataset. Below, we explain the steps taken to conduct our research ethically.

The campus dataset removes any personally identifiable information. All of our data collection and research processes were reviewed and approved by the university's institutional review board (IRB). The UCSB performs separate reviews to obtain researcher access to administrative data. Since this dataset comes from an operational campus network, the data collection and research processes were also separately reviewed by a committee formed by the institute. The committee, which comprises campus stakeholders and IT experts, has approved to use this dataset for research purposes. All researchers who have or have had access to the campus dataset has gone through proper training before accessing and viewing the dataset. Moreover, all researchers and research projects are reviewed again, every two to three years. Figure A.1 shows the data collection pipeline we instrumented at our campus network. Researchers only had access to the collection server, which is colored in shaded-gray.

Anonymization effort: For the campus dataset, to avoid extracting any privacy-sensitive information, we only collected the five tuples from a packet and the set of IDS rules it triggered. To protect user anonymity, our network operator used a modified ONTAS (KIM; GUPTA, 2019) to anonymize all MAC addresses and local, or campus internal, IP addresses. The program anonymizes privacy-sensitive fields in packets at a programmable data plane at line-rate. Thus, the mirrored campus trace was anonymized *before* the packet trace was even captured at our collection server. We have preserved off-campus, or non-local, IP addresses. This is because our analysis depends on identifying certain Autonomous Systems (AS) or applications (*e.g.,* Zoom) in the Internet. Although a remote IP address theoretically can be used to identify a particular off-campus recipient in the Internet, we consider this sufficiently hard to do. Most IP addresses on the Internet belongs to large ISPs and corporations (*e.g.,* AT&T, Amazon, Google) or enterprise networks, and are mostly assigned dynamically using the DHCP protocol. Thus, given just a non-campus IP address, it requires considerable effort to be correctly and confidently

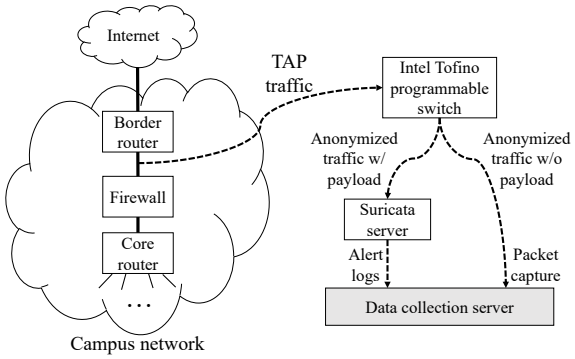


Figure A.1 – IDS alert and packet trace collection system.

pinpoint who exactly owned and used the IP address at a given time. Absolutely no effort was made by researchers to pinpoint or identify a particular user using a non-campus, external IP address.

Dataset management: Our campus dataset is stored at a secure infrastructure that is managed by professional IT staff. Only IRB-trained and approved researchers and authorized IT staff members are allowed to access the storage space. The dataset is strictly prohibited from being moved or copied out of the managed infrastructure in general.

Dataset curation: We curated a dataset of real-world traffic from our university campus network. The collected anonymized headers in nPrint format are available alongside the reproducibility assets of the paper (JACOBS *et al.*, 2022b).

We mirrored to our servers all the traffic of the main campus up-link which destination or source IP addresses were in the university's internal network. We also installed the Suricata IDS 6.0.4 with publicly available rules and selected a subset of rules to be captured, aiming to find the same attacks as were presented in the CIC-IDS-2017 dataset (see corresponding rules in *enable.conf* and *disable.conf* files of the dataset). After that, we configured the mirrored traffic to be sent to the Suricata IDS host without any modifications for intrusion analysis, and at the same time, we configured our equipment to truncate the packets leaving only the first 93 bytes and save remaining headers. The resulting truncated PCAPs and Suricata's *fast.log* make up the dataset for the analysis.

Using the previously described setup, we captured 12 hours of live network traffic during one of the working days in January 2022. We anonymized all the IP addresses and preprocessed the PCAPs with nPrint according to the original nPrint IDS case preprocessing. For the resulting dataset, we considered any flow that triggered the alert from Suricata IDS as an attack of the corresponding type, so port scans with different settings were all labeled simply as "port scan". For the analysis of nPrintML, we selected a balanced subset of each category that we observed during the traffic capture.

APPENDIX B — SCIENTIFIC PRODUCTION

The research work presented in this thesis was reported to the scientific community through paper submissions to renowned conferences and journals. The process of doing research, submitting papers, gathering feedback, and improving the work helped to achieve the maturity hereby presented.

B.1 Papers: Published and Ongoing Work

The following list shows the main papers and articles related to this thesis.

1. **Refining network intents for self-driving networks.** Arthur Selle Jacobs, Ricardo José Pfitscher, Ronaldo Alves Ferreira, Lisandro Zambenedetti Granville. Proceedings of the Annual Conference of the ACM SIGCOMM Afternoon Workshop on Self-Driving Networks (SelfDN 2018). Budapest, Hungary, 2018, pp 15-21. <http://doi.acm.org/10.1145/3229584.3229590>

    - **Abstract**: Recent advances in artificial intelligence (AI) offer an opportunity for the adoption of self-driving networks. However, network operators or home-network users still do not have the right tools to exploit these new advancements in AI, since they have to rely on low-level languages to specify network policies. Intent-based networking (IBN) allows operators to specify high-level policies that dictate how the network should behave without worrying how they are translated into configuration commands in the network devices. However, the existing research proposals for IBN fail to exploit the knowledge and feedback of the network operator to validate or improve the translation of intents. In this paper, we introduce a novel intent-refinement process that uses machine learning and feedback from the operator to translate the operator's utterances into network configurations. Our refinement process uses a sequence-to-sequence learning model to extract intents from natural language and the feedback from the operator to improve learning. The key insight of our process is an intermediate representation that resembles natural language that is suitable to collect feedback from the operator but is structured enough to facilitate precise translations. Our prototype interacts with a network operator using natural language and translates the operator input to the

intermediate representation before translating to SDN rules. Our experimental results show that our process achieves a correlation coefficient squared (i.e., R-squared) of 0.99 for a dataset with 5000 entries and the operator feedback significantly improves the accuracy of our model.

- **Type**: Workshop paper.
- **Status**: Published.
- **Qualis**: None.
- **Award**: Best paper.

2. **Refining network intents for self-driving networks.** Arthur Selle Jacobs, Ricardo José Pfitscher, Ronaldo Alves Ferreira, Lisandro Zambenedetti Granville. ACM SIGCOMM Computer Communication Review, Volume 48, Issue 5, 2018, pp 55–63. <https://doi.org/10.1145/3310165.3310173>

   - **Abstract**: Recent advances in artificial intelligence (AI) offer an opportunity for the adoption of self-driving networks. However, network operators or home-network users still do not have the right tools to exploit these new advancements in AI, since they have to rely on low-level languages to specify network policies. Intent-based networking (IBN) allows operators to specify high-level policies that dictate how the network should behave without worrying how they are translated into configuration commands in the network devices. However, the existing research proposals for IBN fail to exploit the knowledge and feedback from the network operator to validate or improve the translation of intents. In this paper, we introduce a novel intent-refinement process that uses machine learning and feedback from the operator to translate the operator's utterances into network configurations. Our refinement process uses a sequence-to-sequence learning model to extract intents from natural language and the feedback from the operator to improve learning. The key insight of our process is an intermediate representation that resembles natural language that is suitable to collect feedback from the operator but is structured enough to facilitate precise translations. Our prototype interacts with a network operator using natural language and translates the operator input to the intermediate representation before translating to SDN rules. Our experimental results show that our process achieves a correlation coefficient squared (i.e.,

R-squared) of 0.99 for a dataset with 5000 entries and the operator feedback significantly improves the accuracy of our model.

- **Type**: Journal article.
- **Status**: Published.
- **Qualis**: A3.

3. **Deploying Natural Language Intents with Lumi.** Arthur Selle Jacobs, Ricardo José Pfitscher, Ronaldo Alves Ferreira, Lisandro Zambenedetti Granville, Sanjay Rao. Proceedings of the ACM SIGCOMM 2019 Conference Posters and Demos. Beijing, China, 2019, pp 82–84. <https://doi.org/10.1145/3342280.3342315>

   - **Type**: Poster.
   - **Status**: Published.
   - **Qualis**: None.

4. **Hey, Lumi! Using Natural Language for Intent-Based Network Management.** Arthur Selle Jacobs, Ricardo José Pfitscher, Rafael Hengen Ribeiro, Ronaldo Alves Ferreira, Lisandro Zambenedetti Granville, Walter Willinger, Sanjay Rao. 2021 USENIX Annual Technical Conference (USENIX ATC 21). Virtual Conference, 2021, pp 625-639. <https://www.usenix.org/conference/atc21/presentation/jacobs>

   - **Abstract**: In this work, we ask: what would it take for, say, a campus network operator to tell the network, using natural language, to "Inspect traffic for the dorm"? How could the network instantly and correctly translate the request into low-level configuration commands and deploy them in the network to accomplish the job it was "asked" to do? We answer these questions by presenting the design and implementation of Lumi, a new system that (i) allows operators to express intents in natural language, (ii) uses machine learning and operator feedback to ensure that the translated intents conform with the operator's goals, and (iii) compiles and deploys them correctly in the network. As part of Lumi, we rely on an abstraction layer between natural language intents and network configuration commands referred to as Nile (Network Intent LanguagE). We evaluate Lumi using synthetic and real campus network policies and show that Lumi extracts entities with high precision and compiles intents in a few milliseconds. We also report on a user study where 88.5% of par-

ticipants state they would rather use Lumi exclusively or in conjunction with configuration commands.

- **Type**: Conference paper.
- **Status**: Published.
- **Qualis**: A1.

5. **AI-ML and Network Security: The Emperor has no Clothes.** Arthur Selle Jacobs, Roman Beltiukov, Walter Willinger, Ronaldo Alves Ferreira, Arpit Gupta, Lisandro Zambenedetti Granville. ACM Conference on Computer and Communications Security (CCS) 2022. Los Angeles, CA, USA, 2022, pp 1537–1551.

    - **Abstract**: Several recent research efforts have proposed Machine Learning (ML)-based solutions that can detect complex patterns in network traffic for a wide range of network security problems. However, without understanding how these black-box models are making their decisions, network operators are reluctant to trust them and deploy them in their production settings. One reason for this reluctance is a problem called underspecification, defined here as the failure to specify (an ML model) in adequate detail.In practice, this problem manifests itself in ML models that exhibit unexpectedly poor behavior when deployed in real-world settings and is not unique to the network security domain. Common among modern ML models, addressing this problem has prompted growing interest in developing interpretable ML solutions such as decision trees that can "explain" how a given black-box model makes its decisions and that can be easily interpreted by humans, at least in theory. However, synthesizing decision trees that capture a given black-box model's decisions with high fidelity while also being manageable in practice (*i.e.,* small enough in size for humans to comprehend) is challenging. In this paper, we focus on synthesizing high-fidelity and low-complexity decision trees to help network operators determine if their ML models suffer from the problem of underspecification. To this end, we develop TRUSTEE, a framework that takes an existing ML model and training dataset as input and generates a high-fidelity, easy-to-interpret decision tree and associated trust report as output. Using publicly available datasets and published learning models, we show how practitioners can use TRUSTEE to identify three common instances of model underspecification; *i.e.,* evidence of shortcut learning, presence of spurious correlations,

and vulnerability to out-of-distribution samples.

- **Status**: Published.
- **Qualis**: A1.

B.2 Other Works and Collaborations: Published and Ongoing Work

The following list shows the main collaborations that, although not directly related to this thesis proposal, are linked to the design of network management solutions and related topics.

1. **Affinity measurement for NFV-enabled networks: A criteria-based approach. Arthur Selle Jacobs**, Ricardo Luis dos Santos, Ricardo José Pfitscher, Muriel Figueredo Franco, Éder John Scheid, Lisandro Zambenedetti Granville. 2017 IFIP/ IEEE Symposium on Integrated Network and Service Management (IM 2017), Lisbon, Portugal, 2017, pp. 125-133. <http://doi.acm.org/10.23919/INM.2017.7987272>

   - **Type**: Conference paper.
   - **Status**: Published.
   - **Qualis**: A3.

2. **Artificial neural network model to predict affinity for virtual network functions. Arthur Selle Jacobs**, Ricardo José Pfitscher, Ricardo Luis dos Santos, Muriel Figueredo Franco, Éder John Scheid, Lisandro Zambenedetti Granville. 2018 IEEE/IFIP Network Operations and Management Symposium (NOMS 2018). Taipei, Taiwan, 2018, pp 1-9. <http://doi.acm.org/10.1109/noms.2018.8406253>

   - **Type**: Conference paper.
   - **Status**: Published.
   - **Qualis**: A3.

3. **A model for quantifying performance degradation in virtual network function service chains.** Ricardo José Pfitscher, **Arthur Selle Jacobs**, Eder John Scheid, Muriel Figueredo Franco, Ricardo Luis dos Santos, Alberto Egon Schaeffer-Filho, and Lisandro Zambenedetti Granville. 2018 IEEE/IFIP Network Operations and

Management Symposium (NOMS 2018). Taipei, Taiwan, 2018, pp 1-9. <http://doi.acm.org/10.1109/noms.2018.8406253>

- **Type**: Conference paper.
- **Status**: Published.
- **Qualis**: A3.

4. **IDEAFIX: Identifying Elephant Flows in P4-based IXP Networks.** Marcus Vinicius Brito da Silva, **Arthur Selle Jacobs**, Ricardo José Pfitscher, Lisandro Zambenedetti Granville. 2018 IEEE Global Communications Conference (GLOBECOM). Abu Dhabi, United Arab Emirates, 2018, pp 1-6. <http://doi.acm.org/10.1109/GLOCOM.2018.8647685>

   - **Type**: Conference paper.
   - **Status**: Published.
   - **Qualis**: A1.

5. **Predicting elephant flows in internet exchange point programmable networks.** Marcus Vinicius Brito da Silva, **Arthur Selle Jacobs**, Ricardo José Pfitscher, Lisandro Zambenedetti Granville. International Conference on Advanced Information Networking and Applications (AINA). Matsue, Japan, 2019, pp 485-497. <https://doi.org/10.1007/978-3-030-15032-7_41>

   - **Type**: Conference paper.
   - **Status**: Published.
   - **Qualis**: A2.

6. **Guiltiness: a practical approach for quantifying virtual network functions performance.** Ricardo José Pfitscher, **Arthur Selle Jacobs**, Luciano Zembruzki, Ricardo Luis dos Santos, Eder John Scheid, Muriel Figueredo Franco, Alberto Schaeffer-Filho, Lisandro Zambenedetti Granville. Computer Networks, Volume 161, 2019, pp 14-31. <https://doi.org/10.1016/j.comnet.2019.06.001>

   - **Type**: Journal article.
   - **Status**: Published.
   - **Qualis**: A1.

7. **Sample Selection Search to Predict Elephant Flows in IXP Programmable Networks.** Marcus Vinicius Brito da Silva, André Augusto Pacheco de Carvalho, **Arthur Selle Jacobs**, Ricardo José Pfitscher, Lisandro Zambenedetti Granville. International Conference on Advanced Information Networking and Applications (AINA). Caserta, Italy, 2020, pp 357-368. <https://doi.org/10.1007/978-3-030-44041-1_33>

    - **Type**: Conference paper.
    - **Status**: Published.
    - **Qualis**: A2.

8. **A Bottom-up Approach for Extracting Network Intents.** Rafael Hengen Ribeiro, **Arthur Selle Jacobs**, Ricardo Parizotto, Luciano Zembruzki, Alberto Egon Schaeffer-Filho, Lisandro Zambenedetti Granville. International Conference on Advanced Information Networking and Applications (AINA). Caserta, Italy, 2020, pp 858-870. <https://doi.org/10.1007/978-3-030-44041-1_75>

    - **Type**: Conference paper.
    - **Status**: Published.
    - **Qualis**: A2.

9. **dnstracker: Measuring Centralization of DNS Infrastructure in the Wild.** Luciano Zembruzki, **Arthur Selle Jacobs**, Gustavo Spier Landtreter, Lisandro Zambenedetti Granville, Giovane Moura. International Conference on Advanced Information Networking and Applications (AINA). Caserta, Italy, 2020, pp 871-882. <https://doi.org/10.1007/978-3-030-44041-1_76>

    - **Type**: Conference paper.
    - **Status**: Published.
    - **Qualis**: A2.

10. **ShadowFS: Speeding-up Data Plane Monitoring and Telemetry using P4.** Ricardo Parizotto, Lucas Castanheira, Rafael Hengen Ribeiro, Luciano Zembruzki, **Arthur Selle Jacobs**, Lisandro Zambenedetti Granville, Alberto Schaeffer-Filho. 2020 IEEE International Conference on Communications (ICC). Dublin, Ireland, 2020, pp 1-6. <https://doi.org/10.1109/ICC40277.2020.9148954>

- **Type**: Conference paper.
- **Status**: Published.
- **Qualis**: A1.

11. **SecBot: a Business-Driven Conversational Agent for Cybersecurity Planning and Management.** Muriel Figueredo Franco, Bruno Rodrigues, Eder John Scheid, **Arthur Selle Jacobs**, Christian Killer, Lisandro Zambenedetti Granville, Burkhard Stiller. 2020 16th International Conference on Network and Service Management (CNSM). Izmir, Turkey, 2020, pp 1-7. <https://doi.org/10.23919/CNSM50824.2020.9269037>

    - **Type**: Conference paper.
    - **Status**: Published.
    - **Qualis**: A3.

12. **Hosting Industry Centralization and Consolidation.** Luciano Zembruzki, Raffaele Sommese, Lisandro Zambenedetti Granville, **Arthur Selle Jacobs**, Mattijs Jonker, Giovane Moura. 2022 IEEE/IFIP Network Operations and Management Symposium (NOMS 2022). Budapest, Hungary, 2022.

    - **Type**: Conference paper.
    - **Status**: Published.
    - **Qualis**: A3.

13. **DWT in P4: Periodicity Detection in the Data Plane.** Briggette Roman Huaytalla, **Arthur Selle Jacobs**, Marcus Vinicius Brito da Silva, Fabrício Barbosa Carvalho, Ronaldo Alves Ferreira, Lisandro Zambenedetti Granville, Walter Willinger. 2022 IEEE Global Communications Conference (GLOBECOM 2022). Rio de Janeiro, Brazil, 2022.

    - **Type**: Conference paper.
    - **Status**: Accepted.
    - **Qualis**: A1.

14. **On the Consolidation of the Internet Domain Name System.** Luciano Zembruzki, **Arthur Selle Jacobs**, Lisandro Zambenedetti Granville. 2022 IEEE Global Communications Conference (GLOBECOM 2022). Rio de Janeiro, Brazil , 2022.

- **Type**: Conference paper.
- **Status**: Accepted.
- **Qualis**: A1.

APPENDIX C — AWARDS

Throughout the course of the doctoral studies, the present student and research topic were granted the following awards.

1. **2020 IBM PhD Fellowship Award.** Arthur Selle Jacobs

    - **Year**: 2020.
    - **Duration**: 2 years.
    - **Description**: For 70 years IBM has recognized and rewarded outstanding PhD students around the world through a highly competitive IBM PhD Fellowship Award program. The distinguished 2021 IBM PhD Fellowship Award recipients demonstrated expertise in pioneering research areas, such as artificial intelligence, hybrid cloud technology, quantum computing, data science, security, and the next generation of cutting-edge processors. The 2020 IBM PhD Fellowship Award Program received hundreds of applications from 183 universities in 32 countries. Applications were reviewed by eminent technologists from across IBM. The award recipients demonstrated academic excellence as well as provided innovative, exceptional research proposals.

APPENDIX D — RESUMO ESTENDIDO

Conforme as redes modernas crescem em tamanho e complexidade, elas também se tornam cada vez mais sujeitas a erros humanos (BECKETT *et al.*, 2017; FEAMSTER; REXFORD, 2018). Essa tendência tem levado a indústria e a academia a tentar automatizar as tarefas de gerenciamento e controle, com o objetivo de reduzir a interação humana com a rede e erros humanos (BECKETT *et al.*, 2016; APOSTOLOPOULOS, 2020; NETWORKS, ; VMWARE, 2021). Idealmente, os pesquisadores imaginam um projeto de rede que não seja apenas automático (*i.e.,* , dependente de instruções humanas), mas autônomo (*i.e.,* , capaz de tomar suas próprias decisões). A rede autônoma tem sido uma meta buscada há anos, com diversos conceitos, projetos e implementações, mas nunca foi totalmente realizada, principalmente devido às limitações tecnológicas (FEAMSTER; REXFORD, 2018). Avanços recentes em Inteligência Artificial (IA) e Aprendizado de Máquina (*Machine Learning* — ML) introduziram ar fresco neste conceito, ressurgindo como o conceito renomeado de *redes autodirigidas*, em vista de suas contrapartes de automóveis autodirigidos.

D.1 Redes Autodirigidas

Embora o conceito de redes autodirigidas não tenha uma definição padronizada, com cada empresa e pesquisador tendo sua própria visão e arquitetura (MOGUL, 2018; FEAMSTER; REXFORD, 2018; JUNIPER, 2017; NETWORKS, 2018; HUAWEI, 2019; WATTS, 2020; FOSTER *et al.*, 2020), alguns elementos de design são comuns em todas as instâncias. Como nenhuma definição padrão foi adotada como padrão pela comunidade, contamos com a seguinte definição de rede autônoma, resumindo os principais aspectos encontrados na literatura, que está detalhadamente descrita no Capítulo 2.

Definition D.1.1. Uma rede autodirigidas é uma rede autônoma capaz de agir de acordo com as intenções de alto nível de uma operadora e se adaptar automaticamente às mudanças no tráfego e no comportamento dos usuários. Para alcançar essa visão, uma rede precisaria cumprir quatro requisitos principais: *(i)* compreender as intenções de alto nível de um operador para ditar seu comportamento, *(ii)* monitorar-se com base nas intenções de entrada, *(iii)* prevêr e identificar padrões em dados monitorados e *(iv)* adaptar-se a novos comportamentos sem a intervenção de um operador.

Na Figura D.1, apresentamos um projeto de alto nível de uma rede autodirigida que resume as características e requisitos encontrados na literatura.

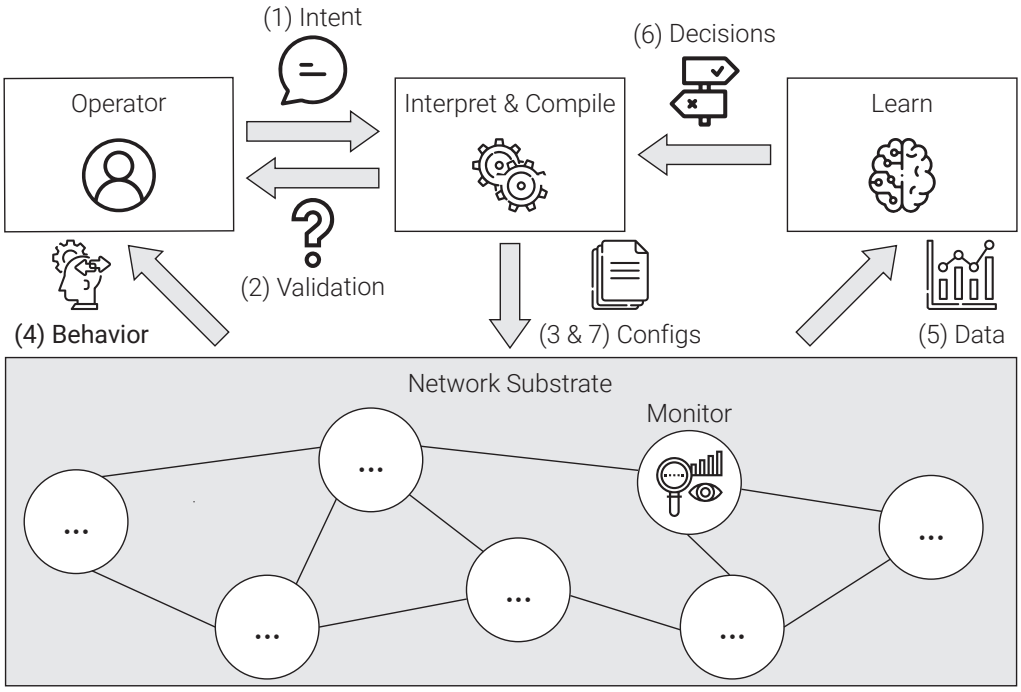


Figure D.1 – Projeto de uma rede autodirigida.

Este projeto de rede autodirigida é composto de dois ciclos de gerenciamento. No lado esquerdo da Figura D.1, vemos o primeiro ciclo de gerenciamento (1, 2, 3 e 4), que começa com um operador (1) especificando intenções de alto nível que ditam como a rede deve se comportar — *e.g.,* , definindo metas relacionadas à qualidade de serviço, segurança e desempenho - sem se preocupar com os detalhes de baixo nível que são necessários para programar a rede para atingir essas metas (também conhecido como redes baseada em intenção, ou *Intent-Based Networking* — IBN) (CLEMM *et al.*, 2019). Idealmente, especificar essas intenções deve ser tão fácil quanto descrevê-las em linguagem natural. No entanto, permitir que os operadores usem livremente a linguagem natural para descrever as intenções da rede requer que a rede empregue técnicas de ML de última geração do Processamento de Linguagem Natural (PLN) (JURAFSKY; MARTIN, 2019), que estão sempre sujeitas a gerar erros e classificações incorretas. Portanto, depois de extrair informações relevantes das intenções de entrada, uma rede autodirigida iria então (2) validar os dados extraídos com o operador antes de (3) compilá-los em configurações reais e implantá-los no substrato de rede. Para fechar o primeiro ciclo de gerenciamento, a rede então monitoraria a si mesma de acordo com as intenções descritas, e coletaria o comportamento do tráfego para (4) apresentá-lo aos operadores para verificar se o comportamento implementado corresponde aos objetivos iniciais.

No lado direito da Figura D.1, o segundo ciclo de gerenciamento (5, 6 e 7) começa depois que as intenções são implantadas no substrato da rede, onde os dispositivos são instrumentados com recursos de monitoramento para coletar o uso e o tráfego dados. Esses dados são então (5) analisados e processados para produzir (6) decisões (autônomas) usando modelos de ML treinados, que idealmente se adaptariam e se retreinariam constantemente à medida que novos dados fossem coletados. As decisões tomadas por tais modelos de ML seriam então processadas e (7) compiladas em configurações para ajustar o comportamento da rede (semelhante a 3), fechando o segundo ciclo de gerenciamento. Mais notavelmente, duas áreas de rede seriam as que mais se beneficiariam desse ciclo de gerenciamento: segurança cibernética e desempenho da rede. Por exemplo, as decisões dos modelos de ML podem incluir a identificação do tráfego de ataque que precisa ser bloqueado ou mitigado, ou até mesmo otimizações de uso de recursos com base na carga de tráfego. Observem que, apesar de também depender fortemente de técnicas de ML propensas a erros, dado o período de tempo e a frequência com que tais decisões e implantação de configurações ocorreria para acompanhar o tráfego de entrada, seria impossível incluir validação humana neste segundo ciclo de gerenciamento, em contraste com o primeiro. Além disso, outro aspecto importante a ser lembrado é que autoconfigurações feitas com base em decisões de modelos de ML no segundo ciclo de gerenciamento podem prejudicar e contradizer as configurações feitas por meio de intents de rede no primeiro ciclo de gerenciamento. A possibilidade de tais conflitos surgirem, juntamente com a incapacidade de incluir validações feitas por humanos, aumenta as possíveis consequências para qualquer decisão tomada por modelos de ML no segundo ciclo de gerenciamento.

D.2 Definição de Problema e Perguntas de Pesquisa

Como os dois ciclos de gerenciamento na rede autodirigida apresentada na Figura D.1 dependem fortemente de modelos de ML para tomar decisões e classificações que afetam diretamente a rede, um problema específico se torna proeminente com este design: confiança. Aplicar ML para resolver tarefas de gerenciamento de rede, como as descritas acima, tem sido uma tendência popular entre os pesquisadores recentemente (BOUTABA *et al.*, 2018). No entanto, apesar do tópico receber muita atenção, os operadores da indústria têm relutado em tirar proveito de tais soluções, principalmente por causa da natureza de caixa preta dos modelos de ML que produzem decisões sem qualquer explicação ou

razão para as quais essas decisões foram tomadas. Dada a natureza de alto risco das redes de produção, torna-se impossível confiar em um modelo de ML que pode tomar ações inadequadas automaticamente e, o mais importante para o escopo desta tese, um desafio proibitivo que deve ser abordado para que uma rede autodirigidas seja alcançada.

Definição de Problema: *A presente tese visa habilitar redes autônomas com ML, abordando o problema da inerente falta de confiança nos modelos de ML que as capacitam, avaliando seus processos de tomada de decisão.*

Para abordar o problema descrito acima, devemos primeiro investigar e avaliar a precisão e a credibilidade das classificações feitas pelos modelos de ML usados para processar intenções de alto nível do operador. Em seguida, devemos analisar e avaliar a precisão e credibilidade das decisões tomadas pelos modelos de ML usados para autoconfigurar a rede de acordo com os dados monitorados. Por fim, devemos investigar se existe um método viável para melhorar a confiança dos operadores nas decisões tomadas pelos modelos de ML em ambos ciclos de gerênciamento. Para pavimentar o caminho para a resolução do problema acima, formulamos três perguntas de pesquisa que orientam o desenvolvimento desta tese. A primeira pergunta de pesquisa que esta tese apresenta diz respeito ao uso de técnicas de ML para analisar informações relevantes de intenções de rede de entrada e compilá-las em uma configuração de rede que atenda às intenções dadas.

PP-1: *Em uma rede autodirigida, os operadores podem confiar nas decisões e classificações feitas pelos modelos de aprendizado de máquina atuais usados para configurar a rede com base em intenções de alto nível?*

A primeira pergunta de pesquisa é abordada no Capítulo 3. Para tanto, é preciso avaliar a confiabilidade e a usabilidade de um sistema habilitado para IBN que permite aos operadores de rede descrever as intenções da rede usando linguagem natural livremente e compilá-las na configuração da rede de acordo, conforme mostrado no primeiro ciclo de gerenciamento do rede de condução apresentada na Figura D.1. No entanto, ao revisar a literatura existente, evidenciamos que nenhum sistema de ponta-a-ponta foi proposto para permitir isso. Portanto, nosso primeiro passo para responder **PP-1** é propor LUMI, um assistente de conversação (*i.e., chatbot*) que permite aos operadores usarem linguagem natural para descrever intenções de rede de alto nível. LUMI utiliza de técnicas de ML altamente precisas de PLN para extrair informações de intenções de entrada e usa sua interface de conversação para confirmar com os operadores se as informações extraidas estão corretas. Para obter a confirmação dos operadores, propomos a *Network Intent Language* (Nile), uma linguagem de definições de intenções de alto nível que se assemelha ao in-

glês estruturado, usando-o como uma camada de abstração entre intenções em linguagem natural e comandos de configuração de rede. Mais importante ainda, LUMI é capaz de coletar *feedback* dos operadores nas intenções do *Nile* criadas com informações extraídas da linguagem natural e usa esse feedback para treinar novamente e melhorar a precisão do modelo de ML subjacente ao longo do tempo. Em nossa avaliação, mostramos que LUMI é capaz de extrair com precisão informações de inteções de entrada sintéticos e do mundo real, e realizamos um estudo de usuário com profissionais de rede de diferentes origens e níveis de especialização para avaliar sua usabilidade. Os resultados deste capítulo indicam que, com proteção mínima, os operadores podem, de fato, confiar nas classificações feitas pelos modelos de ML atuais usados para analisar informações de inteções de alto nível.

A segunda pergunta de pesquisa que esta tese coloca diz respeito ao uso de modelos de ML para tomar decisões com base em dados coletados dos dispositivos de substrato de rede, e configurar automaticamente a rede para ajustar e otimizar o uso de recursos e segurança.

PP-2: *Em uma rede autodirigida, os operadores podem confiar nas decisões e classificações feitas pelos modelos atuais de aprendizado de máquina usados para autoconfigurar a rede com base na análise de dados monitorados?*

A segunda pergunta de pesquisa é abordada no Capítulo 4. Para tanto, focamos nossos esforços na análise da aplicação de técnicas de ML em problemas de segurança de rede, como um caso de uso representando o segundo ciclo de gerenciamento da rede autônoma apresentada na Figura 1.1. Como mencionado anteriormente, muitos trabalhos que aplicam classificadores baseados em ML foram propostos no domínio da segurança de rede. Portanto, para responder a **PP-2**, pesquisamos a literatura existente para selecionar alguns trabalhos reprodutíveis baseados em ML para analisar e reproduzir seus resultados. Enquanto a maioria dos trabalhos na literatura relata uma excelente precisão de classificação em seus cenários de avaliação, estamos preocupados com a credibilidade das classificações feitas. Em particular, como não podemos esperar que operadores revisem todas as decisões tomadas pelos modelos de classificação em um ciclo de gerenciamento acelerado, se um modelo treinado com conjuntos de dados sintéticos ou irreais falhar quando apresentado a tráfego imprevisto do mundo real, isso pode levar a resultados desastrosos. Para esmiuçar os resultados dos trabalhos selecionados, contamos com o emergente campo de técnicas de IA eXplicável (IAX), que nos permitem extrair explicações de caixa-branca de modelos de caixa-preta. O método de IAX específico usado para examinar modelos é descrito posteriormente no Capítulo 5. Nossos resultados

mostram que, quando examinados, os modelos de ML analisados não cumprem suas tarefas, indicando que os operadores estão, de fato, corretos em não confiar nos modelos de ML para configurar automaticamente a rede com base apenas nos dados monitorados.

Finalmente, a terceira e última questão de pesquisa que esta tese apresenta diz respeito à falta de confiança dos operadores nos modelos de ML dos quais as redes autodirigidas dependem, e como essa confiança pode ser aumentada.

PP-3: *Existe uma maneira viável de aumentar a confiança do operador no processo de tomada de decisão de modelos de aprendizado de máquina que configuram uma rede autônoma?*

A terceira e última questão de pesquisa é respondida no Capítulo 5. Para responder **PP-3**, primeiro igualamos *"um operador que confia em um modelo de IA/ML"* com *"um operador que se sente confortável em abrir mão do controle para o modelo de IA/ML"* (LIPTON, 2018). Dada essa definição específica do que significa para um modelo de IA/ML gerar confiança, pesquisamos o estado da arte para encontrar métodos IAX para examinar modelos de ML e identificamos uma lacuna nos métodos IAX existentes para produzir explicações fiéis para qualquer dado modelo de caixa preta. Tais métodos aumentariam potencialmente a confiança dos operadores no processo de tomada de decisão dos modelos de ML, desde que as explicações produzidas fossem fiéis, de tamanho gerenciável e, acima de tudo, sólidas. Ao tentar preencher essa lacuna e responder **PP-3**, neste capítulo, propomos TRUSTEE, um novo método de IAX para produzir explicações de alta fidelidade na forma de árvores de decisão que aproximam a tomada de decisão processo a partir de modelos de caixa preta, independente de modelo. Além disso, também implementamos uma série de análises com base na explicação da árvore de decisão gerada e nos dados subjacentes, condensados em um *relatório de confiança*. Nossos resultados mostram que as explicações produzidas pelo TRUSTEE, juntamente com os *relatórios de confiança* gerados, permitem que operadores e especialistas de domínio examinem cuidadosamente os modelos de ML e decidam se são confiáveis ou sofrem de algum viés indutivo. Os resultados que obtivemos respondendo **PP-2** e **PP-3** confirmam nossa intuição de que os modelos de ML amplamente aplicados para resolver problemas de rede não foram submetidos ao escrutínio adequado e podem quebrar facilmente se aplicados em cenários de mundo real. Esses modelos devem ser corrigidos para cumprir suas tarefas adequadamente na prática, para que possamos um dia alcançar redes autodirigidas de fato.

D.3 Principais Contribuições

No processo de responder às três questões de pesquisa colocadas nesta tese, fazemos as seguintes contribuições:

- Propomos um sistema de gerenciamento de rede baseado em intenção de ponta a ponta com uma interface de conversação que permite aos operadores usar linguagem natural para descrever suas intenções desejadas para o comportamento da rede. Contamos com esse sistema para avaliar a precisão das técnicas atuais de ML na extração de informações relevantes de intents de alto nível, que podem ser verificadas manualmente pelos operadores por meio da interface de conversação.

- Revisamos a literatura existente sobre o uso de técnicas de ML para segurança de rede e examinamos vários casos de uso da literatura para analisar a credibilidade das decisões tomadas por modelos de ML altamente precisos que permitem uma rede autônoma. Os resultados que obtivemos revelam problemas sistemáticos sobre como os pesquisadores vêm empregando classificadores baseados em ML para resolver problemas de segurança de rede.

- Para aumentar a confiança e confiabilidade da abordagem atual aos modelos de ML, propomos um novo *pipeline* metodológico de como pesquisadores e operadores devem empregar e avaliar técnicas de ML para resolver problemas de rede. Para implementar nosso *pipeline* proposto, apresentamos um novo método de IAX para produzir explicações a partir de qualquer modelo de ML de caixa preta na forma de árvores de decisão. Também introduzimos um novo método de poda para garantir que as explicações extraídas sejam concisas, mantendo uma boa fidelidade, e um *relatório de confiança* que condensa e analisa os aspectos mais relevantes da explicação extraída para ajudar os especialistas do domínio a identificar problemas com os modelo de caixa preta.