Rigorous modeling and performance evaluation of networking systems

Siham Khoussi

Université Grenoble Alpes

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITE GRENOBLE ALPES

Spécialité : **Informatique**

Arrêté ministériel : 25 mai 2016

Présentée par

Siham Khoussi

Thèse dirigée par **Saddek BENSALEM**, Université Grenoble Alpes et co-encadrée par **Abdella BATTOU**, National Institute of Standards and Technology

préparée au sein du **Laboratoire VERIMAG**
dans **l'École Doctorale Mathématiques, Sciences et technologies de l'information, Informatique**

Modélisation et vérification formelle des performances des systèmes de réseau

Rigorous modeling and performance evaluation of networking systems

Thèse soutenue publiquement le **3 Décembre 2021**
devant le jury composé de:

M. Ahmed Lbath, Président du jury
Professor, Université Grenoble Alpes
M. Panagiotis Katsaros, Rapporteur du jury
Professor associé, Aristoteleio Panepistimio Thessalonikis
M. Axel Legay, Rapporteur du jury
Professor, Université catholique de Louvain
Mme. Erika Abraham, Examinatrice
Professor, Rheinisch-Westfaelische Tech. Hoch. Aa
M. Eugene Asarin, Examinateur
Professor, Université Paris 7 - Denis Diderot
M. Saddek Bensalem, Directeur de thèse
Professor, Université Grenoble Alpes,
M. Ayoub Nouri, Examinateur
Docteur en sciences, Huawei

Acknowledgements

First, I would like to express my most sincere gratitude and appreciation to my advisor, Professor Saddek Bensalem for giving me the opportunity to pursue my PhD under his supervision. Over the last few years, he provided me with invaluable research advice and continuous support during my PhD study.

I am, also, extremely grateful to my supervisor at the National Institute of Standards and Technology (NIST) Dr. Abdella Battou for his guidance and for providing me with a good research environment and the resources needed to do research. A special thank you to the lovely secretaries Cindy Messina and Jocelyn Malones from NIST for their assistance and encouraging words.

Additionally, I would like to thank all the administrative staff from the University of Grenoble, particularly Mrs Zilora Zouaoui for her help and assistance, remotely.

Next, I would like to thank my previous advisor Dr. Lotfi Benmohamed for his technical support and helpful advice during the initial phase of my PhD.

I would like to express my gratitude to Alan Heckert for his invaluable research advice and the technical support during the second part of my thesis.

Moreover, I am grateful to my collaborators including Ayoub Nouri, Davide Pesavento, Junxiao Shi, James Filliben. I have learnt many things from them that helped me improve and implement my work.

I am very grateful to all the reviewers and the members of the jury for accepting to review this work and for their valuable feedback on the manuscript.

Finally, I would like to thank my family and friends, particularly, my mother, my sister, my brother and my father (although no longer with us) for their support during my studies and for their affection. None of my accomplishments would have been possible without them.

Abstract

The demand for faster performance, increased accessibility, mobility and secure communications has driven significant advancements in Internet architectures, protocols and applications. Whether Internet usage relates to businesses or entertainment, its performance and security are two of the highest orders. Recent advances in modern technology and network innovations have driven the desire to move away from manual error-prone methods of testing network components and evolve from the ad-hoc tools and simulation based testing which are, traditionally, used in assessing the performance of networking components but fail to achieve high accuracy results and obtain trustworthy analysis.

Despite the criticism that formal verification (FV) methods have been receiving and lack of appreciation, they have achieved undeniable results and made great contributions in this field and other mature fields. For this reason, we investigate a FV methodology for analyzing the performance aspect of networking systems. We rely on a model-based approach that is based on building a rich faithful stochastic model of a system, then apply statistical model checking to assess its performance against a specified requirement. We explain that the stochastic behavior of the model is captured by introducing probabilistic variables which are updated via probability distributions. The latter are, typically, obtained by collecting and analyzing measurements from the system's execution using traditional statistical tests to select the best fit distribution (i.e., process of distribution fitting). Unfortunately, distribution fitting requires a good statistical background and familiarity with several distributions which is beyond the expertise of some analysts.

As such, we developed a tool called DeepFit that combines traditional statistical tests and deep learning to automate the distribution fitting task. DeepFit is then integrated into the workflow of our FV methodology for rigorous modeling and performance assessment of networking systems.

Contents

List of Figures

List of Tables

List of Abbreviations

SMC	**S**tatistical **M**odel **C**hecking
DTMC	**D**iscrete **T**ime **M**arkov **C**hain
MDP	**M**arkov **D**dcision **P**rocess
AP	**A**tomic **P**ropositions
LTS	**L**abeled **T**ransition **S**ystem
SSP	**S**ingle **S**ampling **P**lan
SPRT	**S**equential **P**robability **R**atio **T**est
BIP	**B**ehavior **I**nteraction **P**riority
SBIP	**S**tochastic **BIP**
DARPA	**D**efense **A**dvanced **R**esearch **P**rojects **A**gency
OSI	**O**pen **S**ystems **I**nterconnection
IP	**I**nternet **P**rotocol
UDP	**U**ser **D**atagram **P**rotocol
TCP	**T**ransmission **C**ontrol **P**rotocol
FTP	**F**ile **T**ransfer **P**rotocol
SMTP	**S**imple **M**ail **T**ransfer **P**rotocol
IT	**I**nformation **T**echnology
CNMS	**C**omputer **N**etwork **M**onitoring **S**ystem
CCNG	**C**omputer **C**ommunications **N**etworks **G**roup
ARPA	**A**dvanced **R**esearch **P**rojects **A**gency
RTT	**R**ound **T**rip **T**ime
RFC	**R**equests **F**or **C**omment
IETF	**I**nternet **E**ngineering **T**ask **F**orce
DNS	**D**omain **N**ame **S**ystem
IS	**I**nternet **S**ociety
ODL	**O**pen**D**ay**L**ight
VoIP	**V**oice **O**ver **IP**
ICANN	**I**nternet **C**orporation for **A**ssigned **N**ames and **N**umbers
SDN	**S**oftware **D**efined **N**etworking
ONOS	**O**pen **N**etwork **O**perating **S**ystem
NDN	**N**amed **D**ata **N**etworking
NSF	**N**ational **S**cience **F**oundation
QoS	**Q**uality **O**f **S**ervice
QoE	**Q**uality **O**f **E**xperience
FV	**F**ormal **V**erification
MC	**M**odel **C**heking
CNF	**C**onjunctive **N**ormal **F**orm
PD	**P**robability **D**istribution
HT	**H**ypothesis **T**esting
PE	**P**robability **E**stimation
LTL	**L**inear **T**emporal **L**olgic
BLTL	**B**ounded **L**inear **T**emporal **L**olgic
PBLTL	**P**robabilistic **B**ounded **L**inear **T**emporal **L**olgic

DL	**D**eep **L**earning
PDF	**P**robability **D**ensity **F**unction
PPF	**P**ercent **P**oint **F**unction
CDF	**C**umulative **D**istribution **F**unction
KDP	**K**ernel **D**ensity **P**lot
GoF	**G**oodness **O**f **F**it
PPCC	**P**robability **P**lot **C**orrelation **C**oefficient
KS	**K**olmogorov **S**mirnov
AD	**A**nderson **D**arling
AIC	**A**kaike's **I**nformation **C**riterion
BIC	**B**ayesian **I**nformation **C**riterion
CAT	**C**ategorical **C**ross **E**ntropy
MSE	**M**ean **S**quared **E**rror
NN	**N**eural **N**etworks
FNN	**F**eedForward **N**eural **N**etwork
GPU	**G**raphic **P**rocessing **U**nit
CPU	**C**entral **P**rocessing **U**nit
CSV	**C**omma **S**eparated **V**alue
MOM	**M**ethod **O**f **M**oments
MLE	**M**aximum **L**ikelihood **E**stimation

I dedicate this work to my mother

Chapter 1

Introduction

1.1 Motivation

More than 4 billion users access and utilize the internet around the world every day and the number of Internet users increases by more than 7% annually. The Internet revolutionized our world since the last century as now billions of people can interact with one another with simple clicks on their mobile phones. Additionally, The Internet is making information and education available to everyone on the planet, hence creating massive opportunities for advancing knowledge across the spectrum of human endeavors at a fast pace. Moreover, the Internet has contributed enormously to the development of countless devices, applications and services and has completely transformed the world of transportation. The Internet is an essential foundation for business development, entertainments and social interactions and has become integral to our day-to-day lives and indispensable to the operation of all critical sectors of our society.

The Internet is often described as a network of networks. A network can be as small as two computers connected together to having millions of devices interacting and sharing information and data. In fact, the Internet as we know it today is comprised of 65,536 autonomous systems (ASes) [204], which are run by entities such as Internet service providers (ISPs), content providers, or public institutions. Each of these ASes include thousands of smaller networks that implement several routing policies and communication protocols that aim to connect participating end-devices and rule out impractical or uneconomical communication paths as well as regulate all communication exchanges.

Initially, the Internet as we know it today has come along way in the last half century from its original purpose of creating a communication line between two destinations [166, 68, 90]. Historically, the first small network example is the telex messaging created in the 1930s as a way to distribute military text messages during the world war. Another quotable example is the network pioneer *SAGE Air Defense System* (Semi-Automatic Ground Environment) created in the early 1960s in North America (NA) for the purpose of gathering data from many radar sites and sharing it with counter-attack air-crafts in real time to further improve the North America's defense capability in response to any Soviet air attack. The SAGE was based on a network of direct immediate connections between several machines which resulted in cluttered cabling and a difficulty to achieve scalability. However, released in late October 1969, the ARPAnet (Advanced Research Projects Agency Network) [114, 131], developed by the Advanced Research Projects Agency (ARPA) of the United States Department of Defense [20], is the first large-scale connected computer network that implemented the TCP/IP protocol suite which later became the Internet.

Even though it relied on phone lines, the ARPAnet still revolutionized communications because it introduced packet-switching [189] as a replacement to direct connections. Packet-switching consists of a switching system to which machines connect to in order to avoid the cluttered wiring of direct connections. Communicated data through a switch is regularized by a set of communication protocol that follows the TCP/IP architecture so that, exchanged packets are encapsulated with protocol specific headers and trailers containing the source and destination addresses that allow for easier routing from one device to another. These addresses inform the switch of where to send the packet. Once at the destination, the headers and trailers are removed to reveal the transmitted data. This way, the information will reach its intended destination without the need for a single dedicated circuit between each pair of device to create an end-to-end connection. ARPAnet has initially been used for to connect time-sharing computer at mainly government-supported research sites and universities in the United States (US). However, it soon became an important research topic for the computer science and research community all over the world.

The invention of ground-based packet radio systems in the 1970s such as the Packet Radio Network Protocols *PRNET* [99] by the Defense Advanced Research Projects Agency (DARPA) [51], advanced the capability of ARPANET communications to incorporate mobile access of computer resources in the US. Furthermore, the creation of packet satellite networks [165] paved the way for across continent communication between the united states and some European countries. An example of satellite networks include the Atlantic Packet Satellite Network *SATNET* [192] that connected the united states-based ARPAnet to other countries. Later on, to connect end users from different countries, the need for linking the packet satellite networks and the packet radio networks with other networks became vital for a wider range of exposure. We refer the reader to [191] for more information on packet radio and satellite networks.

In an attempt to connect various research networks in the united states to Europe, scientists investigated the standardization of network interfaces. Each standard interface must implement similar protocols and communication guidelines on both ends of the connection in order to convey and re-assemble packets accurately. This effort resulted in the Internet. And this set of protocols is referred to as Internet architectures. Popular examples of Internet architectures include the OSI model (Open Systems Interconnection) and the TCP/IP model [47] (Figure 1.1) which is the most adopted one. Note that, these architectures are established by international universities or industry wide organizations.

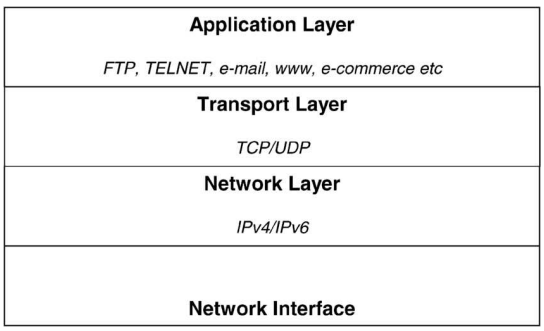

FIGURE 1.1: TCP/IP protocol suite layering diagram [71]

Figure 1.1 shows the tcp/ip architecture which is based on a 4-layer model. The

base layer (Network Access) implements a handful of protocols for hardware (e.g., network adapters and transportation medium) and software (e.g., a network device driver). Similar protocols include the IEEE 802.3 [89] and the IEEE 802.11 [88] standards. The next layer (i.e., Internet) encompasses the Internet Protocol (IP) [93]. The following layer is the transport layer and it incorporates two main protocols, that is the User Datagram Protocol (UDP) [198] and the Transmission Control Protocol (TCP) [194]. Finally, the top layer, also referred to as the application layer, represents a range of application specific protocols such as the Hypertext Transfer Protocol (HTTP) [86], File Transfer Protocol (FTP) [58], Simple Mail Transfer Protocol (SMTP) [185], Domain Name System (DNS) [53], etc.

Note that, today a structured group of several individuals referred to as the Internet Engineering Task Force (IETF) [92] collaborate in the development process for Internet standards (i.e., architectures and protocols) which are maintained by other nonprofit organizations such as the Internet Society (IS) [14], and the Internet Corporation for Assigned Names and Numbers (ICANN) [91]. The latter's main focus is on Internet domain names and numbers.

Present-day Internet and networks are being challenged by the major increase in data rates required to satisfy the emerging applications demands (e.g., streaming and data transfer) efficiently in terms of both cost and energy. This constitutes a limiting factor on how fast applications and services of the future develop as it relied on the available network speeds and quality of service. In this context, one can start questioning whether traditional network architectures should be re-examined from a fresh viewpoint and completely be replaced to meet demanding applications over heterogeneous networks at affordable costs and whether a single suite of communication protocols can perform in these networks at high efficiencies.

The answers to these questions are not so simple. In fact, the Internet is far too complex to opt for a global replacement. Instead researchers tried to tackle this problem by partitioning each network function and deciding on the right architecture for it. Nevertheless, there are still some brave attempts that opted for a global optimization of the current Internet by proposing next generation architectures and protocol suites designed for scalable, fast and reliable content delivery.

New technological advancements in network architectures and protocols are paving the way for novel Internet paradigms [145] that best suit applications and services of the future. Many ideas have participated in this evolution of the Internet, most of which are motivated by the need for improved resource utilization (i.e., computation and storage) and content distribution as well as the the rise in network virtualization and softwarization, etc. Two novel paradigms include:

- Software defined networking (SDN) [190, 57, 178] is a concept aiming to decouple the network forwarding functions from the control plane. This results in several benefits such as allowing networks to become programmable and more flexible as well as providing administrators with more visibility over their networks which helps them automate and improve their security through the use of routing intelligence inside SDN controllers and calibrate the traffic based on certain thresholds (jitter, latency, throughput, network capacity, ...) [72]. The first generation of SDN controllers began with centralized controllers (NOX classic [179], Ryu [94] and Floodlight [70, 64], etc). However they had suffered from several problems such as the single point of failure and limited capacity. Years later, another generation of distributed controllers have emerged that offered solutions to these problems. Among them two open source distributions have gained a lot of attention in the last few years due to their modularity,

multi-vendor support, high availability and cluster configuration: Open Network Operating System (ONOS) [28] and OpenDayLight (ODL) [140, 13].

- Named Data Networking (NDN) [148] is one of the five projects funded by the U.S. National Science Foundation agency (NSF) [164] under its Future Internet Architecture Program (FIA). NDN was initially created to network the world of computing devices by naming contents instead of end points as is the case for the Internet Protocol (IP). This has several advantages such as builtin multi-cast, in-network caching, multi-path forwarding and securing data directly [11]. Furthermore, NDN allows for a resiliency in communication in intermittently connected and mobile ad hoc environments which is hard to achieve seamlessly in the TCP/IP architecture. Many applications, services and protocols from the IP world have been built over this novel architecture in order to demonstrate its viability and capability in resolving the challenging problems that the contemporary Internet faces, i.e., loss of universal connectivity mobility, scalability, etc. Similar applications and protocols include a path tracing utility [108], fast packet forwarding [111], an attempt to NDNize existing IP protocols [129], an Open mHealth Application[212], video Streaming over NDN [67, 196], a routing protocol [153] and others [150],

The demand for faster performance, increased accessibility, mobility and secure communications has driven significant advancements in Internet architectures, protocols, applications and services. Whether Internet usage relates to social media or business, Internet performance and security are two of the highest order. With the immense growth of the Internet today, and the increased number of proposals for novel Internet paradigms and protocols, the need for continued monitoring, evaluation and large scale management of networks is still a priority. Therefore, researchers in both academia and industry have understood this fact and have recently begun building tools and establishing mechanisms designated for the supervision of networks.

In the next section, we will give a state of the art of the research conducted in this area. Particularly, we will be targeting performance related enhancements of Internet architectures and protocols as it is the main concern in this thesis.

1.2 State of the art

The Internet has come along way since it's invention as a research experiment. The Internet is now used to run businesses, streamline services (i.e., Netflix and youtube), communicate via voice over IP (VoIP) [69], support distributed processing and cloud computing [135]. The overwhelming success of the Internet and the complexity of networks, has led to rapid innovations in architectures, applications and network protocols.

Scientists are constantly investigating methods to secure networks against malicious attacks as well as unintentional bugs and errors and to maintain a good quality of service in network speed and performance, particularly when it concerns businesses and healthcare systems. Unfortunately, it has been observed that their attempts of monitoring networks performances and quality of service value more a culture of running code, heuristics and engineering judgements rather than a culture of sound proofs and rigorous verification methods.

In the next subsections, we first overview some of the most important metrics, typically captured and monitored when discussing network performances. Note

that, in this thesis we are particularly targeting the performance aspects of Internet architectures and communication protocols. Next, we review some related works and introduce our methodology.

1.2.1 Performance metrics

In modern days, where communication and information are the center of Information Technology (IT), network performance is of utmost importance. In fact, to run an enterprise effectively, a decent budget is allocated yearly for the provision of Internet connectivity to maintain day-to-day office tasks without unnecessary delay or down times.

Historically, network performance has been monitored by using architectural know-how and the good old-fashioned intuition. However, the accuracy obtained by these methods varies greatly and is especially true as networks grow increasingly complex. Ultimately, loss of real insights into how the network acts can lead to unforeseen performance issues that can be difficult to remedy. Nevertheless, benchmarking and simulation based testing are still considered the default option for network analysis and supervision. Whether the subject of interest is an application, a service, a protocol or a novel Internet paradigm, the fundamental motivations for network performance analysis include:

1. Observing the performance to determine if the quality of service level is satisfactory;
2. Unravelling and checking for malfunctions and performance decline;
3. Troubleshooting connectivity interruptions;
4. Assessing the viability and efficiency of newly developed applications and protocols.
5. Measuring the resources used and making appropriate energy and cost charges.

For this reason, many metrics have been created to indicate networks performance levels [74]. Some of these have been documented in standards such as Requests for Comments (RFCs) of IETF [92] and some are commonly collected by network operators based on their intuitions and heuristics. However, the ones of greatest relevance that are critical in assessing the service level of network traffic can be classified to five main categories:

1. Delay;
2. Round Trip Time;
3. Bandwidth;
4. Jitter;
5. Loss and error;

Delay

Delay, sometimes referred to as latency is a bi-directional performance metric quantified in time units to indicate any form of delay in communication that occurs over a network between a client and a server and between distant end hosts. This metric is usually assessed against a threshold beyond which the network experience worsens and the performance declines. Figure 1.2 shows examples of the latency between two hosts A and B. Note that, T_{AB} is the delay from B to A and T_{BA} is the delay from A to B.

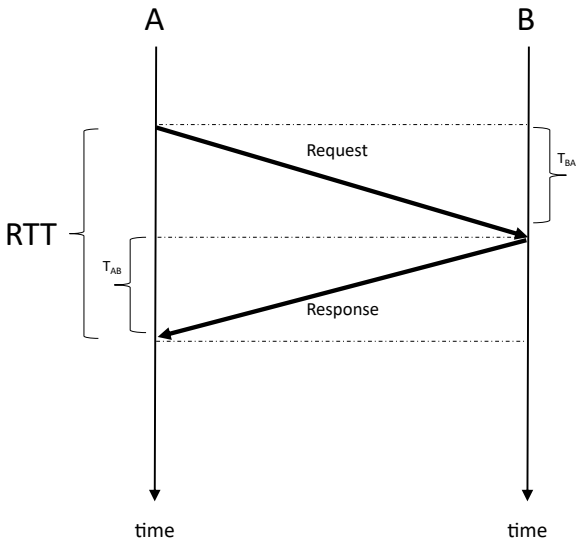

FIGURE 1.2: Delay and RTT

Round Trip Time (RTT)

Round Trip Time (RTT) is a performance metric that is quantified in time units. It is often used when assessing client server communications as it indicates the time it takes for for a server to respond to a client's request. The RTT is expected to go higher when performance worsens and lower when optimal conditions are met. Note that, RTT is more or less defined as the network delay from point A to B and back. Figure 1.2 shows how the RTT is calculated as a function of the latency, that is: $RTT = T_{BA} + T_{AB}$, where T_{AB} is the delay from B to A and T_{BA} is the delay from A to B.

Bandwidth

Bandwidth, sometimes referred to as throughput or Internet speed. This is a bidirectional performance metric that's computed as the rate of data crossing the network from point A to point B per second. In other words, the bandwidth is the amount of data that was successfully delivered over a communication channel in bits per second (bit/s or bps).

Jitter

Jitter is defined as the variation in delay for received packets causing some of them to take longer routes to travel between the same two points in a network. It is often

due to network congestion, low bandwidth and poor hardware. As a consequence, excessive Jitter leads to audio and video quality distortions such as display monitor flickering, undesired effects in audio signals and loss of transmitted data.

Loss and error

Often, data delivery encounters setback in the form of packet loss due to buffer overflow or receiving damaged packets. In similar scenarios, it is best practise to compute the rate of successfully transmitted packets. This metric is helpful to know as it can be remedied by the re-transmission of lost or distorted packets in real time in order to avoid a degraded quality of service in the network.

Over the past years, network specialists have relied on the use of automated software engineering techniques for network monitoring and performance evaluation. These techniques are often based on:

1. The use of uni-tests, and simulation based testing;
2. The use of existing application layer protocols such as Traceroute [193, 108] and Ping [87] or the development of tools centered around management protocols such as SNMP [12], [73] (SNMP is used for gathering information exchanges in networks).

In the literature, many researchers published in this field. Surveys that overview measurement related research in the context of networking are found in [18, 143]. The latter summarize different network supervision solutions, in addition to the different tools used for monitoring and analyzing network traffic. Another survey on network measurement for Software-Defined Networks (SDN) is presented in [190]. In [144], the authors propose a network performance methodology based on Quality of Experience (QeE) benchmarks from an end-user experience. Another tool that enables network performance measurement is developed in [186]. And a computer network monitoring system (CNMS) developed by the University of Waterloo Computer Communications Networks Group (CCNG) is described in details in [146]. Additionally, in [46] the authors describe their successful attempt and toolset used to supervise the performance of the Advanced Research Projects Agency (ARPA) network. Furthermore, [136] proposes a prototype implementation of network equipment availability and performance reporting. The latter uses simulations based on the SNMP protocol as well. More tools developed for the same purpose include [50] and [134]. Finally, [105] presents an approach that combined controlled experimentation and machine learning to estimate Quality of Experience (QoE) from encrypted video traces using network level measurements only.

In the context of simulation based methods and benchmarks, researchers are constantly looking to build newer software for evaluating and assessing networks performances and behaviours by either making use of existing application layer protocols or by inventing newer protocols. The latter is generally the case when it relates to assessing future Internet paradigms such as our work in [108] and [150, 54]

1.2.2 Related work

Recent advances in modern technology and network innovations have driven the desire to move away from manual error-prone methods of testing network components and evolve from the ad-hoc tools and simulation based testing that have been

traditionally used in assessing the correctness of networking components. Despite the criticism that formal verification (FV) methods have been receiving and lack of appreciation, they have achieved undeniable results and made great contributions in this field. In this section, we overview some popular FV methods and explore their applications in the context of networking.

Techniques for formal verification

Formal verification [202, 180] refers to having two inputs: a rich and rigorous model for the system that is being formally assessed and a set of properties that indicate the requirement to be satisfied by this system. Existing computer-based tools for verification typically, support at least one modeling formalism. In this thesis, we focus on the BIP framework [24] which we will discuss in the next chapter. Generally, there are many formal verification approaches that include both automated and interactive techniques. We refer the reader to [170] for a survey that examines an exhaustive list of methods and associated tools. Below we list popular methods:

1. Model Checking [126], a method developed by Clarke and Emerson [45] and by and by Queille and Sifakis [80, 79] in the 1980s. Traditional formal verification methods that came before model checking, were generally associated with logic-based axiomatic or deductive techniques for establishing proofs of correctness. In model checking, the statement $M \models \phi$ (i.e., M satisfies ϕ) is assessed by representing the system behavior as a model M with a Kripke structure or a labeled transition graph as the underlying formalism, whereas the property to assess is a formula that is generally written in temporal logic. Unfortunately, the state explosion problem limits the use of model checking to small scale systems. Nevertheless, other lighter version of model checking have been proposed to cope with this issue such as: symbolic model checking, bounded model checking, and statistical model checking. We will cover the latter (statistical model checking) in the next chapter.

2. Theorem proving [21] consists of focusing on deductive and sound proofs to verify the system behavior against a set of properties, using computer programs. The system behavior and the properties are expressed as mathematical theorems. Note that, fully automated theorem provers strive for power and efficiency, often at the expense of guaranteed soundness [1]. For this reason, interactive provers that require human-machine collaborations, are often used. Similar tools include Isabelle, Isabelle/HOL [34], Coq [211], Hol Light [75], and many more.

3. Static analysis [15, 176] is a technique that relies on analyzing software programs or configuration files without executing code. The way it works is by extracting information about the run-time behavior of the software and configuration files and assessing it prior to the deployment step to discover bugs.

4. Satisfiability solvers (SAT) [17, 180]: the main idea of traditional SAT solvers is to express the verification problem $M \models \phi$ (i.e., M satisfies ϕ?) in propositional logic which is then negated and transformed to conjunctive normal form (CNF) in such a way that ascertaining the Satisfiability of this CNF allows us to deduce the validity of $M \models \phi$ [170].

Examples of formal verification in networking

The application of formal verification (FV) in the context of networking, particularly to tackle their performance aspect is still not a priority for the networking community. FV is mainly used to assess safety critical and functional properties of a network component rather than to examine quality of service and performance related criteria. As such, the networking field still lacks rigor when compared against other mature industries. Fortunately, the community is slowly catching up to the trend of FV and realizing the need for better mechanisms for rigorous verification and assessment of networking systems. However, given the fact that formal methods and verification techniques require the use of professionals with a certain level of expertise and take a considerable amount of time and computational resources to apply, the networking industry typically relies on cheaper manpower and commercial software and tools to complete the task and assess the performance of networks. There are at least six networking domains to which formal verification has been applied:

1. **Protocols**: they are defined as a set of rules of communication between various processes and devices in a network. Generally, FV has been applied to protocols to detect weaknesses and vulnerabilities at an early stage of the design. The properties evaluated are often associated with the functional behaviour of the system and safety properties, such as:

   (a) The deadlock, i.e., when the protocol constantly waits for a condition that can never be fulfilled;

   (b) The livelock, i.e., when the protocol repeatedly executes a sequence with no progress;

   (c) The improper termination of the protocol.

   The primary FV methods used in this context are model checking and theorem provers. Some related works include [85, 138, 31]. In the case of verifying non-functional properties and establishing performance related concerns, some researchers followed the direction of statistical model checking and probabilistic model checking instead [26, 121, 112]. We refer the reader to [35] and [170] for surveys that summarized the various efforts of applying FV to networking protocols.

2. **Network Property Verification**: in this case, scientists analyze the traffic throughout in a network topology. Some of the investigated properties include: reachability, i.e., "Can a packet from node A reach node B?", and loop detection, i.e., the manipulation of packet headers by network routers and switches leading to some packets being circulated constantly over the same paths without reaching their destination. In this context, model checking, symbolic execution and theorem provers are often used in addition to many other methods.

3. **Network configuration management**: Writing wrong configuration files has several consequences on the networks and can cause costly problems. Issues like access control failure, loops, black-hole can manifest, hence causing inefficient performance by extension. In this case, scientists use static analyzers to asses and detect faulty configuration files offline prior to their execution.

4. **Network security** is a topic of utmost importance in the community. It is crucial that security properties such as secrecy and authentication are satisfied. Thus, a multitude of FV methods have been applied. A good use case

is strengthening firewall security against malicious attacks or unintentional bugs during deployment. For this category, techniques such as static analyzers, model checking and SAT solvers are often used.

5. **Formal synthesis**: There have been efforts made for synthesizing protocol implementations from high level specifications. We refer the reader to [201] and [170] for examples.

6. **Hardware verification**: Formal verification can be applied to the hardware aspect of networks (e.g., routers, switches). Existing research includes [19] [36] [37].

7. **Future Internet paradigms**: the continued success of the Internet is constantly being threatened by the various sophisticated security attacks and challenged by the lack of performance reliability of Internet services. Unfortunately, very few scientists studied the application of FV early on in the design phase, since their main concern at this stage is about providing evidence of the viability and feasibility of their proposals rather than their performances. In one of our studies, we successfully applied statistical model checking to design and build a high throughput software forwarder for a novel Internet architecture called Named Data Networking (NDN), prior to its deployment [108].

1.3 Discussion and organization

The networking industry, routinely monitors the performance indicators to establish the correctness and the robustness of applications, services and networking protocols by ensuring a minimum level of testing, since exhaustive testing is generally impossible. The absence of rigorous techniques often results in connectivity interruption and performance decline or subtle bugs that remain undetected until they manifest themselves during critical situations (e.g., life threatening and financial losses).

With the significant role that networks play in all aspects of our lives (business, entertainment, etc), it has come to attention that the correct functioning of networking components (e.g., protocols, devices, configurations) must be a top priority to move away from manual error-prone methods of testing. As such, in recent years, formal verification has been introduced to networking in various domains such as network devices (e.g., routers, switches) [103], communication and routing protocols [35, 31], network configurations and security [30], large software programs [42] and many others.

Sadly, the community is not yet motivated by the application of formal verification (FV) in the context of assessing the performance of networking components (i.e., ensuring a good QoS and the continuity of service). Very few researchers addressed this subject (e.g., [108], [23]). Indeed, given the limiting factor on how much effort FV requires (i.e., the knowledge and experience, time and computational resources), network analysts prefer to stick to trial and error methods of performance monitoring and simulation based-testing. Hence, due to the importance of this topic and recent advancements in the networking community, we embark in a research that aims to tackle the problem that relates to ensuring a good performance level and quality of service by extension, of networking software components. In particular, we target applications, protocols, services and novel Internet architectures. We propose a methodology for evaluating and assessing the performance of existing

networking software components as well as the ones that are in the design phase. Note that, in this thesis, we do not concern ourselves with the functional behavior, the safety properties nor hardware related problems since this topic has been widely explored.

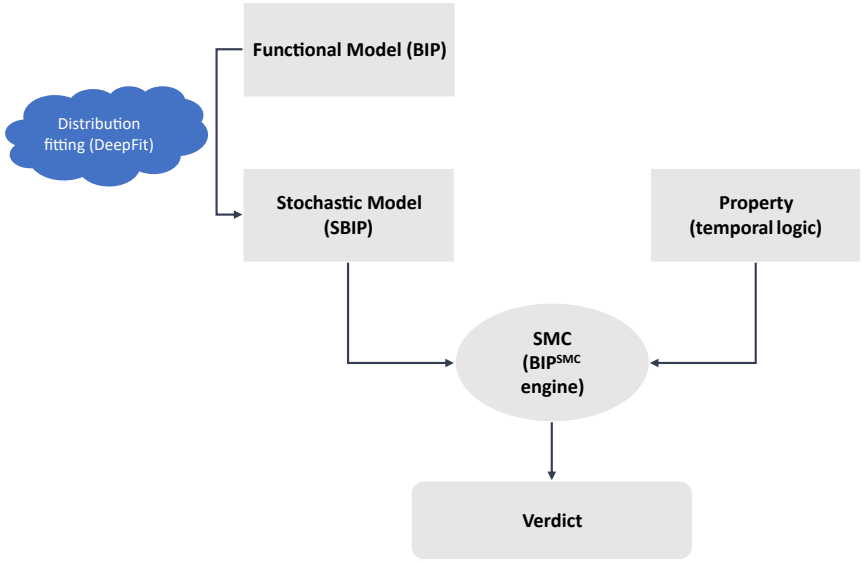

FIGURE 1.3: Proposed methodology

This thesis has two parts. The first one presents our adopted methodology which is based on a framework for modeling systems (e.g., applications, protocols, software forwarders, architectures) called BIP and a formal verification approach that relies on statistical model checking (SMC). SMC takes a stochastic model written in the BIP formalism and a property to verify. The stochastic model is generally obtained by modeling the functional behavior of a system, augmented with probability distributions (PD). The latter are, typically, obtained by collecting measurements from the system's execution and analyzed using traditional statistical tests to select the best fit distribution (i.e., distribution fitting). We demonstrate the benefits of our methodology in addition to its feasibility and ease of use via an example of a software forwarder to which we aim to maximize the throughput in chapter 2 of part 1. This chapter further emphasizes the importance of distribution fitting for the correct assessment via SMC. The second part of this thesis highlights our main contribution of automating the selection process of the 'best' fit probability distribution. We developed a tool called DeepFit that combines traditional statistical tests and machine learning for distributional modeling. DeepFit is then integrated into the workflow of our methodology as shown in figure Figure 1.3.

Part I

Methodology and application

Chapter 2

Formalisms and methodology

In this first part of the thesis, we propose a methodology for evaluating and improving the performance of any network related system. This includes protocols, applications, forwarders, etc. Our methodology targets fundamental and common network performance metrics [74, 8] (e.g., bandwidth also called throughput and latency also called delay, packet loss, jitter) which are typically analyzed by monitoring live or synthetic traffic, running tests and gathering statistics to conclude the overall performance of the system under study, from an end-user perspective. Given the increasing complexity of networks, the variety of protocols included in each communication, and the diverse intrusion attempts, networks are now evolving in an unpredictable manner and are subject to unexpected situations. Therefore, regular simulations and tests encompass a high degree of uncertainty and can't be used to conclude an accurate assessment with high confidence.

Our proposed methodology is model-based and relies on a framework that encompasses a stochastic modeling formalism to correctly and faithfully capture the behaviors of the studied component, in addition to its associated rigorous and efficient analysis techniques. This approach can be applied to existing network implementations with the aim of increasing their current performances as well as to systems that are still in their early design phases. Note that, the latter is typically considered when the goal is to create implementations that satisfy the performance requirement early on in the process. We propose a three step approach:

1. Building a rich faithful functional model of a networked system;
2. Augmenting the functional behavior with probability distributions obtained by instrumenting the system and producing a calibrated stochastic model;
3. Assessing the performance of the system using statistical model checking (SMC) which takes as input a stochastic model, a specification and a set of parameters to control the accuracy of the assessment.

In this chapter, we start off by explaining how our methodology works in section 2.1 which is based on statistical model checking (SMC). In section 2.2, we explore the underlying semantics of the modeling formalism adopted in this methodology. Here, we provide general theoretical background information on stochastic systems modeling, specifically a group of models referred to as Markov models. Next, in section 2.3 we examine the theory of SMC as well as some of its algorithms. Then, in section 2.4, we profoundly explore the requirement formalization of the specified performance (i.e., the expected performance of the studied system) which is one input to SMC. Additionally, in section 2.5, we present the stochastic modeling formalism embraced in this thesis which constitutes another input to SMC. Furthermore, we illustrate the software used in the analysis which implements the theory of SMC

and its corresponding formalisms in section 2.6. Finally, we conclude this chapter with a discussion about the advantages and/or challenges of our approach.

2.1 Methodology

As stated in the introductory chapter, often evaluating the performance of network systems relies solely on simulations and tests to generalize the results. However, the correctness of these statements is not supported by any verification technique or formal proof. In this thesis, we propose using a formal model-based approach to overcome the uncertainties of the traditional benchmarks. This is based on a framework encompassing a stochastic modeling formalism as well as the associated analysis techniques.

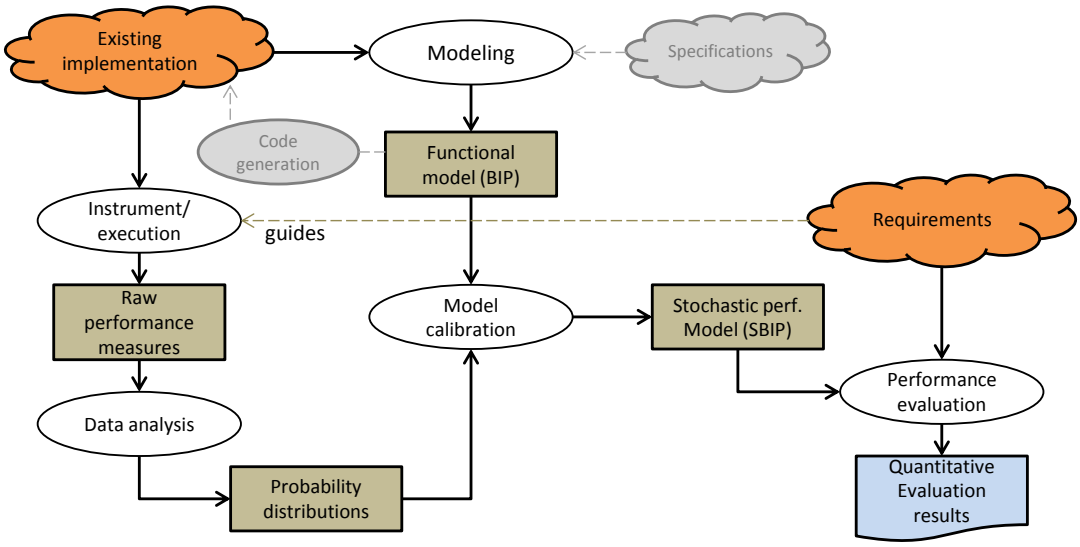

FIGURE 2.1: Performance evaluation approach

Our methodology (Figure 2.1) is based on a formal model. In order to evaluate a system's performance, its model must be faithful, i.e. it must reflect the real characteristics and behavior of the system. Moreover, to allow for exhaustive analysis, this model needs to be formally defined and the subsequent technique used in the analysis needs to be trustworthy and scalable. Our approach adheres to these principles in two ways. First, by relying on the formal framework *SBIP* (introduced later in this chapter) that encompasses a stochastic component-based modeling formalism and a Statistical Model Checking (SMC) engine for analysis [139]. Second, by providing a method for systematically building formal stochastic models for verification that combine accurate performance information with the functional behavior of the system. This approach takes a functional system model and a set of requirements to verify. The functional model could be obtained in two ways:

1. From a high level specification if the system is still in the design stage;
2. From an existing implementation, if we aim to improve its current performance.

After rigorously constructing a rich and faithful functional model, the next step is to augment the model by incorporating the performance information. This is necessary because our methodology is based on SMC which requires running simulations of the system to generalize, under certain assumptions, the partial result (obtained by simulating the system a number of times) to the whole system with a fixed confidence and a controllable accuracy The collection of performance information depends on the way the functional model was built. In fact, if using a high level specification, an additional step must be performed. That is, code generation. The execution of such code will be used to capture measurement data. Luckily, the framework we proposed incorporates a handful of tools such as a compiler which automatically generates and executes code tailored to the written model [33]. In the case where the model is obtained from an existing implementation of the system, we simply instrument it and collect the necessary performance measurements regarding the requirements of interest (e.g., latency, throughput).

Once all measurements are collected, they are analyzed and characterized in the form of probability density functions [205] with the help of statistical techniques such as sensitivity analysis and distribution fitting [169], [118]. The obtained probability density functions are then introduced in the functional model using a well defined calibration procedure [158]. The latter produces a stochastic timed model (when measurements concern time), which will be analyzed using the SMC engine.

Note that the considered models in this approach or workflow can be parameterized with respect to different aspects that we want to analyze and explore. Basically, the defined components types are designed to be instantiated in different contexts, e.g. with different probability density functions thus showing different performance behaviors. While, the model considered for analysis using SMC is a specific instance for which all the parameters are fixed, some degree of parameterization is still allowed on the verified requirements. We will demonstrate this via a real-world example in the next chapter.

In the next section, we provide a necessary theoretical background information on stochastic systems as they serve as the underlying semantics for the formalism adopted in SMC.

2.2 Stochastic Systems Modeling

Stochastic models can be grouped into various categories. However, in this thesis we specifically focus on a category referred to as Markov models which are characterized by the Markov property of being memory-less. This property indicates that the evolution of the system relies solely on the current state and not on its progression history. That is, the future behavior of the whole system is independent of its past behavior except the current state.

In the next subsection, we present two well known Markov models [141]: Discrete Time Markov Chains (DTMC) and Markov Decision Process (MDP). Note that we mainly focus on finite and discrete models. Additionally, we adopt a state-based representation, with state labels, of stochastic processes referring to sequences of random variables progressing in time.

2.2.1 Discrete Time Markov Chains

Definitions

First, we define the terminology that we use, hereafter, in this thesis:

- Let AP be a set of atomic propositions[1];
- Let the alphabet $\sum = 2^{AP}$ and let us denote the subsets of $\sum$ with symbols;
- Let $\sum_*$ (or $\sum_w$) be the sets of finite/infinite words over $\sum$ [2];
- Let $S = \{s_o..s_N\}$ be a finite set of N states (nonempty) [3];
- Let $L : S \mapsto \sum$ be a a state labeling function. This function assigns a set of true atomic propositions to each state;
- Let $\nu : S \mapsto [0,1]$ be the initial states distribution that satisfies the following condition: $\sum_{e \in S} \nu(e) = 1$ referring to the states from which the system starts evolving;
- Let $\mu : (S,S) \mapsto [0,1]$ be the state transition probability function over S. μ has to satisfy the following condition $\forall e \in S, \sum_{e\prime \in S} \mu(e,e\prime) = 1$. This condition specifies the probability to move from any state $s \in S$ to $s\prime \in S$ through a single transition. Note that for the discrete case, this probability is represented by a matrix of size N x N such that N is the number of states in S.

A DTMC M is defined as the tuple $(S, \sum, L, \nu, \mu)$.

Note that, for a given state $s \in S$, its set of predecessors is defined as all the states $s\prime$ of S such that $\mu(s\prime, s) > 0$ and its set of successors is respectively defined as all the states $s\prime$ such that $\mu(s, s\prime) > 0$. Additionally, we define the set of deterministic states $s \in S$ all states with a unique successor. That is, there exists a unique state $s\prime \in S$ such that $\mu(s, s\prime) = 1$. Observe that, a DTMC M is considered deterministic only if the following two conditions persist:

1. $\exists s_0 \in S$ such that $\nu(s_0) = 1$
2. $\forall s \in S, \forall w \in \sum$, there is at most one $s\prime \in S$ such that $\mu(s, s\prime) > 0$ and $L(s\prime) = w$

A path d **of** M is defined as a possible execution/evolution of M. That is an infinite sequence $s_0 s_1 s_2...$ such that $\nu(s_0) > 0$ and $\forall i \geq 0$, $\mu(s_i, s_{i+1}) > 0$.

A trace t associated to a path $d = s_0 s_1 s_2...$of M is the infinite words $w_0 w_1 w_2...$ such that $\forall i \geq 0$, $L(s_i) = w_i$. Generally, given a path d, the corresponding trace can be insinuated.

Let $M = (S, \sum, L, \nu, \mu)$ be a DTMC and d a path $d = s_0 s_1 s_2...$ of M, the suffix of d at the i^{th} state is a sub-path of d that starts at s_i and is referred to as $d[i..] = s_i s_{i+1}....$ Respectively, the prefix of d at the j^{th} state is a sub-path of d that starts at the initial state s_0 and ends at s_j. The latter is referred to as $d[..j] = s_0..s_j$.

[1] In this thesis we focus on finite discrete DTMC, thus finite AP
[2] In this thesis we focus on finite discrete DTMC, thus finite words
[3] In this thesis we focus on finite discrete DTMC, thus finite states

Example of DTMC

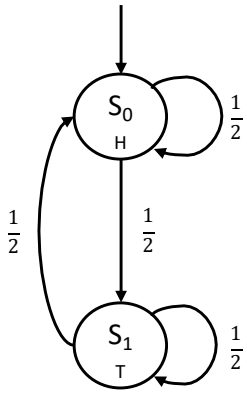

FIGURE 2.2: A DTMC of a fair coin flipping game

To illustrate the above mentioned definitions, let's consider a simple coin toss game using a fair coin. Each flip produces one of two outcomes with equal probabilities: to land on heads H or on tails T. In this way, we generate a sequence like "HTTTH..." If we continue to toss the coin. This behaviour can be depicted using a DTMC which we graphically present in Figure 2.2. We define:

- $S = \{s_0, s_1\}$
- $L(s_0) = H$ (i.e., heads), $L(s_1) = T$ (i.e., tails)
- $\nu(s_0) = 1$
- $\mu = \begin{matrix} \\ s_0 \\ s_1 \end{matrix} \begin{matrix} \begin{matrix} s_0 & s_1 \end{matrix} \\ \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix} \end{matrix}$

This example illustrates a DTMC with two states s_0 and s_1 and four transitions of equal probabilities since each state can transition back to itself. ν and μ are valid probability distributions. Note that the transition probability function μ is represented by a matrix since this is a discrete scenario.

2.2.2 Markov decision process

A Markov Decision Process (MDP) is an extension of the Markov chain in which non-determinism is present in addition to the probabilistic behavior. A MDP provides a mathematical framework for modeling decision-making situations where transitions typically depend on an action introduced via an input parameter (e.g. user input, or some environment interaction). In the next subsections, we formally present a few MDP related definitions and an example to illustrate their behavior. Note that, unlike DTMCs, MDPs are assumed to have labels on both states and transitions.

Definitions

An MDP is a tuple $(S, L, Act, \sum, \nu, \mu)$ such that: [4]

[4] We assume a finite number of states and actions

- $S = \{s_o ... s_N\}$ is a finite non-empty set of states;
- AP is a set of atomic propositions and the alphabet $\sum = 2^{AP}$;
- $L : S \mapsto \sum$ is a state labeling function. This function assigns a set of true atomic propositions per state;
- Act is a finite set of action labels and $Act(s)$ is the set of enabled actions at state $s \in S$;
- $\nu : S \mapsto [0,1]$ is the initial states distribution which satisfies the following condition: $\sum_{s \in S} \nu(s) = 1$;
- $\mu : (S, Act, S) \mapsto [0,1]$ is the transition probability function over S which satisfies the following condition $\forall s, s\prime \in S$ and $\forall a \in Act$, $\sum_{s\prime \in S} \mu(s, a, s\prime) \in \{0, 1\}$.

An action $a \in Act$ is enabled in $s \in S$ only if $\sum_{s\prime \in S} \mu(s, a, s\prime) = 1$. Note that, for any $s \in S$, $Act(s)$ should not be empty. Additionally, we define two more sets as follow:

- $\forall s \in S$, the set of successors of s with action $a \in Act$ as $Successors_a(s) = \{s\prime \in S, \text{s.t } \mu(s, a, s\prime) > 0\}$;
- The set of states with a single deterministic transition is: $Det_M(S) = \{s \in S, \text{s.t, } \exists s\prime \in Successors_a(s), \text{s.t., } \mu(s, a, s\prime) = 1\}$.

An MDP, the process starts from a state s_0 determined probabilistically using the initial states distribution ν. An action a can be, non-deterministically, selected from the set of enabled actions at s_0 (i.e., $a \in Act(s_0)$). Then the system can transition to a new state $s\prime$ probabilistically via the transition probability function μ. Note that, it is possible to have more than one initial distribution in a general MDP model. However, in this thesis we focus on having a single ν.

A path d **of** M is defined as an infinite sequence $s_0 a_1 s_1 a_2 ...$ of states and actions, such that $\nu(s_0) > 0$ and $\forall i \geq 0$, $\mu(s_i, a_{i+1}, s_{i+1}) > 0$.

A trace t associated to a path $d = s_0 a_1 s_1 a_2 ...$of M is the infinite word $W = w_0 a_1 w_1 a_2 w_2 ...$ such that $\forall i \geq 0$, $L(s_i) = w_i$. Note that given a path d, the corresponding trace can be obtained.

Example MDP

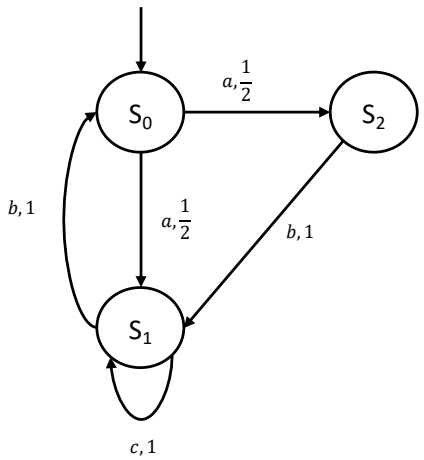

FIGURE 2.3: A MDP example

Consider the MDP example in Figure 2.3. We define:

- $S = \{s_0, s_1, s_2\}$ the set of states
- $Act = a, b, c$, the set of actions
- A unique initial state s_0 such that $\nu(s_0) = 1$
- The sets of enabled actions per states are: $Act(s_0) = \{a\}$, $Act(s_1) = \{b, c\}$, $Act(s_2) = \{b\}$
- The transition probabilities are:
  $\mu(s_0, a, s_1) = 1/2$
  $\mu(s_0, a, s_2) = 1/2$
  $\mu(s_1, b, s_0) = 1$
  $\mu(s_1, c, s_1) = 1$
  $\mu(s_2, b, s_1) = 1$

In this example, we notice the non-determinism present between actions c and b at the level of state s_1.

2.3 Statistical Model Checking

2.3.1 Background

We recall that in this thesis, the methodology we propose for assessing the performance of networked systems is a model-driven approach in which we rely on model checking [126] for the verification of performance related requirements. The latter suggests the need for quantitative methods to analyze the stochastic behavior. There are several works in the literature that tried to investigate the probabilistic setting as it relates to the traditional model checking [199, 175, 44, 98, 83, 40, 22, 168, 48]. This resulted in the development of many mature tools [120, 101, 39] that have been used in diverse application domains such as: industrial process control [117], avionics communication protocols [27], cloud computing [113], power management [154], automotive [16], security [183], biology [78], network data plane [110, 112] and many more.

Given a stochastic model M and a requirement ϕ, probabilistic model checking typically answers the following question: *what is the probability that M satisfies ϕ?* One way to solve this problem is to try and conduct an exhaustive state space exploration of the system via numerical techniques (i.e., solving optimization problems). This is very efficient and often produces accurate results. Nevertheless, when confronted with real-world applications with large state spaces, these techniques are not scalable. However, there is a wide range of ongoing research that try to mitigate this concern via techniques such as symmetry reduction [119], abstraction [167, 102], multi-terminal binary decision diagrams [65], etc. The second way is based on Statistical Model Checking (SMC) and it is the method embraced in this thesis (next subsection).

2.3.2 SMC in a nutshell

Statistical model checking [81, 207] is a formal verification method that combines simulations with statistical reasoning to provide answers on whether a stochastic

system, under certain assumptions, satisfies some requirements with a fixed confidence. It emerged as a solution to the state space explosion problem associated with using the classical model checking on probabilistic models. SMC consists of only exploring a sub-part of the state space and mainly relying on statistics and simulations to generalize, under certain assumptions, the partial results that are obtained by simulating the system a number of times to the whole system with a fixed confidence and a controllable accuracy.

Given a stochastic model and a formal property, SMC can answer the two following questions:

1. **Qualitative:** Can the model satisfy the specified requirement with a probability p such that $p \geq \theta$, respectively $p \leq \theta$, where θ is a certain threshold?
2. **Quantitative:** What is the probability that the model satisfies the requirement specification?

There are two techniques that were proposed in the literature for statistical model checking to answer these two questions. On one hand, one algorithm provides a qualitative answer to the first question by positioning the probability with respect to a given threshold without computing it. This approach has been proposed by Younes and Simmons [209, 210] and is referred to as **hypothesis testing**. On the other hand, the algorithm proposed by Lassaigne and Peyronnet [124] provides a quantitative answer to the second question by computing the actual probability for a model to satisfy the required property. The latter is referred to as **probability estimation**.

Formally, SMC relies on simulations and statistics in order to evaluate a Markov Chain M (previous section) against a specified requirement ϕ written in some temporal logic variant (typically LTL). Depending on the choice of algorithm selected, SMC can provide a qualitative or a quantitative answer to $M \models \phi$. Each simulation of the modeled system is individually evaluated against ϕ. For this reason, a simulation can be represented as a discrete random variable B_i with a Bernoulli distribution of parameter p where i is the simulation index. This variable has two values when $B_i = 1$ (i.e., the simulation satisfies the requirement ϕ) such that $P(B_i = 1) = p$ or $B_i = 0$, i.e., the simulation doesn't satisfy the requirement, such that $P(B_i = 0) = 1 - p$. The goal of the SMC algorithm is to generate N simulations of the system and exploit the Bernoulli outcome to extract the global confidence on the system, given some conditions.

In the next subsections, we overview two of the algorithms used in the early stages of SMC to answer both the quantitative and the qualitative questions.

2.3.3 Qualitative analysis of SMC

Initial works [210, 181] proposed to answer the qualitative question are based on hypothesis testing. Let M be a Markov chain model of a stochastic system, ϕ be the specification to verify and p is the probability that M satisfies ϕ (i.e $p = P(M \models \phi)$. In order to check if the probability p is greater or equal to a certain threshold θ (i.e $p \geq \theta$)[5], two hypothesis are tested against each other. That is:

- Hypothesis H: $p \geq \theta$
- Hypothesis K: $p < \theta$

[5]The same analysis can be applied to $p \leq \theta$.

Typically, a traditional test-based solution cannot promise an accurate assessment of the model and can't guarantee that H and/or K is certainly true. However, it is possible to bound the probability of making an error by controlling the confidence results of the test via two parameters α and β. The pair (α, β) constitute the strength of the test and are defined such that the probability of accepting K (respectively, H) when H (respectively, K) holds, called a Type-I error (respectively, a Type-II error), is less or equal to α (respectively, β). However, it is impossible to ensure a low probability for both types of errors at the same time [200, 209]. A possible solution [156] is to relax the test by considering an indifference region $[p_1, p_0]$ with $p_1 \leq p_0$ and $\theta \in [p_1, p_0]$ such that the new thresholds p_0 and p_1 are typically defined in terms of the single threshold δ. That is, $p_0 = \theta + \delta$ and $p_1 = \theta - \delta$. We denote the size of the indifference region as $t = p_0 - p_1$. The solution consists of testing two new hypotheses with regards to the indifference region as opposed to the two previously stated hypothesises H and K. The new hypothesis are:

- $H_0 : p \geq p_0$
- $H_1 : p \leq p_1$

Note that if p is between p_1 and p_0, we say that the probability is sufficiently close to θ so that we are indifferent with respect to which of the two hypotheses K or H is accepted. Younes in [208] proposed two hypothesis testing algorithms. Notably **Sequential Probability Ratio Test (SPRT)** and **Single Sampling Plan (SSP)**.

Single Sampling Plan

In this algorithm [187], to test H_0 against H_1, a constant c must be specified in advance such that if $\sum_{i=1}^{n} b_i > c$ [6], then H_1 is accepted, otherwise H_0 is accepted. The pair (n, c) is referred to as the single sampling plan (SSP), hence the name of the algorithm. The main concern of this approach is the ability to find the necessary number of simulations n and the constant c such that the two error bounds α and β are respected.

Typically, one tries to work with the smallest value of n possible to minimize the number of simulations performed. In [210], Younes proposes a binary search based algorithm that, given p_0, p_1, α, β, computes an approximation of the minimal value for c and n. Note that, when the threshold θ has to compare to 0 or 1, it is better to use this algorithm.

Sequential Probability Ratio Test

In Sequential Probability Ratio Test (SPRT), two values A and B are selected such that $A > B$, to ensure that the strength of the test is respected. Let m be the number of simulations observed of the system under study. The test consists on computing the following quotient:

$$\frac{p_{1m}}{p_{0m}} = \prod_{i=1}^{m} \frac{Pr(B_i = b_i | p = p_1)}{Pr(B_i = b_i | p = p_0)} = \frac{p_1^{d_m}(1-p_1)^{m-d_m}}{p_0^{d_m}(1-p_0)^{m-d_m}}$$

[6] b_i denotes the outcome of B_i, the Bernoulli variable for each simulation of the system: b_i, is 1 if the simulation satisfies ϕ and 0 otherwise.

where m is the number of observations that have been made so far, b_i is the value of the Bernoulli variable for simulation index i and $d_m = \sum_{i=1}^{m} b_i$. The idea behind the test is to accept:

- H_0 if $\frac{p_{1m}}{p_{0m}} \geq A$;
- H_1 if $\frac{p_{1m}}{p_{0m}} \leq B$.

SPRT computes the quotient $\frac{p_{1m}}{p_{0m}}$ for a successive number of simulations (m) until H_0 or H_1 is satisfied. This algorithm terminates with probability 1. This has the advantage of minimizing the number of simulations. In [209], Younes proposed a logarithmic based algorithm SPRT that given p_0, p_1, α and β implements the sequential ratio testing procedure.

Note that so far in this thesis, we demonstrate the hypothesis testing method by considering the following specification: $P(M \models \phi) \geq \theta$. However, the same analysis can be applied to the opposite specification $P(M \models \phi) \leq \theta$.

2.3.4 Quantitative analysis of SMC

Peyronnet et al [82] and [123] propose a probability estimation procedure called **PESTIM** to quantify the probability p that a Markov chain model M meets the requirement ϕ (i.e., $p = Pr(M \models \phi)$. Their solution relies on the Chernoff bound [84] to compute the number of simulations needed to estimate p. The idea is to compute an approximation $p\prime$ of p, given a precision δ, such that

$$|p - p\prime| \leq \delta$$

with a confidence parameter α. That is:

$$P(|p - p\prime| \leq \delta) \geq 1 - \alpha$$

Formally, let $B_1, B_2, ..., B_n$ be n Bernoulli variables of parameter p corresponding to n simulations of the system. Each variable can only have two values: 1 if the simulation satisfies ϕ and 0 if it doesn't. Following Peyronnet's procedure, if we set the approximation $p\prime$ to be:

$$p\prime = \frac{\sum_{i=1}^{n} b_i}{n}$$

Then, the Chernoff-Hoeffding bound gives:

$$P(|p - p\prime| > \delta) < 2e^{-\frac{n\delta^2}{4}}$$

Note that, if n is selected in a way that the condition below hold:

$$n > \frac{4}{\delta^2} log(\frac{2}{\alpha})$$

Then, it is guaranteed that: $P(|p - p\prime| \leq \delta) \geq 1 - \alpha$. Additionally, if $p\prime$ for a number of simulations n returns $p\prime \geq \theta - \delta$, then, it is certain to achieve $P(M \models \phi) \geq \theta$ with the confidence α.

2.4 Requirement formalization

We recall that SMC takes as input a stochastic model M and a formal specification ϕ written in a variant of Linear Temporal Logic (LTL). This section explores the formalism used to express the latter.

2.4.1 Linear Temporal Logic

A Linear-time Temporal Logic (LTL) formula is constructed using:

- A finite set of atomic propositions AP;
- Basic logical operators (e.g., negation $\neg$, conjunction $\wedge$);
- Basic temporal operators (e.g., Next N, Until U);

Note that, there are additional logic operators other than the negation $\neg$, conjunction $\wedge$ such as the disjunction ($\oplus$), the implication ($\Rightarrow$) and the equivalence ($\equiv$) operators. Equivalently, additional temporal operators can be inferred using the two basic operators Next N, Until U. This includes: G (always) and F (eventually). G and F are defined as follow:

$$F\phi \equiv true U \phi$$

$$G\phi \equiv \neg F \neg \phi$$

Moreover, by combining G and F, more temporal operators are obtained (e.g., $GF\phi$ infinitely often, $FG\phi$ eventually forever).

Given a finite set of atomic propositions AP, we define the alphabet $\sum = 2^{AP}$ and all elements of $\sum$ as symbols. Let $\sum *$ be the set of infinite words over $\sum$. LTL formulas are interpreted on infinite traces $\sigma = \sigma_0\sigma_1... \in \sum *$ as follows:

- $\sigma \models true$;
- $\sigma \models p$ if $p \in \sigma_0$;
- $\sigma \models \neg\phi$ if $\sigma \not\models \phi$
- $\sigma \models \phi_1 \wedge \phi_2$ if $\sigma \models \phi_1$ and $\sigma \models \phi_2$
- $\sigma \models N\phi$ if $\sigma[1..] \models \phi$
- $\sigma \models \phi_1 U \phi_2$ if $\exists k \geq 0$ s.t. $\sigma[k..] \models \phi_2$ and $\forall j \in [0, k[, \sigma[j..] \models \phi_1$

2.4.2 Bounded LTL: BLTL

In the context of SMC, it is necessary to run a number of simulations of the system in order to generalize the partial results. Therefore, it is important to install a mechanism that allows the termination of each simulation within a finite time. For this, the formalism to be used to express the specification must be bounded (the bounds will be introduced in the next paragraphs). Thus instead of LTL, Bounded LTL (BLTL) is better suited for writing the requirements.

BLTL is a variant of LTL with bounded temporal operators used to enhance LTL operators with bounds expressed in steps or time increments [32]. These bounds generally provide the length of the run on which the BLTL requirement must hold.

The difference between the classical LTL and the bounded LTL is the use of the additional temporal operators $\geq$ and $\leq$ to express the bounds in terms of time or as a number of steps.

In bounded LTL [32], a finite sequence of states in the system is considered while LTL formulas specify the infinite behaviour of the system. BLTL binds the temporal operators (F, G, N, U) by a temporal bound. This bound may be either a number of steps using the syntax $\leq$ *#Numerical*, respectively $\geq$ *#Numerical*, or a real-time bound using the syntax $\leq$ *#Numerical*, respectively $\geq$ *#Numerical*. If the model M is untimed, the two syntaxes are equivalent.

In terms of the temporal operators, this translates to using the new bounded variants U^i, N^i, F^i and G^i instead of U, N, F and G. Formally, given a set of atomic propositions AP, BLTL's grammar is:

$$\begin{aligned}\phi :=& true\\ &|a \in AP\\ &|\neg\phi\\ &|\phi_1 \wedge \phi_2\\ &|N^i\phi\\ &|\phi_1 U^i \phi_2\end{aligned}$$

Similarly, additional logical operators can be used. That is, the disjunction ($\oplus$), the implication ($\Rightarrow$) and the equivalence ($\equiv$). In the same way, G^i and F^i can be inferred as follow:

$$F^i\phi \equiv true U^i \phi$$

$$G^i\phi \equiv \neg F^i \neg\phi$$

As stated previously, BLTL formulas are used to bounded simulations and express a finite behaviour of the system. However, in order to allow expressing probabilistic queries, a probabilistic variant of BLTL is usually used instead [156]. This refers to PBLTL (Probabilistic Bounded Linear Temporal Logic) and it is typically the formalism used in SMC to specify the requirements. We can define PBLTL formulas as BLTL encapsulated within a probabilistic operator $P[\phi]$. We write the P operator in two ways according to desired requirement. If the requirement is qualitative, we use $P_{\leq\theta}[\phi]$ or $P_{\geq\theta}[\phi]$. Otherwise, we use $P =_? [\phi]$.

2.5 Stochastic Component-based Modeling

So far in this chapter, we introduced our methodology that aims to formally assess and evaluate the performance of networked systems and networks overall. We explained that our approach is model-based and relies on statistical model checking for quantitative analysis of stochastic systems. In section 2.3, we established that SMC takes as input a model and a requirement to verify. Then, it provides insights on the probability for the model to satisfy the requirement, with a controllable accuracy, using a combination of simulations and statistics. In section 2.4, we described the formalism used to express the requirements, i.e., PBLTL.

In this section we examine the modeling formalism adopted in our methodology, i.e., BIP (Behavior, Interaction and Priority). BIP allows to faithfully express and model the system's functional behavior. However, to incorporate the ability to

make performance related decisions in networks (e.g., latency, throughput), we explore an extension of BIP called SBIP (Stochastic BIP) that is used to augment the functional model with probability settings. SBIP will be explained in the subsequent subsections.

2.5.1 BIP formalism

BIP (Behavior, Interaction, Priority) is a highly expressive component based framework for rigorous system design [24]. It allows the construction of complex, hierarchically structured models from atomic components characterized by their behavior and their interfaces. Such components are Labeled Transition System (LTS) [122] enriched with variables. Transitions are used to move from a source to a destination location. Each time a transition is taken, component variables may be assigned new values, computed by user-defined C/C++ functions. Composition of BIP components is expressed by layered application of composition operators (i.e., interactions and priorities) also referred to as the *glue*. Interactions express synchronization constraints between actions of the composed components while priorities are used to filter among possible interactions e.g. to express scheduling policies.

As denoted above, the behavior of a BIP atomic component is an LTS which consist of a set of states and a set of transitions between them with one state being the initial state. Transitions are labeled by a set of actions. Formally, an LTS is defined as the tuple $(S, S_0, Act, Tran)$ such that:

- S is a set of states;
- S_0 is a set of initial states such that $S_0 \subseteq S$;
- Act is a set of action labels ;
- $Tran \subseteq S$ x Act x S is the set of labeled transitions.

Atomic components

Atomic components are the basic elements of BIP models. Their underlying behavior is an LTS with variables to store local data. Ports are action labels that can be associated with data and used for interactions (i.e., synchronization) with other components. States denote control locations at which components wait for synchronization and transitions are execution steps between states to which action labels and guards are associated. A guard is a Boolean condition that must hold to execute the transition. In BIP, data and their related computations are written in C/C++, whereas the LTS behavior is expressed in the BIP language (we refer the reader to [195] for a detailed tutorial on BIP).

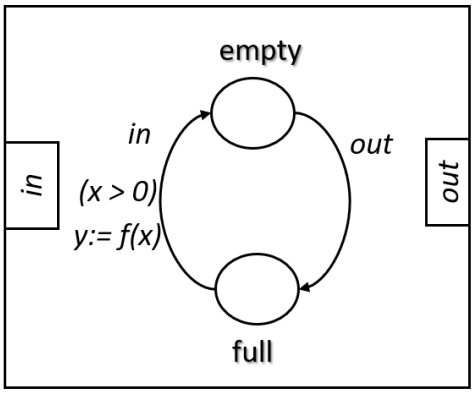

FIGURE 2.4: A BIP atomic component

To sum up, atomic components can be defined as the tuple (S, S_0, P, T, V) consisting of:

- A set of communication ports $P = \{p_1 ... p_n\}$, representing action labels, are used for synchronization with other components;
- A set of control states $S = \{s_1 ... s_k\}$ in which the components await for external interactions with other components;
- A set of deterministic variables $V = \{v_1 ... v_h\}$ to store local data;
- A set of transitions T representing a move from a control state $s_i \in S$ to a control state $s_j \in S$ that is enabled if a guard is true allowing some interaction if a port $p_m \in P$ is offered. Note that, guards refer to a boolean condition over V. Executing a transition starts with verifying if a guard condition $g \in Bool(v)$ [7] is satisfied and then is synchronized with one or many components via a port p (if p is associated to this transition) with possible data exchange. Finally, an internal computation specified by a function update $f \in Func(V)$[8] can take place (i.e., if v is a valuation of V after the interaction, then f(v) is the new valuation when the transition is completed).
- S_0 is the initial state set [9].

Figure 2.4 shows a simple atomic component (based on [25]) with two control states $S = \{empty, full\}$, two ports $P = \{in, out\}$ and two variables $V = \{x, y\}$. At control state $empty$, the transition labeled in is executable if:

1) a synchronization through in takes place which eventually updates the value of the variable x; 2) and if the guard condition $x < 0$ holds.

The transition in ends with updating the value of y by executing the function f. At control state $full$, the transition labeled out can occur with an insinuated true guard and no required internal computation.

In BIP, internal computations, variable updates and Boolean condition evaluations are done in C/C+. The BIP syntax of the atomic component in Figure 2.4 is:

```
Atom type atomExample
port in, out
```

[7] Given a set of variables V, $Bool(V)$ is the set of boolean conditions over V

[8] Given a set of variables V, $Func(v)$ is the set of functions over V

[9] In this thesis, we consider models with a unique starting state

```
data x, y
place empty, full
    initial to empty
    on in from empty to full provided (x > 0) do {y = f(x);}
    on out from full to empty

end
```

This short BIP description consists of a declaration of the atomic component *atomExample*, its two ports and two variables, followed by the definition of its behavior. A tutorial for writing BIP components can be found at [195]. Note that, for a given control state, the BIP semantics forbid using the same port to enable outgoing transitions.

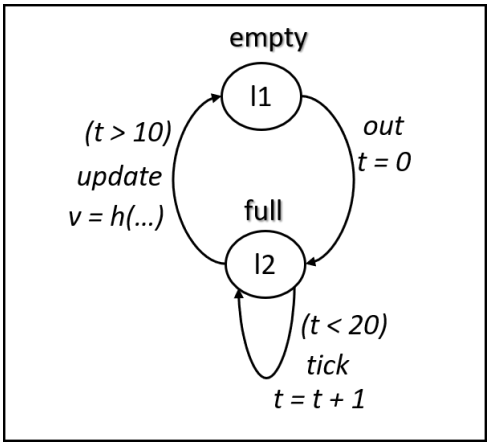

FIGURE 2.5: Non-determinism at the level of the atomic component [156]

In [156], an example of non-determinism at the atomic level is given, which occurs when several transitions are enabled simultaneously. Figure 2.5 illustrates this. In the control state l_2, the transitions *tick* and *update* are both enabled at some point in time t because both their guards are true (i.e., when $t \geq 10$ is true, $t < 20$ is also true). In this case, BIP typically conducts a non-deterministic choice. Typically, such non-determinism can be remedied in two ways in order to force a single transition at a time:

1. Using disjoint guards;
2. Using priorities (next subsection).

In the next section we introduce the composition operators that BIP uses to *glue* the atomic components and construct complex hierarchical structures.

Composition operators

As previously noted, atomic components are the basic building blocks of a BIP model that express individual sub-behavior of a larger model. In order to build a more complex structure, atomic components are assembled together to form a hierarchical structure representing the global behavior of the system being modeled.

This requires having a composition operator that allow connecting and synchronizing the atomic components together (think of it as a *glue* operator). In BIP, there are two operators: 1) interactions; 2) and priorities.

1. **Interactions:** The first composition operator is the connector. It provides a mechanism to connect (i.e., synchronize) ports of different components towards composition. Connectors are sets of interactions and an interaction is a non-empty set of ports to be executed jointly. At each interaction, a guard and a potential update function must be verified and executed to allow data exchange across the involved ports.

   To understand connectors and interactions a little better [25], let's consider a set of ports $\{p1, p2, p3\}$ (we write $p_1|p_2|p_3$ for simplification). We consider that atomic components in a BIP model are composed using $p_1|p_2|p_3$ with at most one port from each atomic component. $p_1|p_2|p_3$ is a referred to as a connector, that is a set of ports of atomic components composed through ports p_1, p_2 and p_3. As previously stated, an interaction is a non empty subset of a connector. In this demonstrated example, with three ports from distinct components, the connector $\gamma = p_1|p_2|p_3$ has seven possible interactions that constitute a synchronization between involved ports:

   (a) p_1
   (b) p_2
   (c) p_3
   (d) $p_1|p_2$
   (e) $p_1|p_3$
   (f) $p_2|p_3$
   (g) $p_1|p_2|p_3$

   In BIP, the connectors' syntax starts with a description of the connector and its participating ports and is followed by the behavior of the minimal complete interaction list. Similarly to the atomic component's transition behavior, connectors may also have one or more specified guards associated with the described interaction. Notably, the execution of a connector may result in data sharing and a function update across participating ports, respectively atomic components. BIP characterizes two types of interactions of a connector γ with each expressing a specific mode of synchronization:

   (a) Broadcast interactions indicate a weak synchronization between involved atomic components. This translates to having a single port, respectively atomic component, initiating the interaction. From the previous example where $\gamma = p_1|p_2|p_3$, the possible broadcasts initiated by port p_1 are: 1) p_1 2) $p_1|p_2$; 3) $p_1|p_3$; 4) $p_1|p_2|p_3$.
   Consider the example in Figure 2.6 consisting of three atomic components. The connector γ is constructed using three ports each from a different atomic component. We graphically represent a **broadcast** interaction (i.e., weak synchronization) with a leaf/triangle on the initiating port (in this example it's p_1). There are four possible interactions triggered by p_1 in the connector γ, we assume, in this example, that only the maximum interaction features a guard condition over the variable x associated with the transition labeled by p_1 and two variable update y and z (y and z are

associated with the transition labeled by p_2 and p_3 respectively) using a function f. This results in data sharing across the participating atomic components. Note that, we can also define guards on the remaining interactions as well. In BIP, the syntax example for this connector is written as:

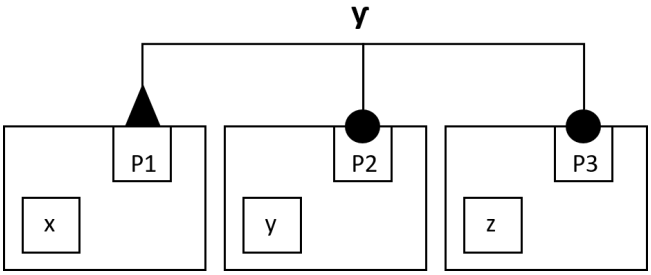

FIGURE 2.6: Example of a broadcast interaction

```
Connector γ
define p1, p2, p3
    on p1
    on p1|p2
    on p1|p3
    on p1|p2|p3 provided (p1.x > 0)
        do {p2.y = f(p1.x); p3.z = f(p1.x)}
end
```

(b) Rendezvous interactions indicate a strong synchronization between the involved ports. That is all ports of a connector are involved in the interaction. This translates to selecting the maximum interaction of a given connector. In the example above where $\gamma = p_1|p_2|p_3$, the rendezvous interaction is $p_1|p_2|p_3$.

Consider the example in Figure 2.7 consisting of three atomic components. The connector γ is constructed using three ports each from a different atomic component. We graphically represent a **rendezvous** interaction (i.e., strong synchronization) with a bullet in the end points of the connector γ. In this example, we assume that this interaction features a guard condition over the variable x associated with the transition labeled by p_1 and two variable update y and z (y and z are associated with the transitions labeled by p_2 and p_3 respectively) using a function f. This results in data sharing across the participating atomic components. In BIP, we write it as follow:

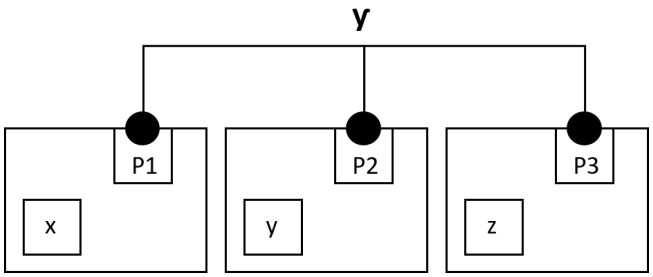

FIGURE 2.7: Example of a rendezvous interaction

```
Connector γ
define p1 , p2 , p3
    on p1|p2|p3 provided (p1.x > 0)
        do {p2.y = f(p1.x); p3.z = f(p1.x)}
end
```

Note that, when an interaction is active, the involved transitions labeled by the participating ports in the interaction are also enabled, provided that their guarding conditions hold. This is referred to as synchronization.

Previously we demonstrated that at the level of atomic components, non-determinism can manifest in having several transitions enabled simultaneously. We explained that BIP conducts a non-deterministic choice which can be remedied by having disjoint guards or by implementing policies (i.e., **priorities** are explained in the rest of this section). Similarly, at the level of interactions, the non-determinism can be encountered when several interactions are enabled simultaneously (see [156] for more examples and details about this observation). This can also be resolved by using **priorities**.

It is worth mentioning that, assembling deterministic atomic components (respectively non-deterministic atomic components) does not guarantee a deterministic compound component (respectively a non-deterministic compound component).

2. **Priorities** are the second form of BIP's composition operators. A priority is basically a scheduling policy to coordinate the execution of simultaneously enabled interactions. Each priority is a rule consisting of an ordered pair of interactions associated with a condition. Priorities can be established at the atomic component level to eliminate the non-determinism when several transitions are enabled simultaneously in a specific state and can also be specified at the level of the compound component (i.e., a component composed by assembling several atomic components and their interactions) to enforce an order of execution of internal interactions (We refer the reader to [195] for BIP implementation details and tutorials). Note that, guards can be declared over priorities as well. The syntax for a priority in BIP follows the structure below:

```
priority name
    I_a < I_b provided guard
```

In the syntax above, we set a policy to order two interactions I_a and I_b and allow I_b to be executed prior to I_a if the guarding condition *guard* is true. In BIP, the guard is a boolean expression over the variables declared in the components involved in I_a and I_b. Note that guards are written in the C/C++. When the boolean condition holds, and both I_a and I_b are enabled, the higher priority interaction is selected (I_b in this case).

Compound component example

We can construct new components called **compound components** by assembling several atomic components together via the composition operators (i.e., interactions and priorities). Note that atomic components and connectors are recyclable, that is, they could be re-used several times in a BIP model to construct larger and hierarchical complex structures.

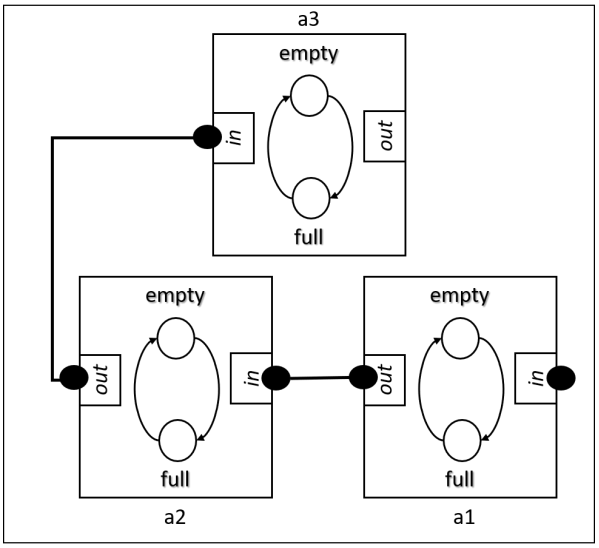

FIGURE 2.8: Example of a compound component

In this section, we demonstrate how to build a simple compound component in BIP using three atomic components a_1, a_2, a_3, rendezvous interactions and priorities. In this case, we are re-using the example in Figure 2.4 three times within the same compound component. We recall that each instance a_i of this atomic component has two control states $S = \{empty, full\}$, two ports $P = \{in, out\}$ and two variables $V = \{x, y\}$. At control state *empty*, the transition labeled *in* is executable if a synchronization through *in* takes place which eventually updates the value of the variable x and if the Boolean condition $x < 0$ is true. Eventually, *in* updates y through f. At state $full$, the transition labeled *out* can occur with an insinuated true guard and no required internal computations. To construct a compound component, we need connectors. we recall the BIP description for the atomic component below:

```
Atom type atomExample
port in, out
data x, y
place empty, full
    initial to empty
    on in from empty to full provided (x > 0) do {y = f(x);}
```

```
    on out from full to empty

end
```

Let the BIP description below be a connector *rdv* for a strong synchronization between two ports with no guards:

```
Connector type rdv
define p1, p2
    on p1|p2
end
```

Now let's assemble the compound component. The BIP syntax for this example is below. Notice the use of '*' in the priority rule to give less priority to interaction *connect1* regardless. Many other rule policies can be implemented either featuring guards or not.

```
compound type compoundExample
    component atomExample a1, a2, a3
    connector rdv connect1(a2.in, a1.out)
    connector rdv connect2(a3.in, a2.out)
    priority scheduler connect1:* < *:*
end
```

In this section, we overviewed the component-based modeling formalism BIP and its semantics and showed an example of how to compose a compound component from the basic building blocks of BIP (i.e., atomic component) in addition to the composition operators (i.e., connectors and priorities). Furthermore, we explained where the non-determinism at the level of atomic components and/or interactions can occur and how BIP resolves it using disjoint guards and scheduling policies (i.e., setting priorities). In the next section, we present an extension of BIP referred to as SBIP for the stochastic construction of models to include the probabilistic settings.

2.5.2 BIP stochastic extension: SBIP

In the previous section, we briefly examined the BIP formalism and its underlying semantics (LTS) and grammar. Additionally, we located the possible non-determinism at the level of atomic components and interactions and examined how BIP can resolve it via disjoint guards and priorities.

In this section, we introduce SBIP, a stochastic extension of the BIP formalism for the probabilistic setting. We recall that our goal is to build stochastic models to capture the performance of network systems over time and enable their quantitative analysis, via SMC. We also recall that SMC takes as input a stochastic markov chain M and a PBLTL ϕ to answer two questions about the system under study:

1. Is the probability for M to satisfy ϕ greater or equal (lower or equal) to a certain threshold θ?

2. What is the probability for M to satisfy ϕ?

The stochastic semantics of SBIP was initially introduced in [162] and extended for real-time systems in [161]. They enable the definition of stochastic components encompassing probabilistic variables updated according to user-defined probability distributions. The underlying mathematical model behind SBIP is a Discrete Time Markov Chain (DTMC).

Stochastic atomic components

The stochastic aspect at the atomic component level is introduced by the use of probabilistic variables which are variables that are updated via probability distributions [169, 205]. Formally, a stochastic atomic component is an atomic component extended with probabilistic variables and is defined as the tuple (S, S_0, P, T, V) where:

- S is a set of control states $S = \{s_1 ... s_k\}$ in which the components await for external interactions with other components;
- P is a set of communication ports $P = \{p_1 ... p_n\}$ representing action labels used for synchronization with other components;
- S_0 is the initial state;
- V is a set of deterministic $V^d = \{v_1^d ... v_h^d\}$ and non deterministic variables $V^{nd} = \{v_1^{nd} ... v_k^{nd}\}$ such that $V = V^d \bigcup V^{nd}$. Note that, each probabilistic variable $v_i^{nd} \in V^{nd}$ is associated to a probability distribution $\pi_{v^{nd}}$ over $D \rightarrow [0, 1]$ where D is a finite universal data domain. $\pi_{v^{nd}}$ is a valid probability distribution if the following condition applies:

$$\sum_{x_i \in D} \pi_{v^{nd}}(x_i) = 1$$

- T is a set of transitions representing a move from control state $s_i \in S$ to control state $s_j \in S$ and it requires that a guard $g \in Bool(v)$ holds [10] and some interaction including a port $p_m \in P$ is enabled. Upon executing a transition, an internal computation specified by a function update occurs via $f = (f^d, f^{nd})$ where f^d is a deterministic update function over V and f^{nd} is a probabilistic function.

For demonstration purposes, let's consider the simplest example [156] of a stochastic atomic component in Figure 2.9 with two states a and b, a probabilistic variable v which is updated via the probability distribution π and a single transition between the two states enabled via the port p. We graphically represent the probabilistic update of v via π by a triangle near the variable name. Given the initial value of v is set to x, when executing this transition, the variable v can take different valuations according to π, thus engendering the stochastic behavior Figure 2.10 with several possible transitions from a having the same action label p.

[10] Given a set of variables V, $Bool(V)$ is the set of boolean conditions over V

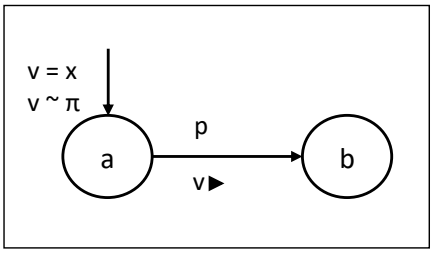

FIGURE 2.9: Example of a stochastic atomic component

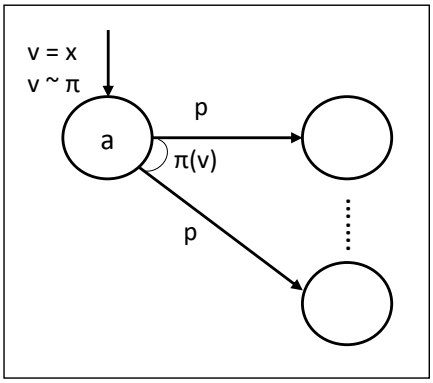

FIGURE 2.10: Stochastic behavior of the atomic component

It is worth mentioning that updating multiple probabilistic variables results in a distribution on transitions that's the product of the distributions associated to each variable. We refer the reader to [156] for some examples on this matter.

In the previous section on BIP, we stated that the non-determinism at the level of atomic components occurs when multiple transitions are enabled simultaneously in a given state, in which case, BIP conducts a non-deterministic choice unless specified otherwise by using disjoint guards and/or priorities. With SBIP, adding the stochastic element to BIP via the probabilistic variables, yields not only the non-determinism aspect but also the stochastic aspect. The semantics of a stochastic atomic component is thus a Markov Decision Process (MDP). Note that SBIP handles the non-determinism similarly to BIP.

Example: Figure 2.11 depicts a client behavior in a client-server setting where the client issues a request (**snd**) each p time units. The period p is set probabilistically by sampling a distribution function ($p \triangleright$) given as a parameter of the model. Time is introduced by explicit **tick** transitions and waiting is modeled by exclusive guards on the **tick** and **snd** transitions with respect to time (captured in this example by the variable t).

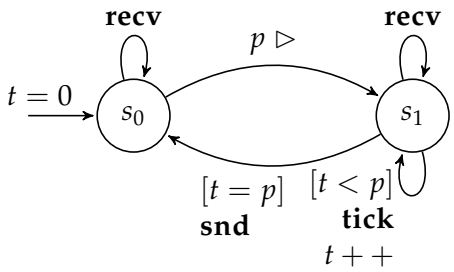

FIGURE 2.11: A stochastic BIP component; client behavior issuing requests each time unit p.

In this example, we observe that the atomic component (i.e., client-server behavior) is modeled as a classical BIP component augmented with a probabilistic variables ($p \triangleright$) which is the time it takes the client to submit a new request.

Composition of Stochastic Components

In this section, we examine the component-based composition of stochastic atomic components. Similarly to subsection 2.5.1, the composition operators of SBIP remain the same as in BIP. That is, we use the same definitions of the connectors and priorities when assembling SBIP compound components. Note that data transfer functions on interactions might also be probabilistic. However, for the sake of simplicity, in this thesis, we focus on deterministic functions only in order to have both probabilistic and deterministic variables transferred in a deterministic manner. By construction, assembling stochastic atomic components using the composition operators inherited from BIP, we produce stochastic atomic compound components with a Markov Decision Process (MDP) semantics that encompasses non-deterministic and probabilistic decisions.

In order to assess the performance of MDP models using statistical model checking, it is required to resolve any non-determinism and obtain purely stochastic behaviors where only probabilistic choices are allowed. In [156], it is proposed using probabilistic schedulers in order to produce purely stochastic Markov Chain semantics of SBIP models from a Markov Decision Process.

2.6 The BIP^{SMC} engine

In this section we present the BIP^{SMC} engine [155, 160], a tool that implements statistical model checking and the associated algorithms described in the SMC section 2.3, i.e., hypothesis testing and probability estimation. Figure 2.12 indicates that the tool takes as inputs a stochastic model description in SBIP, a PBLTL performance specification, i.e., requirement to check and a set of confidence parameters to control the accuracy of the simulations.

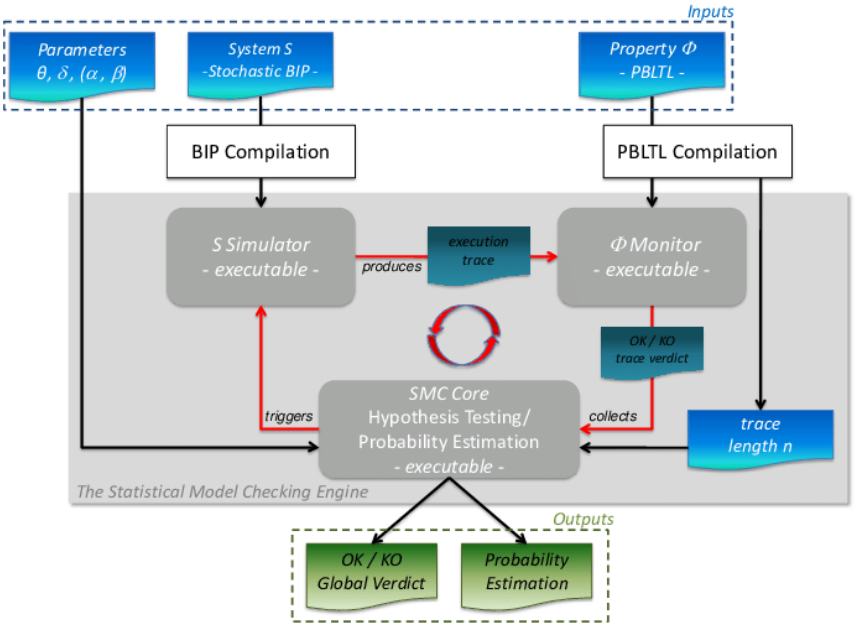

FIGURE 2.12: A Statistical Model Checking Engine for the BIP framework [160]

Initially, the tool starts by validating the entered PBLTL property through a parser module. Then with the BIP engine, the tool compiles the stochastic model previously written in SBIP to build an executable model. Next, the BIP^{SMC} engine triggers the BIP engine to generate several execution traces of the runnable model. Each trace is monitored against the PBLTL formula to produce a local verdict. Finally, after N executions and depending on the selected algorithm, the SMC core module reaches a global verdict.

2.7 Conclusion

So far in this chapter, we presented our methodology that relies on building rich stochastic performance models using the BIP formalism and its stochastic extension, then using statistical model checking to quantitatively analyze them. We examined BIP's strong modeling feature which allows the construction of complex hierarchical structures by composing reusable atomic components via the composition operators: interactions and priorities. Furthermore, BIP models can be augmented to cover the probabilistic setting with Markov Decision Processes as their underlying semantics. Additionally, we discussed how the non-determinism can be alleviated by using policies at the level of atomic components and at the level of interactions in BIP and referred the reader to [156] where the author discusses the probabilistic schedulers to produce purely stochastic components in SBIP with pure stochastic Markov chain (MC) semantics. The reason why we emphasize the use of purely stochastic MC is that our proposed methodology for evaluating the performance of networked systems, in this thesis, relies solely on statistical model checking, that we also described in this chapter, which takes three parameters as inputs:

1. A stochastic markov chain M;
2. A PBLTL formula ϕ;

3. A set of parameters to control the accuracy of the results, depending on the used algorithm (SSP, SPRT, PESTIM).

We recall that, SMC typically, answers two questions about the stochastic system under study: 1) *Is the probability for M to satisfy ϕ greater or equal (lower or equal) to a certain threshold θ?* 2) *What is the probability for M to satisfy ϕ?*

In this chapter, we also presented a tool that was designed for statistical model checking of SBIP models. This tool implements both Hypothesis testing and the parameter estimation techniques for verifying PBLTL formulas. Moreover, via examples, we showed how simple and straightforward is to model MDPs and/or DTMCs using BIP and it's stochastic extension SBIP which also inherits BIP's expressiveness. We refer the reader to the many successful case studies that modeled systems using the BIP and SBIP formalisms and successfully analyzed them with SMC. Such examples cover a multitude of domains including: (i) automotive [128],[127], (ii) avionics [23, 27] (iii) many-cores embedded systems [159], (iv) multimedia [172] (v) network data plane and protocols [110, 112]

In the next chapter, we present a real-world use case to demonstrate the viability of our methodology in the context of networked systems. The use case relates to a software forwarder (NDN-DPDK) designed for a novel internet architecture callled Named Data Networking (NDN). The idea is to apply our methodology to the forwarder's design to increase its throughput performance on a high speed fiber optic testbed (100Gbps). In this study, we use BIP to build a rich faithful model of the forwarder and we calibrate the model with the stochastic behaviour captured via SBIP. Then, we apply the SMC knowledge to parameterize the model and formally verify if NDN-DPDK is capable of reaching a maximum throughput of 100Gbps.

Chapter 3

Use case

In this chapter, we apply the methodology we proposed in the previous chapter to a novel Internet architecture, one of the five projects funded by the National Science Foundation agency (NSF) [163]. This emerging Internet architecture is called Named Data Networking (NDN).

NDN addresses some of the weaknesses of the Internet Protocol (IP). Since Internet users and applications have demonstrated an ever-increasing need for high speed packet forwarding, research groups have investigated different designs and implementations for fast NDN data plane forwarders and claimed they were capable of achieving high throughput rates. However, the correctness of these statements is not supported by any verification technique or formal proof. In this study, we propose using a formal model-based approach to overcome this issue. We consider the NDN-DPDK prototype implementation of a forwarder developed at NIST, which leverages concurrency to enhance overall quality of service. We use our approach to improve its design and to formally demonstrate that it can achieve high throughput rates.

3.1 Introduction

With the ever growing number of communicating devices, their intensive information usage and the increasingly critical security issues, research groups have recognized the limitations of the current Internet architecture based on the internet protocol (IP) [149]. Information-Centric Networking (ICN) is a new paradigm that transforms the Internet from a host-centric paradigm, as we know it today, to an end-to-end paradigm focusing on the content and it is hence more appropriate to our modern communication practices. It promises better security, mobility and scalability.

Several research projects grew out of ICN. Examples include content-centric architecture[182], Data Oriented Network Architecture [116] and many others [206], but one project stood out the most and was sponsored by the NSF called NDN [213]. NDN is gaining rapidly popularity and has even started being advertised by major networking players [38].

IP was designed to answer a different challenge, that is of creating a communication network, where packets named only communication endpoints. The NDN project proposes to generalize this setting, such that packets can name other objects, i.e. *"NDN changes the semantics of network services from delivering the packet to a given destination address to fetching data identified by a given name. The name in an NDN packet can name anything - an endpoint, a data chunk in a movie or a book, a command to turn on some lights, etc."* [213]. This simple change has deep implications in term of routers forwarding performance since data needs to be fetched from an initially unknown location.

Being a new concept, NDN (Section 3.2) has not yet undergone any formal verification. The initial phase of the project was meant to come up with a proof-of-concept prototype for the proposed architecture. This has lead to a plethora of less performing implementations in terms of packets' forwarding rates (throughput). A lot of effort was then directed to optimizing NDN forwarders' performance by trying different data structures (Hash maps) and targeting different hardware (GP-GPU). Unfortunately, validation was mainly carried using pure simulation and testing techniques.

In this work, we take a step back and try to tackle the performance problem differently. We consider the model-based approach, that we proposed in the previous chapter, which allows for rigorous reasoning and formal verification. In particular, we rely on the framework [139, 161] offering a stochastic component-based modeling formalism and Statistical Model Checking (SMC) engine which is used along an iterative and systematic design process that consists of four phases:

1. Build a parameterized functional system model, which does not include performance;
2. Run a corresponding implementation in order to collect context information and performance measurements, characterized as probability distribution functions;
3. Use these distributions to create a stochastic timed performance model;
4. Use SMC to verify that the obtained model satisfies requirements of interest.

This approach is applied to verify that the NDN Data Plane Development Kit (NDN-DPDK) (an effort to develop a high performance forwarder for NDN networks at NIST can perform at high packet forwarding rates (Section 3.4). We investigate different design alternatives regarding concurrency (number of threads), system dimensioning (queue sizes) and deployment (mapping threads to multi-cores). Using our approach, we were able to figure out what are the best design parameters to achieve higher performances (Section 3.5). These were taken into account by the NDN developers at NIST to enhance the ongoing design and implementation. To the best of our knowledge, this is the first work using formal methods in the context of the NDN project.

Note that, in the next part of this thesis, we will propose an approach for automating the distribution fitting process using deep leaning. But first, we demonstrate, the usefulness and efficiency of our proposed methodology with the NDN-DPDK use case. At the end of this chapter, we show that the design modifications we made through SMC, allowed the forwarder to achieve high throughput rates (up to 100Gbps). We published our findings in [112, 110].

3.2 Named Data Networking

This section describes the NDN protocol and introduces the NDN-DPDK forwarder being designed and implemented at NIST.

3.2.1 Overview

NDN is a new Internet architecture different from IP. Its core design is exclusively based on naming contents rather than end points (IP addresses in the case of IP) and its routing is based on name prefix lookups [95].

The protocol supports three types of packets, namely *Interest*, *Data* and *Nack*. Interests are consumer requests sent to a network and Data packets are content producers replies. The Nack lets the forwarder know of the network's inability to forward Interests further. One of NDN's advantages is its ability to cache content (Data) everywhere the Data packet propagates, making the NDN router stateful. Thus, future Interests are no longer required to fetch the content from the source, instead Data could be retrieved directly from a closer node that has a cached copy.

Packets in NDN travel throughout a network as follows: first a client application sends an Interest with a name prefix that represents the requested content. Names in NDN are hierarchical (e.g., /YouTube/Alex/video1.mpg denotes a YouTube video called Video1.mpg by a Youtuber Alex). Then, this packet is forwarded by the network nodes based on its name prefix. Finally, this Interest is satisfied with Data by the original source that produced this content or by intermediate routers that cached it due to previous requests. It is also crucial to note that consecutive transmissions of Interest packets with similar name prefix might not lead to the same path each time, but could rather be forwarded along different paths each time a request is made, depending on the forwarding strategy in place. This means that the same Data could originate from different sources (producers or caches).

The NDN forwarding daemon (NFD) [152], has three different data structures: *Pending Interest Table (PIT)*, *Content Store (CS)* and *Forwarding Interest Base (FIB)*. The packet processing, according to the NDN protocol, is as follows: [1 –]

For Interests, the forwarder, upon receiving an Interest, starts off by querying the CS for possible copies of the Data, if a CS match is found during this operation, the cached Data is returned downstream towards the client. Otherwise, an entry is created in the PIT with its source and destination faces (communication channels that the forwarder uses for packet forwarding) for record keeping. Using the PIT, the forwarder determines whether the Interest is looped in the network by checking a global unique number called Nonce in the Interest against existing previous PIT entries. If a duplicate nonce is found the Interest is dropped and a Nack of reason *Duplicate* is sent towards the requester. Otherwise, the FIB is queried for a possible next hop to forward the Interest towards an upstream node; if there is no FIB match, the Interest is immediately dropped and replied with a Nack of reason *No Route*.

For Data, the forwarder starts with by querying the PIT. If a PIT entry is found, the Data is sent to downstream nodes listed in the PIT entry, then the PIT arms a timer to signal the deletion of this entry and a copy of the Data is immediately stored in the CS for future queries. If no record is found in the PIT, the Data is considered malicious and discarded.

3.2.2 The NDN-DPDK Forwarder

NDN-DPDK is a forwarder developed at NIST to follow the NDN protocol and to leverage concurrency. In this study, we evaluate its capacity to achieve high throughput rates using Statistical Model Checking (SMC).

The NDN-DPDK forwarder's data plane has three stages: input, forwarding, and output (Figure. 3.1). Each stage is implemented as one or more threads pinned to CPU cores, allocated during initialization. **Input** threads receive packets from a Network Interface Card (NIC) through faces, decode them, and dispatch them to forwarding threads. The **forwarding** thread processes Interest, Data, or Nack packets according to the NDN protocol. **Output** threads send packets via faces then queue them for transmission on their respective NIC.

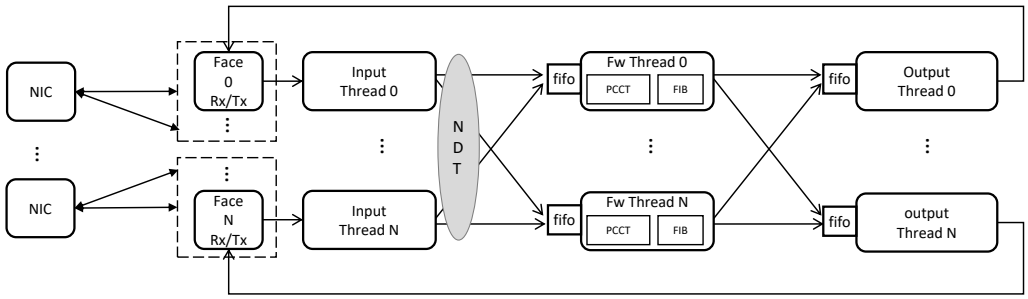

FIGURE 3.1: Diagram of the NDN-DPDK forwarder

During the forwarder's initialization, each hardware NIC is provided with a large memory pool to place incoming packets. The input thread continuously polls the NIC to obtain bursts of 64 received packets. Then decodes, reassembles fragmented packets, and drops malformed ones. Then, it dispatches each packet to the responsible forwarding thread which is determined as follows:

a) For an Interest, the input thread computes SipHash of its first two name components and queries the last 16 bits of the hash value in the Name Dispatch Table (NDT), a 65 536 entry lookup table configured by the operator, to select the forwarding thread;

b) Data and Nack carry a 1-byte field in the packet header which indicates the forwarding thread that handled the corresponding Interest. Once identified, Data (or Nack) will be dispatched to the same one.

The forwarding thread receives packets dispatched by input threads through a queue. It processes each packet according to the NDN protocol, using two data structures both implemented as hash tables:

a) The FIB records where the content might be available and which forwarding strategy is responsible for the name prefix;

b) The PIT-CS Composite Table (PCCT) records which downstream node requested a piece of content, and also serves as a content cache; it combines the PIT and CS found in a traditional NDN forwarder.

The output thread retrieves outgoing packets from forwarding threads through a queue. Packets are fragmented if necessary and queued for transmission on a NIC. The NIC driver automatically frees the memory used by packets after their transmission, making it available for newly arrived packets.

3.3 Formal Model-based Approach

In this section, we summarize the adopted methodology in this thesis, which includes the underlying modeling formalism as well as the associated analysis technique. Then we explain how we apply it to the NDN-DPDK forwarder.

3.3.1 Summary of the methodology

As we explained in the previous chapter, the methodology adopted in this thesis is a model-based approach that relies on the construction of a faithful model of a system that rigorously captures its most relevant characteristics. Then, the execution traces of the model are verified against a set of requirements via statistical model checking. Note that, our methodology requires that the model is well defined and reflects the behavior of the system that's being assessed, in order to have trustworthy analysis and high accuracy results.

This methodology embraces a specific framework which is based on the *BIP* modeling formalism and it's stochastic extension *SBIP* for stochastic component-based modeling in addition to a statistical model checking engine BIP^{SMC} [139] (Figure 3.2)

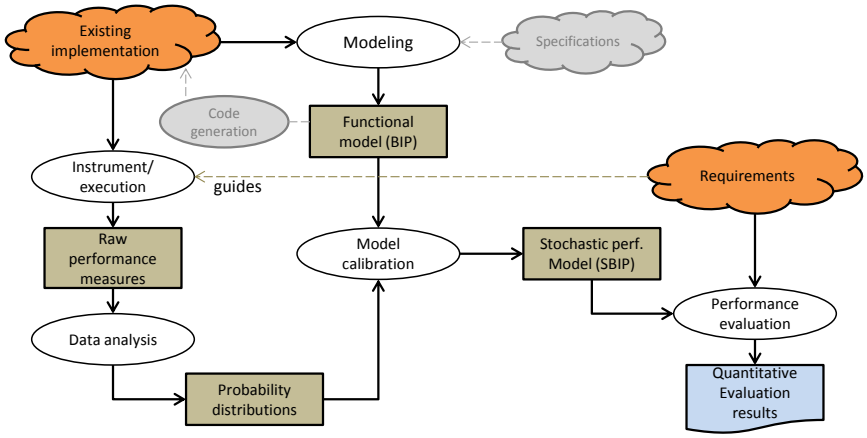

FIGURE 3.2: Methodology and framework

The advantages of the framework is that it can either take a functional model of a system obtained from a high-level specification or model an existing implementation. For the former case, an implementation is automatically generated from the model. In this use case, we consider the latter.

The system's implementation which could also be obtained by automatic code generation, is instrumented and used to collect performance measurements regarding the requirements of interest, e.g. throughput. These measurements are analyzed and characterized in the form of probability density functions with the help of statistical techniques such as sensitivity analysis and distribution fitting. The obtained probability density functions are then introduced in the functional model using a well defined calibration procedure [158]. The latter produces a stochastic timed model (when measurements concern time), which will be analyzed using the SMC engine.

Note that the considered models in this approach or workflow can be parameterized with respect to different aspects that we want to analyze and explore. Basically, the defined components types are designed to be instantiated in different contexts, e.g. with different probability density functions thus showing diverse performance behaviors. While, the model considered for analysis using SMC is a specific instance for which all the parameters are fixed, some degree of parameterization is still allowed on the verified requirements.

3.4 NDN-DPDK Modeling

In this section we present the modeling process of the NDN-DPDK from a functional to a stochastic timed model for throughput evaluation.

3.4.1 A Parameterized Functional BIP Model

Figure 3.3 depicts the BIP model of the NDN-DPDK forwarder introduced in Section 3.2 which shows its architecture in terms of interacting BIP components that can easily be matched to the ones in Figure 3.1. The presented model is parameterized with respect to the number of components, their mapping into specific CPU cores, FIFOs sizes, etc. Due to space limitations, we present in [110] the behaviors of all the components of the NDN-DPDK forwarder in Figure 3.3. It is worth mentioning that the model is initially purely functional and untimed. Time is introduced later through the calibration procedure.

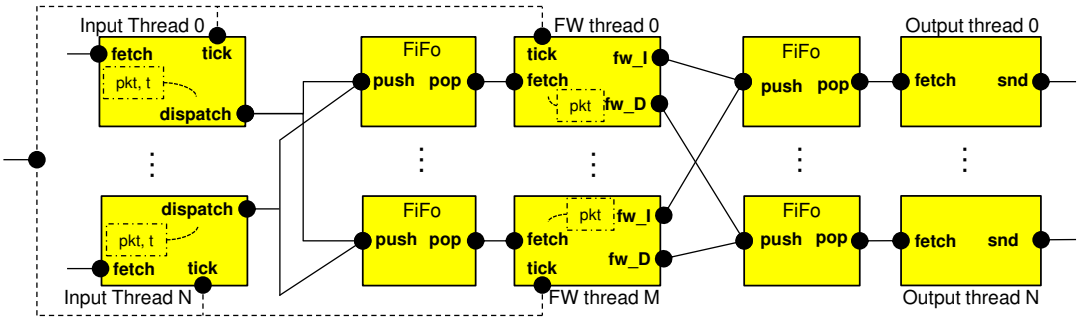

FIGURE 3.3: A functional BIP model of the NDN-DPDK forwarder

3.4.2 Building the Performance Model

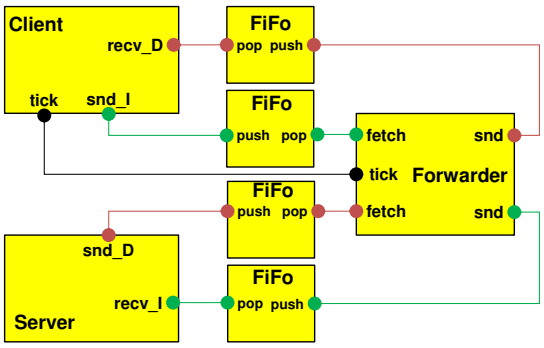

FIGURE 3.4: Considered network topology

To build a performance model for our analysis, we consider the network topology in Figure 3.4 which has a traffic generator client (consumer), a forwarder (NDN-DPDK) and a traffic generator server (producer), arranged linearly.

The green line shows the Interest packet path from the client to the producer through the forwarder and the red line indicates the Data path towards the client.

The structure of our model (Figure 3.3) calls for four distribution functions to characterize the performance:

1. Interest dispatching latency in input threads.
2. Data dispatching latency in input threads.
3. Interest forwarding latency in forwarding threads.
4. Data forwarding latency in forwarding threads.

Notice that Nack packets are out of the scope of these experiments. We identified the following factors that can *potentially* affect the system's performance:

a. **Number of forwarding threads.** Having more forwarding threads distributes workload onto more CPU cores. The cores can compete for the shared L3 cache, and potentially increase forwarding latency of individual packets.

b. **Placement of forwarding threads onto Non Uniform Memory Access nodes (NUMA).** Input threads and their memory pools are always placed on the same NUMA node as the Ethernet adapter whereas the output threads and the forwarding threads can be moved across the two nodes. If a packet is dispatched to a forwarding thread on a different node, the forwarding latency is generally higher because memory access is crossing NUMA boundaries.

c. **Packet name length measured by the number of its components.** A longer name requires more iterations during table lookups, potentially increasing Interest forwarding latency.

d. **Data payload length.** Although the Data payloads are never copied, a higher payload length increases demand for memory bandwidth, thus potentially increasing latencies.

e. **Interest sending rate from the client.** Higher sending rate requires more memory bandwidth, thus potentially increasing latencies. It may also lead to packet loss if queues between input and forwarding threads overflow.

f. **Number of PIT entries.** Although the forwarder's PIT is a hash table that normally offers $O(1)$ lookup complexity, a large number of PIT entries inevitably leads to hash collisions, which could increase forwarding latency.

g. **Forwarding thread's queue capacity.** the queues are suspected to impact the overall throughput of the router through packet overflow and loss rates. However, it does not influence packets individual latencies.

After identifying the factors with potential influence on packet latency, we instrument the real forwarder to collect latency measurements. Then, we perform statistical analysis to identify which factors are more significant. This narrows down the number of factors used and associated distribution functions.

Forwarder Instrumentation.

Factors a., b., c., d., e. and g. can be controlled by adjusting the forwarder and traffic generator configuration, while factor f. is a result of network traffic and is not in our control. To collect the measurements, we modified the forwarder to log

packets latencies as well as the PIT size after each burst of packets. We minimized the extra work that input threads and forwarding threads have to perform to enable instrumentation, leaving the measurement collection to a separate logging thread or post-processing scripts. It is important to mention that this task does in fact introduce timing overhead. Therefore, the values obtained will have a bias (overestimate) that translates into additional latency but the trends observed remain valid.

We conducted the experiment on a Supermicro server equipped with two Intel E5-2680V2 processors, 512 GB DDR4 memory in two channels, and four Mellanox ConnectX-5 100 Gbit/s Ethernet adapters. The hardware resources are evenly divided into two NUMA nodes. To create the topology in Fig. 3.4, we connected two QSFP28 passive copper cables to connect the four Ethernet adapters and form two point-to-point links. All forwarders and traffic generator processes were allocated with separate hardware resources and could only communicate over Ethernet adapters.

In each experiment, the consumer transmitted either at sending intervals of one Interest per 700 ns or per 500 ns under 255 different name prefixes. There were 255 FIB entries registered in the NDN-DPDK forwarder at runtime (one for each name prefix used by the consumer), all of which pointed to the producer node. The producer would reply to every Interest with a Data packet of the same name. The forwarder's logging thread was configured to discard the first 67 108 864 samples (either latency trace or PIT size) during warm-up period, and then collect the next 16 777 216 samples and ignore the cool down session. Each experiment represents about 4 million Interest-Data exchanges.

TABLE 3.1: Factors used. NUMA mapping is described below.

Factors	forwarding threads	Name length	Payload length	Sending intervals
Values	{1, 2, 3, 4, 5, 6, 7, 8}	{3, 7, 13}	{0, 300, 600, 900, 1200}	{500 ns, 700 ns}

We repeated the experiment using different combinations of the factors in Table 3.1 and the following NUMA arrangements:

(P1) Client and server faces and forwarding threads are all on the same NUMA,

(P2) Client face and forwarding threads on one NUMA, server face on the other,

(P3) Client face on one NUMA, forwarding threads and server face on the other,

(P4) Client face and server face on one NUMA, forwarding threads on the other.

In **(P1)**, packet latency is expected to be the smallest because all processes are placed on the same NUMA therefore, no inter-socket communication and no overhead are introduced. In **(P4)**, both Interests and Data packets are crossing NUMA boundaries twice since the forwarding threads are pinned to one NUMA whereas the client and the server faces, connected to the Ethernet adapters, reside on another. This is suspected to increase packet latency tremendously as opposed to **(P1)**, **(P2)** and **(P3)**. These suspicions predict that placement **(P1)** is the best case scenario and placement **(P4)** is obviously the worst. However, we aim at getting more insight and confidence through quantitative formal analysis. This will provide a recommendation as to which placement is better suited based on the remaining parameters combinations.

Model Fitting.

Before calibrating our functional BIP model with multiple distinctive probability distributions representing each combination of the factors, we choose to reduce the number of used distributions by performing a sensitivity analysis. This analysis examines the impact of several factors on the response (packet latency) and discovers the ones that are more important. In this chapter, we use DataPlot [49] to produce the Main Effect Plot (Figure **??**) for factors a. to e..

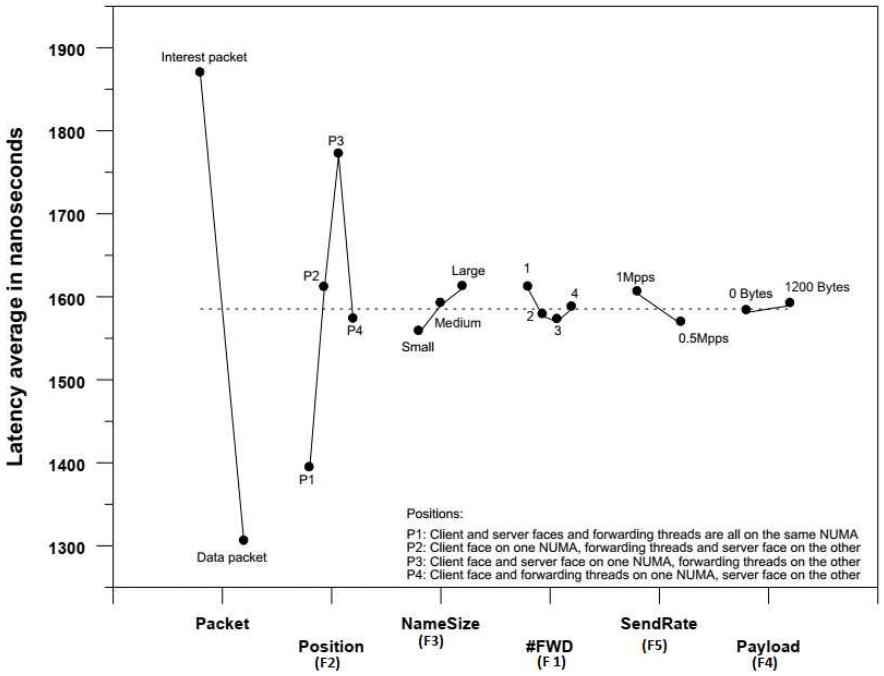

FIGURE 3.5: Main Effects Plot for Interest and Data packets

The plot shows steeper line slopes for the packet type (packet type is not a factor). We intend to show how the NDN-DPDK forwarder processes both Interest and Data differently) as well as factors (a.), (b.), (c.), and (e.) which indicates a greater magnitude of the main effect on the latency. However, the plot shows almost a horizontal line for factor d. inducing an insignificant impact on the latency. The latter is explained by the fact that the forwarder processes packet names (headers) only and doesn't read Data payloads. As for the PIT size (factor f.), it is expected to heavily increase packet latency when it is full. However, because this table's implementation is optimized for high performance and entries are continuously removed when Data packets arrive (PIT entries being satisfied), we confirmed through a correlation analysis that we can ignore this factor's impact.

Based on the analysis above, we build distribution functions for each of the factors that have greater impacts on packet latency in this study. These factors are:

1. The number of forwarding threads;
2. NUMA placement;
3. Packet name size (header);
4. Sending rate and;
5. FIFO capacity (FIFO impacts the loss rates and not individual packet latency).

With the help of an expert statistician at NIST, we followed the traditional statistical approach of conducting distribution fitting in order to map the collected measurements to their best fitted probability distribution for each of these factors [110]. We describe this process in detail in chapter 4.

Model Calibration.

Calibration is a well defined model transformation that transforms functional components into stochastic timed ones [157]. In this section, we use the probability distributions obtained above to calibrate the functional BIP model of the NDN-DPDK forwarder shown in Figure 3.3. Due to space limitations, we refer the reader to [110] where we describe the calibrated models of all the BIP components of the NDN-DPDK forwarder.

In the next section, we perform SMC on the calibrated model of the NDN-DPDK forwarder and explain the results.

3.5 Performance Analysis using SMC

3.5.1 Experimental Settings

We run the SMC tests using the probability estimation algorithm with a required confidence of $\alpha = 0.1$ and a precision of $\delta = 0.1$. Each test is configured with a different combination of values for the factors previously presented. And each execution of a test with a single set of parameters generates a single trace. The property evaluated with the SMC engine is: *Estimate the probability that all the issued Interests are satisfied, i.e. a Data is obtained in return for each Interest*. The SMC result is a probability estimation $\hat{p}$ which should be interpreted as being within the confidence interval $[\hat{p} - \delta, \hat{p} + \delta]$ with probability at least $(1 - \alpha)$. In the experiments below, the shown results corresponds to $\hat{p} = 1$.

3.5.2 Analyses Results

Queues Dimensioning.

First, we explore the impact of sizing forwarding threads queues. Each forwarding thread has an input queue. Initially, we consider a model with a single forwarding thread and vary its queue capacity with 128, 1024 or 4096 (in packets). Then set the client's sending rate to: 10^5 packets per second (pps), 10^6 pps or 10^7 pps. The results are shown in Figures 3.6 and 3.7. The Y-axis represents the Interest satisfaction rate such that 100 % (resp. 0 %) indicates no loss (resp. 100 % loss) and the x axis represents the queue capacity under different sending rates.

Figure 3.6 indicates that at 10^5 pps (blue), the Interest satisfaction rate is 100%. This means that the forwarder (with one forwarding thread) is capable of handling all packets at this sending rate (10^5 pps of packet size 1500 bytes is equivalent to 1.2 Gbps), under any queue size. However, under a faster sender rate (where a single forwarder shows signs of packet loss) we unexpectedly observed a better Interest satisfaction rate with a smaller queue (Q=128). After a thorough investigation of the real implementation, we found out that the queues don't have proper management in terms of insertion and eviction policies that would give priority to Data over Interest packets. In the absence of such policy, more Interests would be queued while Data packets would be dropped resulting in Interests not being satisfied, thus lower

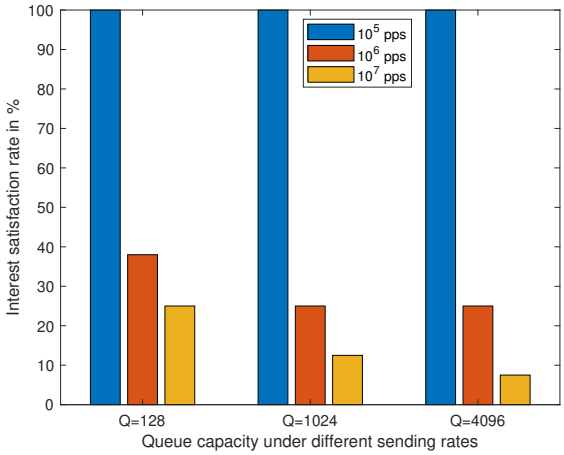

FIGURE 3.6: One Forwarding thread with different sending rates

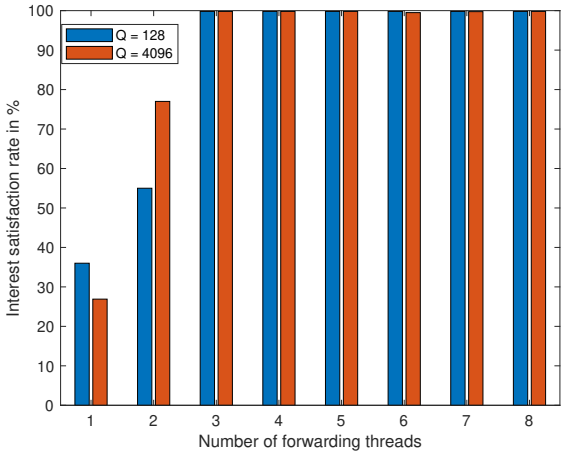

FIGURE 3.7: Many Forwarding threads with a sending rate set to 10^6 pps

performance (Interest satisfaction rate). *It is thus advised for the final implementation of the NDN-DPDK forwarder, to use a queue capacity smaller than* 128 *packets when the forwarder has a single forwarding thread and packets are sent at a fast rate.*

Similarly, we explore whether this observation remains true with more forwarding threads. In order to do that, we run SMC again on eight different models each with a different number of forwarding threads (1 to 8) under a sending rate of 10^6 pps (1 Interest per 1 us) where a loss rate was observed in Figure 3.6. Then, we experimented with two queue capacities, namely 128 and 4096 packets. The results are reported in Figure 3.7. The x Axis represents the number of forwarding threads while the y axis depicts the Interest satisfaction rate.

We observe that the queue size matters mainly in the case of a model with one and two forwarding threads. In fact, for a two threads model, a bigger queue size is preferred to maximize the performance, unlike when a single thread is used. As for the other six models, both sizes achieve almost 100 % Interest satisfaction. This is due to the fact that three forwarding threads or more are capable of splitting the workload at 10^6 pps and can pull enough packets from each queue with a minimum loss rate of 0.02 % . *This result stresses that, to avoid being concerned about a proper queue*

size, more threads are needed for handling a faster sending rate with minimum Interest loss.

NUMA placement, number of forwarding threads and packet name length.

Another aspect to explore, is the impact of mapping the forwarding threads and/or NDN Faces to the two NUMA nodes (0, 1) under different sending rates and for multiple name lengths where Face 0 exchanges packets with the client and Face 1 with the server. To do that, we consider the four NUMA arrangements (P1), (P2), (P3) and (P4) in section 3.4 as well as the factors in Table 3.1 in the SMC analysis.

In Figures 3.8 to 3.13, each row represents experiments with similar packet name lengths {small=3, medium=7, large=13} and a queue capacity of 4096. The right-hand column indicates results for a faster sending rate of $2 * 10^6$ pps (500 ns interval) while the left-hand one shows results for a slower sending rate of $1.42 * 10^6$ pps (700 ns interval). The six figures includes four curves where each corresponds to the four NUMA arrangement options: P1 to P4.

The six Figures 3.8 to 3.13 show that Interest satisfaction rates scale up with the increase of forwarding threads then reach a saturation plateau where adding more threads can no longer improve the performances. Furthermore, with fewer forwarding threads, the loss rate is unavoidable and exceeds 80 %. This is because the sending rate is faster than the forwarding threads processing capabilities causing their FIFO queues to saturate and start dropping packets frequently. However, under a slower sending rate and packets with small, medium and large name lengths (3, 7, 13), Figures 3.8, 3.10 and 3.12 show that a maximum satisfaction rate of over 90 % is achievable with only five forwarding threads. Whereas when the client is generating packets faster at 2 Mpps, a saturation plateau of over 90 % is reached at six threads or more for small and medium names (Figures 3.9 and 3.11) and a plateau of slightly over 70 %, with five threads, for larger names (Figure 3.13). Also, Figures 3.8 and 3.10 demonstrate that placing all processes (threads and faces) on a single NUMA (placement P1) outperforms the other three options. This observation is explained by the absence of inter-socket communication thus less timing overhead added such as in the case of the purple plot where packets are crossing NUMA boundaries twice from Face 0 to the forwarding threads then through Face 1 and back (placement P4).

Figures 3.9 and 3.11 show the impact of increasing the sending rate on packets with smaller names. In this case, it is preferred to also position all the processes on one NUMA such as the case of the yellow plot of the P1 series because NUMA boundary crossing usually downgrades the performance. In fact, the difference between no NUMA crossing and the double crossing (yellow and purple series respectively) is approximately 30 % loss rate with more than five threads. The second best option P2 which is placing the forwarding threads on the NUMA receiving Interest packets with Face 0 (NUMA hosting the Ethernet adapter that receives Interests from the Client). However, when the number of threads is not in the saturation zone and the threads get overworked and start to loose packets, it is recommended to opt for placement P3. *Based on these results, we recommend that for small to medium names, to use a maximum of eight threads but no less than five arranged as in placement P1 for optimum performances under a slower or a faster sending rate.*

With a larger name however, Figure 3.12 depicts an unexpected behaviour when using three threads or less. In this case, placing the forwarding threads on the same NUMA as Face 1 (which is the Ethernet adapter connected to the server and receives Data packets), surpasses the other three options. Our explanation is that since forwarding threads take longer times to process incoming packets due to their longer

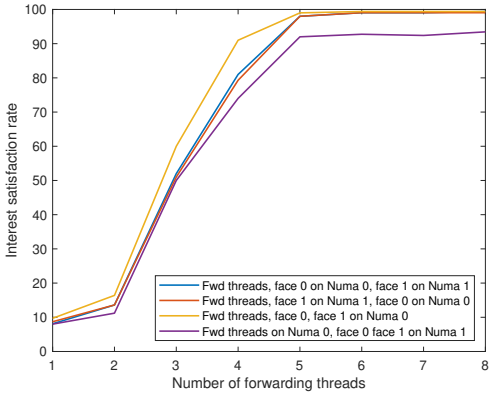

FIGURE 3.8: small names, 700 ns

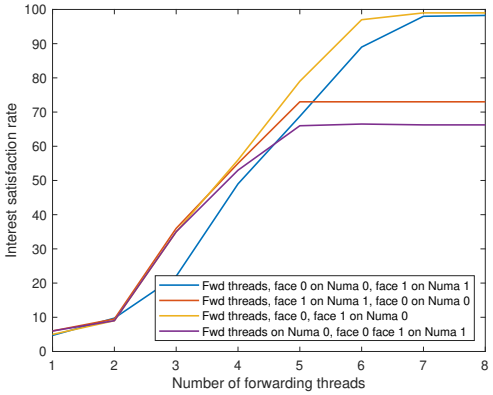

FIGURE 3.9: small names, 500 ns

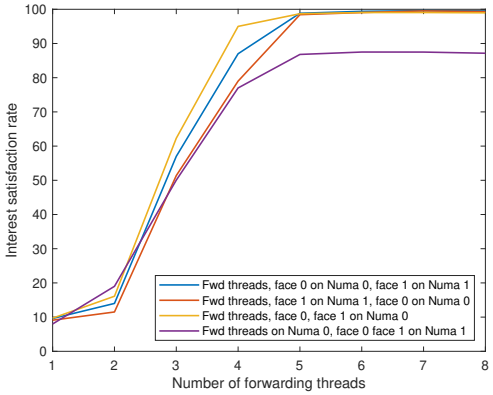

FIGURE 3.10: medium names, 700 ns

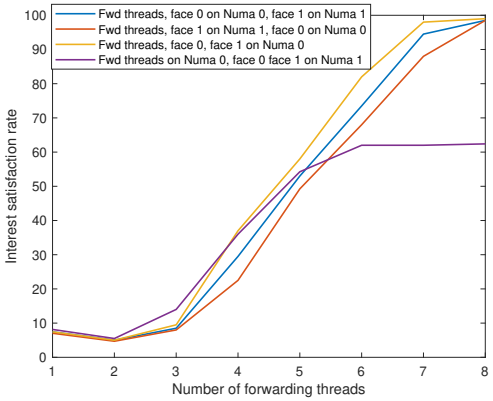

FIGURE 3.11: medium names, 500 ns

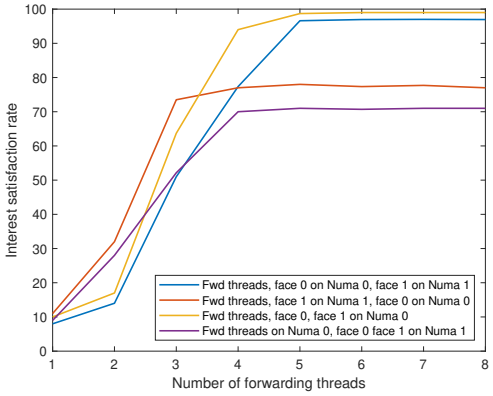

FIGURE 3.12: large names, 700 ns

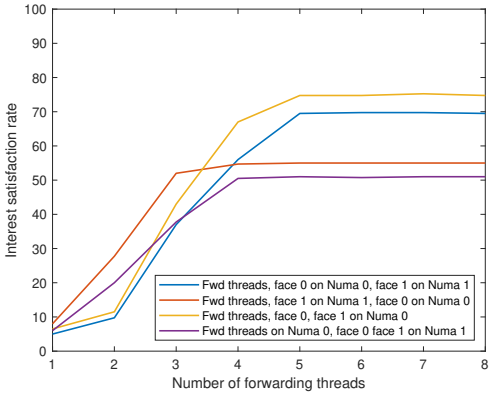

FIGURE 3.13: large names, 500 ns

name and timely lookup, particularly for Interests as they are searched by names inside the two tables (PCCT and FIB) rather than a token such as the case for Data packets. Placing the forwarding threads with the Data receiving Ethernet adapter connected to Face 1, has the potential to yield better results by quickly processing packets after a quick token search especially when the workload is bigger than the threads' processing capacity. When the sending rate is increased, the same results are observed in Figure 3.13 for a similar name length but with a decrease in performance. *Thus, we recommend for larger names to use NUMA arrangement P3 only when the number of forwarding threads is less than three regardless of the sending rate (not advised due to high loss rate).*

3.6 Lessons learned

This study shed light on a new networking paradigm called Named Data Networking (NDN) and its forwarding daemon. Ongoing NDN research includes the development of high-speed data plane forwarders that can operate at a hundred gigabits per second while using modern multi-processor server hardware and kernel bypass libraries. In this paper, we discussed the results of a performance evaluation effort we undertook to reach well-founded conclusions on how the NDN forwarder prototype developed by NIST (NDN-DPDK) behaves in a network in terms of achievable Interest satisfaction rate.

We conducted an extensive analysis under different factors such as the number of threads carrying tasks and function mapping to CPUs, using a model-based approach and statistical model checking. Given the wide array of design parameters involved, this effort contributes valuable insights into protocol operation and guides the choice of such parameters. The use of statistical model checking for performance analysis allowed us to discover potential sub-optimal operation and propose appropriate enhancement to the queue management solution. This has been taken into account in the ongoing NDN-DPDK forwarder implementation. Moreover, our extensive analysis provides a characterization of the achievable forwarding throughput for a given forwarder design and available hardware resources which would not have been possible to obtain, with such controllable accuracy, using traditional measurements and statistic methods. Furthermore, these results were communicated and shared with members of the NDN community in a conference throughout a poster interaction and gained attention from researchers who were interested in the methodology and its applications. In addition to that, the use of a BIP model refined at the right level of abstraction allows the generation of executable code that could be used instead of the real implementation.

It is important to note however, that our analysis depends largely on a stochastic model obtained using samples of data collected from the actual implementation of the forwarder which is suspected to have introduced timing overhead. Nevertheless, the trends observed throughout this study remain accurate and have provided valuable insight to the actual code. In the future, this analysis will be extended to answer the reverse question, namely **Given a desired throughput, what is the best hardware setup and the forwarder design to use?** rather than the question **Given a hardware setup and a forwarder design, what is the maximum achievable throughput?** that we have investigated in this study which we published in the International Symposium on Automated Technology for Verification and Analysis (ATVA) [112, 110].

So far in this chapter, we examined an application of our proposed methodology that relies on building parametrized BIP models of the NDN-DPDK forwarder. Then, we collected measurement data by executing the forwarder's prototype on a local testbed. Each set of measurements is then analyzed via several statistical tests to select probability distributions that best fit each dataset. This is referred to as the distribution fitting process. We conducted this step under the supervision of an expert statistician Dr. James J Filliben [97]. The identified probability distributions are then integrated into the BIP models via the stochastic extension SBIP which we evaluate using the BIP^{SMC} engine.

It is important to emphasize that mapping a dataset to a probability distribution is a tedious statistical task that is typically done by expert analysts as it requires a good statistical knowledge and familiarity with many commonly used distributions, mostly because the statistical methods used involve human interpretation of graphs or numerical tests which is above the expertise of some analysts. In this context, it is really important to fit the system data to accurate probability distributions in order to build rich stochastic performance models and to be able to quantitatively analyze them via SMC. Any misconception on the data and inaccurate fitted probability distributions will result in wrong verification of the whole system.

We discuss this point in the upcoming chapters (part 2 of this thesis). We present a semi-blackbox method for automating the distribution fitting process to help analysts select the 'best' probability distribution among many commonly models. Our approach relies on deep learning. In chapter 6, we overview our developed tool 'DeepFit' that combines traditional statistical tests and the trained neural networks to conduct distributional modeling with high accuracy.

Part II

Automated distributional modeling

Chapter 4

Statistical inference

In the first part of this thesis, we presented a methodology and its associated analysis tools for rigorous system modeling and performance evaluation of networking systems (e.g., throughput, latency, delay). We recall that, this methodology is model-based and takes as inputs: (a) a stochastic performance model with the semantics of a Discrete Time Markov Chain (DTMC) expressed in the *SBIP* formalism. (b) a formalized performance requirement expressed in Probabilistic Bounded Linear Temporal Logic (PBLTL). (c) and a set of parameters to control the accuracy of the verification. The analysis phase relies on the statistical model checking techniques, implemented by the BIP^{SMC} engine, which evaluates the DTMC model against the expected properties. The results of this process produces an answer to one of these two questions, depending on the selected algorithm (i.e., Hypothesis Testing (HT), Probability Estimation (PE)):

1. ***Qualitative:*** *Can the model satisfy the specified requirement with a probability p?*
2. ***Quantitative:*** *What is the probability that the model satisfies the requirement specification?*

Additionally, we also briefly described the missing element in the workflow of this methodology which is how to characterize measurement data, obtained from the system to be verified, as probability distributions (i.e., distribution fitting). In general, the latter is an important preliminary step for any further analysis in science or engineering. However, in the context of our methodology, it is extremely crucial to obtain accurate probability distributions since any wrong characterization of the data could lead to faulty verification results. Typically, distribution fitting is a task that is done in collaboration with expert statisticians since it requires a good statistical knowledge, familiarity with many commonly used distributions and the ability to correctly interpret the results of graphs. As such, this second part of the thesis presents an alternative approach which doesn't require prior knowledge of statistical methods nor previous assumptions on the available data. Instead, using Deep Learning, the best candidate distribution is extracted from the output of a neural network that was previously trained on a large suitable database in order to classify an array of observations into a matching distributional model.

This part of the thesis is organized as follows. Chapter 4 presents an overview of the concept of distribution fitting and examines the traditional and associated tools that expert statisticians rely on to fulfill this task. Next, chapter 5 introduces our suggested method for distributional modeling which is based on the use of trained neural networks to predict the best fit distribution from datasets, without the need for a background in statistics, prior considerations of the process or phenomenon under study nor familiarity with several distributions. Finally, chapter 6 concludes this thesis with a tool, called *DeepFit*, that we developed, solely, for this purpose

and highlights its usefulness and benefits. In this final chapter, we also explain how *DeepFit* can be incorporated into the statistical model checking analysis or used as a standalone tool for any data analysis in science and engineering.

In this chapter we start with describing the concept of distribution fitting. Next, we present the traditional workflow that has been used for many years by statisticians and examine its strengths and weaknesses. Finally, we propose a complementary solution that can be used hand in hand with the traditional approach to offload some of its pitfalls.

4.1 Distribution fitting

Probability distributions [197, 133] are a fundamental concept in statistics. They play an important role in science and engineering due to the multitude of applications that they can serve, some of which include:

1. Generating random numbers that follow a specific distribution in certain simulation studies, in order to learn how a stochastic system would react.
2. Modeling univariate data with a specific probability distribution.

The first application is rather simple as the analyst already knows which distributional model to use to generate random numbers. Given that each model is typically defined in terms of a probability density function (pdf), that is, a mathematical expression representing the probability that a measured variate has the value x. The analyst can simply use one of these well known pdf to create a data set that follows a specific model. Figure 4.1 shows a probability density plot for the normal distribution in its standard form (i.e., the mean is equal to zero and the standard deviation is equal to one). The pdf function is:

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$$

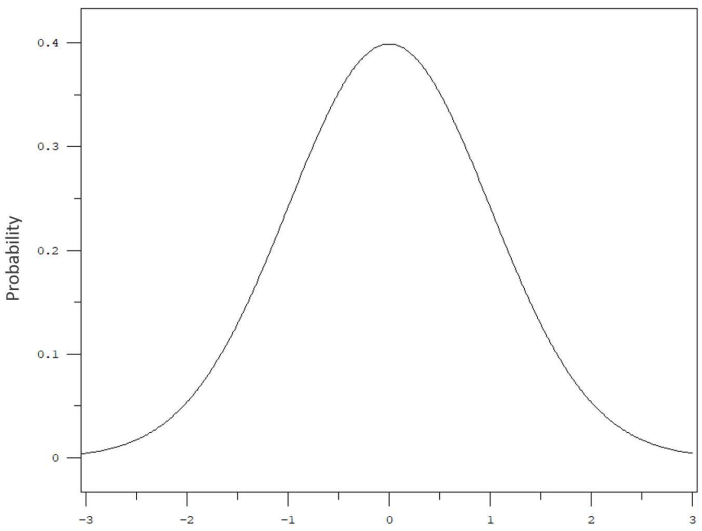

FIGURE 4.1: Normal probability distribution

Figure 4.2 shows a plot of the extreme value type I (also known as the Gumbel distribution). Note that the gumbel distribution has two forms:

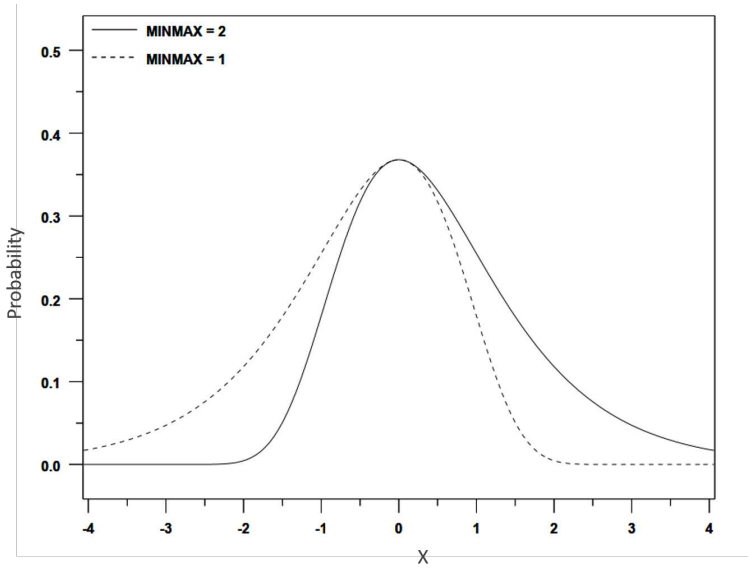

FIGURE 4.2: Gumbel distribution: types I and II

(a) Gumbel min: the general form of the extreme value type I probability density function is:

$$f(x) = \frac{e^{\frac{x-\mu}{\beta}} e^{-e^{\frac{x-\mu}{\beta}}}}{\beta}$$

(b) Gumbel max: the general form of the extreme value type I probability density function is:

$$f(x) = \frac{e^{\frac{-(x-\mu)}{\beta}} e^{-e^{\frac{-(x-\mu)}{\beta}}}}{\beta}$$

Where μ is the location parameter and β is the scale parameter.

The second application is another common use case of probability distributions. Modeling a set of empirical observations obtained from repeatedly measuring a variable of a system or phenomenon by a well known probability distribution is referred to as distributional modeling or distribution fitting. Distribution fitting is a powerful statistical technique that can be used in several contexts. Appropriate distributional models are used to more accurately assess uncertainty of the measured variable and, in particular, to assess the uncertainty over the full range of the data. They can also be used to interpret the random fluctuations in the measurements as well as possibly predict future frequency values of the measured variable to fall under a certain interval.

Distributional modeling consists of running several statistical tests and typically depends on certain assumptions for the underlying distribution. Many tests are based on the assumption of normality, that is the underlying model is the normal probability distribution (Figure 4.1) which has a bell shape like curve and has symmetric tails. However, this is not always true as the collected data may not be normally distributed.

Although a poorly chosen distributional model may suffice for measuring and assessing the uncertainty of averages, this will not be the case for data that show

signs of asymmetry and/or extended tails. In many applications (e.g., reliability), accurate assessment of the tail behavior is more critical than the average. (see Figure 4.2 for an example of a pdf with a tail). Obtaining an appropriate distributional model can be an exhaustive process that takes time, patience and requires previous knowledge of statistics and is, therefore, a difficult task for some analysts.

In the next section we explore a non extensive list of tools that statisticians have been using for years to map data to an appropriate distributional model.

4.2 Traditional methodology

Statisticians rely on several graphical and quantitative methods to fit data to specific distributions. Figure 4.3 presents the traditional approach used by statisticians to obtain an appropriate distributional model of a uni-variate dataset. This approach consists of four steps:

1. Data screening;
2. Exploratory analysis;
3. Parameter estimation;
4. Evaluation.

In the next subsections, we will examine these steps in detail.

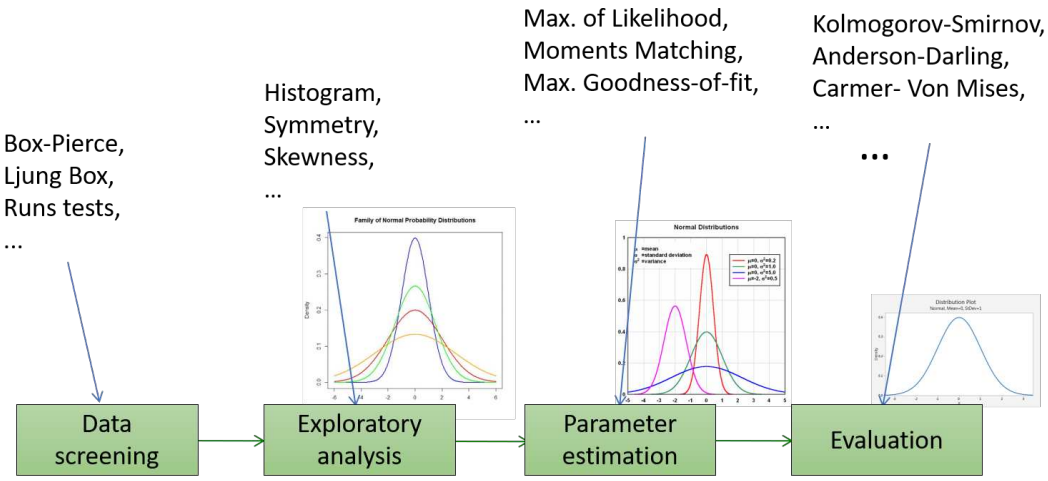

FIGURE 4.3: Traditional approach of conducting distribution fitting

4.2.1 Data screening

This initial step consists of assessing whether the data are in fact independent with each other, i.e. the collected dataset is from a random process. This can be done either via standard statistical tests or via graphical tools.

Numerical techniques:

There are many standard statistical tests to assesses data randomness or auto-correlation. Such tests include the runs test and Ljung Box:

- **Ljung Box** is based on the auto-correlation plot but assess the overall randomness of a time series Y (i.e. data points collected over a specific time horizon) over several lags using the following mathematical formula:
  Let H_0 be the null hypothesis that data comes from a random process
  let H_a be the hypothesis that data doesn't come from a random proces.

  Given a time series Y of length n, the test statistic is defined as:

$$Q(m) = n(n+2)\sum_{k=1}^{m}\frac{r_k^2}{n-k}$$

  where m is the number of lags considered and r_k is the auto-correlation estimate at lag k. r_k is computed as follows:

$$r_k = \frac{\sum_{i=1}^{n-k}(Y_i - \bar{Y})(Y_(i+k) - \bar{Y})}{\sum_{i=1}^{n}(Y_i - \bar{Y})^2}$$

  This test rejects the null hypothesis (H_0) when:

$$Q(m) > X^2_{1-\alpha,h}$$

  where $X^2_{1-\alpha,h}$ is the chi-square distribution table value with h degrees of freedom and significance level α (see [9, 76] for more details).

- **The runs Test** is another statistical test that is used to search for data auto-correlation. A runs test is defined as a sequence of symbols (e.g., $+/-$). For example, tossing a coin 20 times could produce the following sequence of heads (+) and tails (-):

$$++--+-+++++-++---++$$

  A run is basically a series of consequent similar symbols and its length is the count of symbols. In a random data set, the probability that the $(k+1)^{th}$ symbol is larger or smaller than the k^{th} symbol follows a binomial distribution, which forms the basis of the runs test. Let:

  H0: The sequence was produced in a random manner

  H1: The sequence was not produced in a random manner

  The test statistic is calculated using the following formula:

$$Z = \frac{R - \bar{R}}{s_r}$$

  where R is the observed number of runs, $\bar{R}$, is the expected number of runs, and s_r is the standard deviation of the number of runs. The values of $\bar{R}$ and s_r are computed as follows:

$$\bar{R} = \frac{2n_1n_2}{n_1+n_2} + 1$$

$$s_r^2 = \frac{2n_1n_2(2n_1n_2 - n_1 - n_2)}{(n_1 + n_2)^2(n_1 + n_2 - 1)}$$

n_1 and n_2 are the total numbers of positive and negative values in the series. Given a significance level α, the H_0 hypothesis is rejected if:

$$|Z| > Z_{1-\frac{\alpha}{2}}$$

Note that for large samples, $Z_{1-\frac{\alpha}{2}} = 1.96$ and for smaller samples, there are tables to determine critical values based on n_1 and n_2 (See [4] for more information).

Graphical techniques

Data independence can also be assessed graphically using plots such as the lag plot and the auto-correlation:

- **The Lag plot**, for example, can show if the dataset exhibits any identifiable structure or patterns and can generally provide answers to the following questions:

  1. Is the data random?
  2. Is there serial correlation in the data?
  3. What is a suitable model for the data?
  4. Are there outliers in the data?

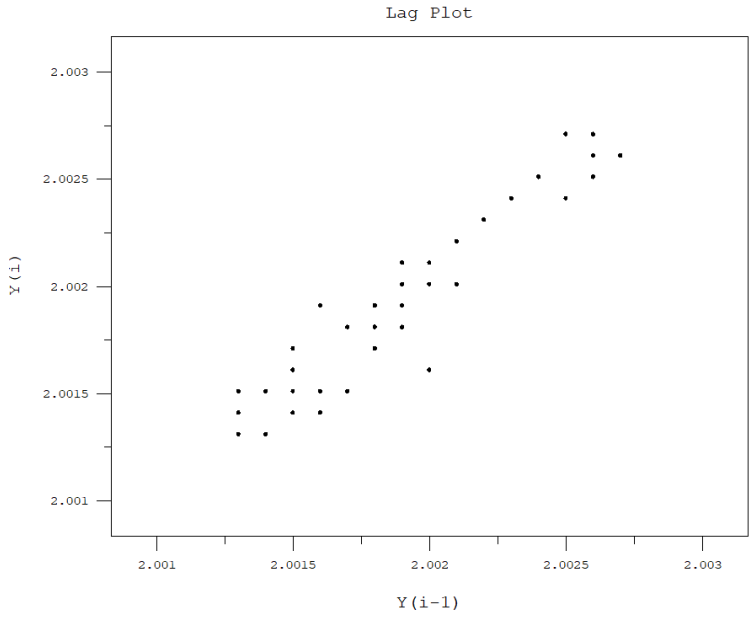

FIGURE 4.4: Lag plot indicating a strong dependency in the data

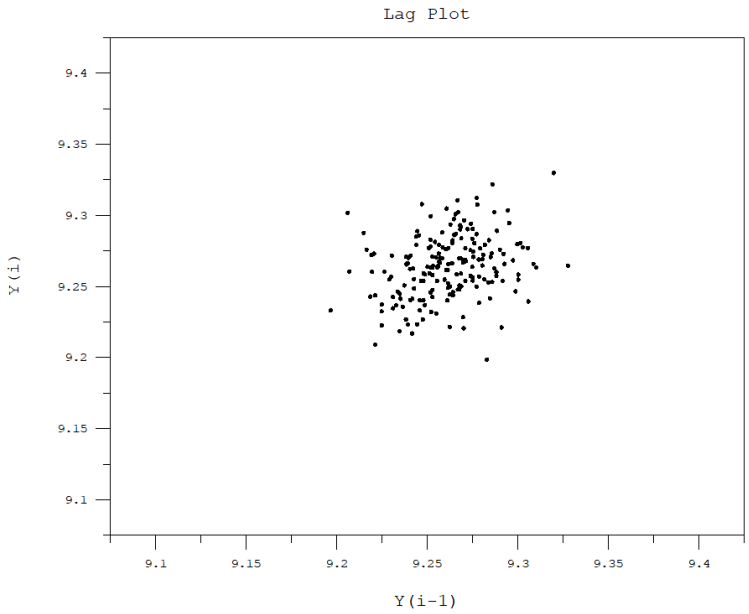

FIGURE 4.5: Lag plot indicating independence in the data

For a given dataset $Y = Y_1, Y_2, ..., Y_n$, the lag plot is obtained by plotting $Y_i - 1$ for all i in the x-axis against Y_i for all i on the y-axis. Figures 4.4 and 4.5 show two Lag plot examples obtained from two different datasets. The first plot exhibits a linear pattern which means that the data is strongly non-random and a strong dependency is present. The second plot, shows no visible patterns or structures which indicates that the dataset comes from a random process.

- **The auto-correlation plot** is another great tool for checking randomness in a time series. This plot is obtained by computing the auto-correlation function for the data values at different time intervals. Its x-axis represents the time progress and the y-axis represents the values of the auto-correlation function. The graph itself shows vertical lines ("spikes") corresponding to each lag. The height of each spike shows the value of the auto-correlation function for the lag. If at any interval separation an auto-correlation value is found to be significantly greater than zero (i.e., the lag is very high), then surely the dataset is non random. However, the same can not be said on the opposite statement. That is, if the dataset is uncorrelated (i.e. lag is small), it doesn't necessarily mean that it's completely random as the auto-correlation function is just one of many measures for checking randomness and the non-randomness feature can still be exhibited via other techniques.

  Consider the examples in Figures 4.6 and 4.7. The plot shows that most of the spikes are not statistically significant which insinuates that the measured variable is not highly correlated and might potentially be random (further tests are still required). Note that the first spike is significantly higher than the rest. This is completely normal as it is always set to 1 by definition. The remaining lags are all near zero. In addition, no significant patterns are apparent. From the plot in Figure 4.7, however, we conclude that the data is sourced from an auto-regression model that shows strong positive auto-correlation.

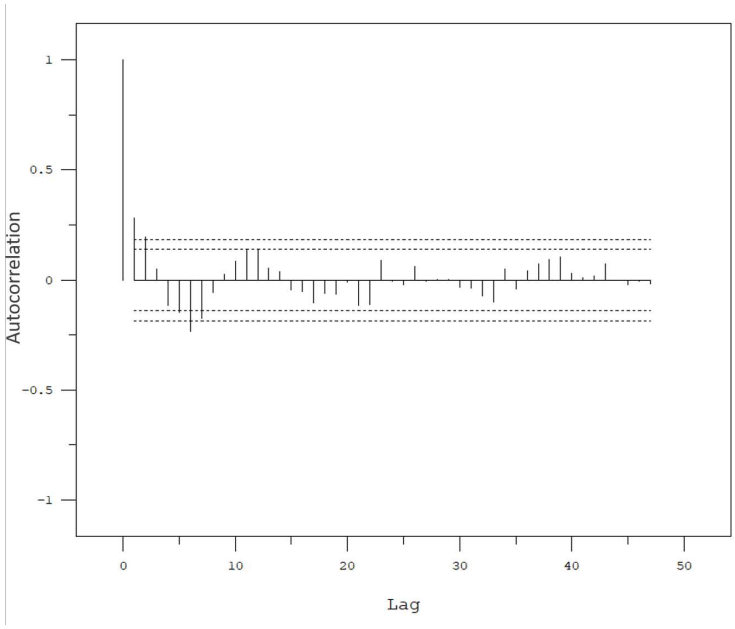

FIGURE 4.6: No strong correlation is noticed

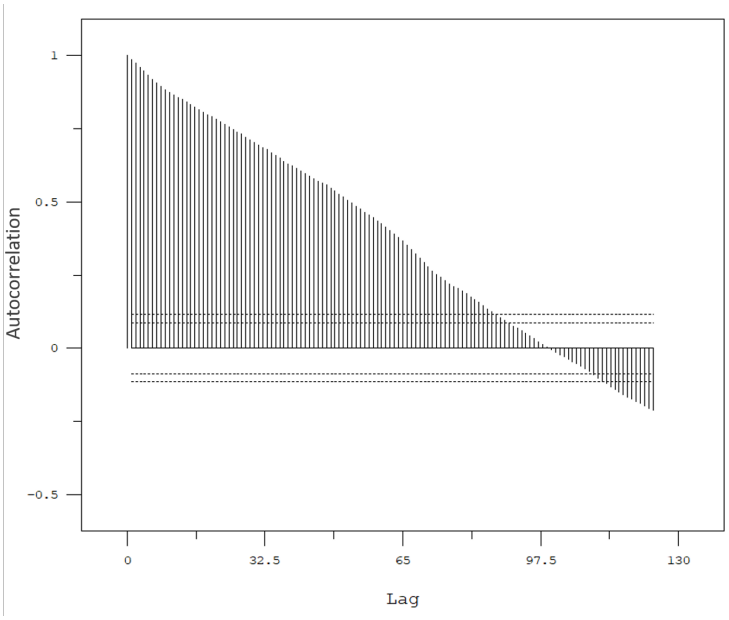

FIGURE 4.7: Strong correlation is noticed

4.2.2 Exploratory analysis

This step consists of graphically summarizing the distribution of a univariate dataset. That is, identifying potential distributional models that fit the data using well known plots. Typically, a histogram [96] or a kernel density plot [104] is used to help identify the basic shape of the underlying distribution as well as certain properties such as the skewness and the presence of multiple modes in the data. Generally, these two plots provide answers to the following questions:

1. What kind of population distribution does the data come from?
2. Where is the data located?

3. How spread out is the data?
4. Are the data symmetric or skewed?
5. Are there outliers in the data?

- **Histograms:** are a graphical tool for displaying the distribution of a variable. The response variable is divided into equally sized intervals referred to as bins. The number of occurrences of the response variable is calculated for each bin. There are at least three basic ways of displaying the y-axis which produces three types of histograms:

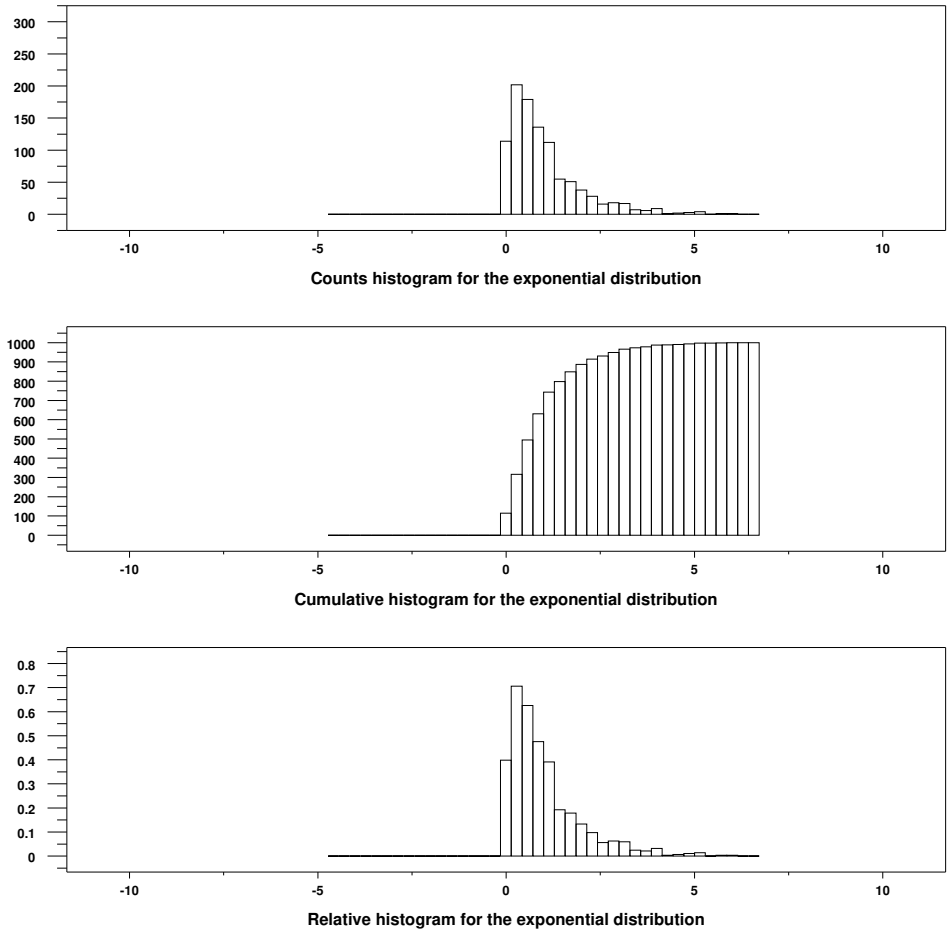

FIGURE 4.8: Counts histogram

1. The counts histogram in which the y-axis simply indicates a count of observations in each bin (Figure 4.8);
2. The cumulative histogram that is a variation of the count histogram in which the vertical axis gives the observation counts for that bin plus all previous bins (Figure 4.8);
3. The relative histogram that is obtained by normalizing the counts per bin usually by dividing the count of each bin by the total number of observations to generate the relative proportion of observations. In this relative histogram, the sum of the y-coordinates should be one or 100 if a

percentage scale is used (Figure 4.8). This is actually an estimator of the underlying probability distribution.

- **Kernel density plot (kdp)** also referred to as the prazen window is obtained by plotting the kernel density estimator $f_n(n)$, of a set of n points $\{X_1, X_2, ...X_n\}$:

$$f_n(x) = \frac{\sum_{i=1}^{n} K(x - X_i)}{nh}$$

where K is the kernel function and h is the smoothing parameter or window width. Figure 4.9 shows a kdp of a dataset with the uniform distribution as the underlying model.

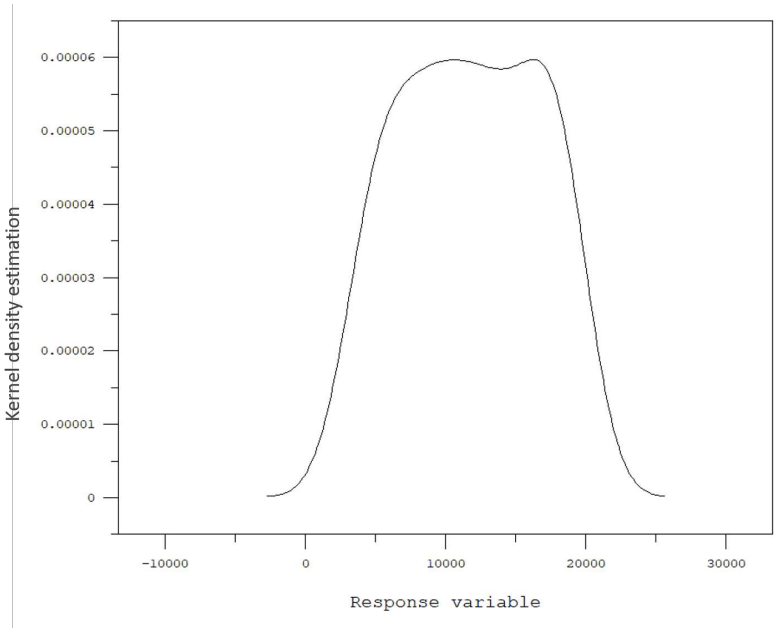

FIGURE 4.9: Kernel density plot for the Uniform distribution

Using the knowledge provided in this section, we attempt to apply it to a dataset. Figures 4.10 and 4.11 present (respectively) a counts histogram and a kernel density plot of 500 normally distributed observation points. Without knowing the origin of the dataset and based solely on the plots, we learn that the set has no extreme points and that it's uni-modal (i.e., has a single peak). Moreover, the shapes of these two plots are symmetric with a bell-like curve. All these observations clearly indicate that the dataset follows a normal distribution.

However, identifying good potential models from plots requires human interpretation which maybe subjective and change from one person to another. Furthermore, many probability distributions are not a single distribution but are in fact a family of distributions. In these families a shape parameter is responsible for changing the shape of the distribution. Therefore, exploratory analysis typically requires some degree of statistical knowledge, experience and familiarity with several commonly used distributions.

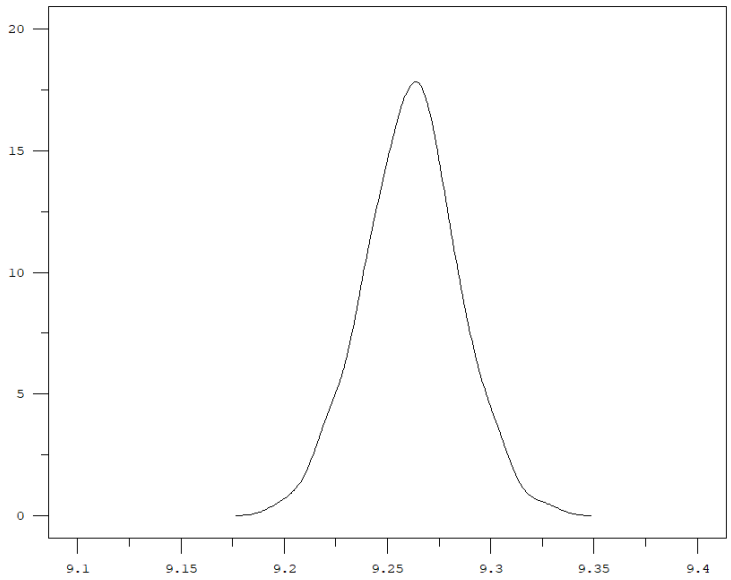

FIGURE 4.10: Kernel density plot

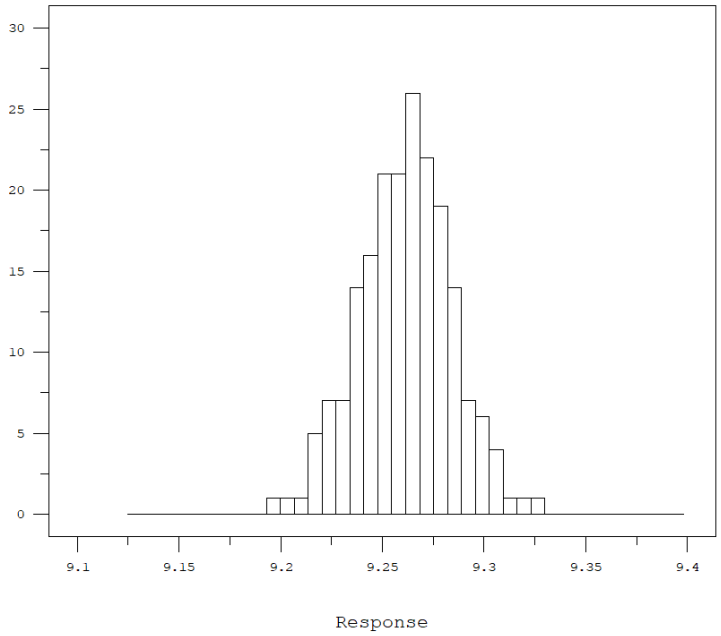

FIGURE 4.11: Counts histogram

4.2.3 Parameter estimation

After selecting the best candidate distribution that fits the data from the previous step, the next step focuses on estimating the parameters of the chosen distribution (i.e., location, scale and shape).

Probability distributions are characterized by three types of parameters: a location, a scale and one or more shape parameters. The standard form of the distribution is the case where the location parameter is zero and the scale parameter is one. Given a graph of the standard form of the probability density function, the effect of a non-zero location parameter is to shift the graph left for negative location

values or right for positive location values, on the horizontal axis. The effect of a scale parameter is to either stretch the graph on the horizontal axis, for scale parameters greater than one, or to compress the graph along the horizontal axis, for scale parameters less than one. Figures 4.13 and 4.14 show examples of this for the normal distribution whereas Figure 4.12 shows the impact of the shape parameter γ for the weibull distribution. The relationship between the probability density function's general form and its standard form (i.e., location and scale not equal to zero and one) is:

$$f(x;a,b) = \frac{f(\frac{(x-a)}{b};0,1)}{b}$$

where a and b are the location and scale parameters, respectively.

Any parameter that is not a location or scale (or a parameter that is a function of the location and scale) is a shape parameter. The shape factor allows a distribution to take a variety of forms, thus, creating a distribution family. By shape, we mean properties such as skewness and kurtosis (peakedness). The location and scale parameters have no effect on these properties. The weibull distribution is an example of a distribution that has a shape parameter (γ). Figure 4.12 plots the weibull pdf for different γ values: 1, 2, 3, 0.5.

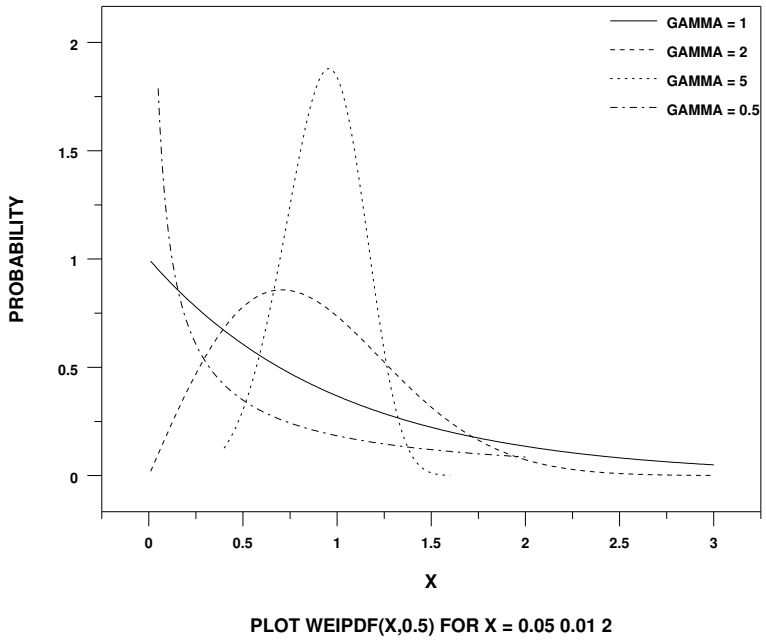

FIGURE 4.12: Weibull pdf for various values of the shape parameter (γ)

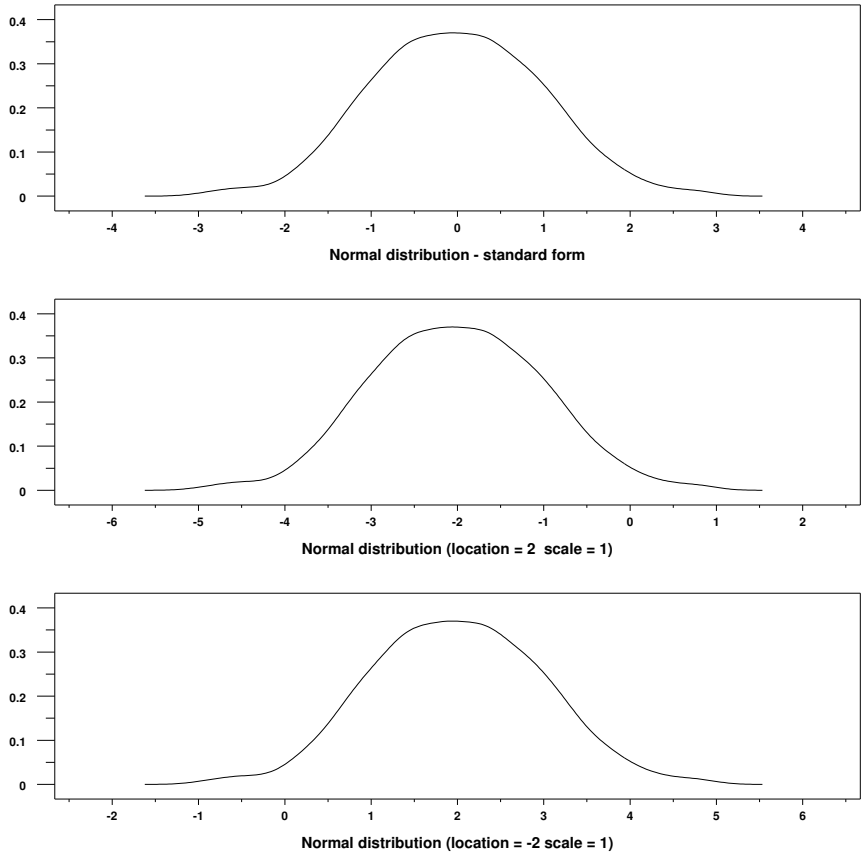

FIGURE 4.13: Impact of the location parameter

Estimating the parameters of a probability distributions (i.e. identifying the location, scale, shape values) is generally done by solving an optimization problem in which the unknown parameters of the distribution are the variables of the objective function that's being minimized or maximized and the data points are the coefficient of this function. There are various methods, both numerical and graphical, for estimating the parameters of a probability distribution. However, only a few methods are widely used because they result in parameter estimators with good statistical properties. The major ones are the Maximum likelihood and the least squares method. In this section, we illustrate four methods for the parameter estimation:

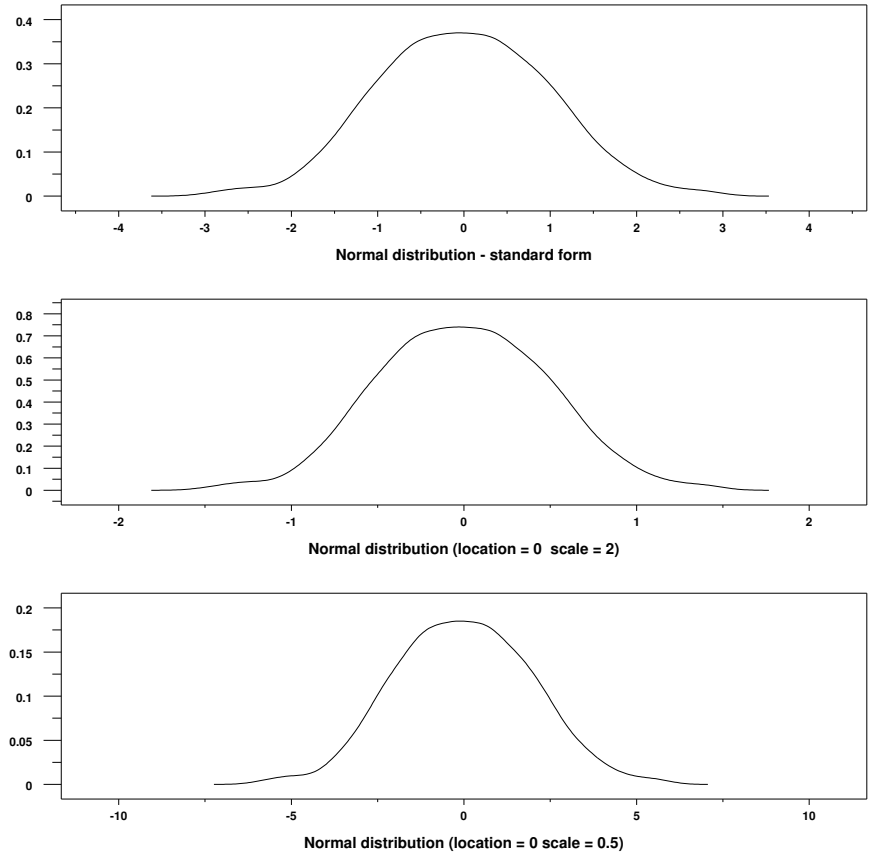

FIGURE 4.14: Impact of the scale parameter

1. **The maximum likelihood estimation (MLE)** consists of maximizing the likelihood function defined by:

$$L(\theta) = \prod_{i=1}^{n} f(x_i, \theta) \tag{4.1}$$

where f is the probability density function for the selected distribution, $x1, x2, \ldots, xn$ are the observation points and $\theta = \theta_1, \theta_2, \ldots, \theta_k$ are the parameters to estimate.

Let's consider the exponential distribution as an example. The pdf is defined by:

$$f(x) = \lambda e^{-\lambda x} \forall x \geq 0 \tag{4.2}$$

The likelihood function to maximize is:

$$L(\theta) = \prod_{i=1}^{n} \lambda e^{-\lambda x} = \lambda^n e^{-\lambda \sum_{i=1}^{n} x_i} \tag{4.3}$$

In this case, solving this function means finding the value of the scale parameter $b = 1/\lambda$. Note that the location value is set to 0 in this pdf.

MLE is a very consistent and efficient approach to the parameter estimation problem, particularly when the model is correctly assumed. Its estimates can be developed for a large variety of estimation situations and its algorithm is

already implemented in several statistical software packages and for many of the commonly used distributions.

MLE produces unbiased estimates in larger data samples. However, it is usually sensitive to outliers and is often biased for smaller samples. In addition to that, MLE's numerical estimation is usually non-trivial and requires heavy mathematical solvers to solve the likelihood function for certain distribution models. Fortunately, this is easily mitigated as it is currently supported by many general purpose statistical software programs.

2. **The Probability Plot Correlation Coefficient Plot (PPCC)**: This method is a graphical technique that is used to identify the shape parameter when the underlying distribution of the data is suspected to be a distribution family such as weibull, lognormal, etc. As stated before, distribution families are defined by three parameters: location, scale and one or more shape parameters. Once the PPCC plot identifies a good value for the shape parameter, the probability plot is then generated to find the remaining parameters (location an scale). The probability plot can also assess the accuracy of the fitted parameters.

3. **Method of moments (MOM)**: This method had the advantage of simplicity as its formula tends to be straightforward. However, it is not currently supported on most general purpose statistical software. Moreover, this method is not known to have the desirable optimal properties of the two major estimators: maximum likelihood and least squares.

MLE is typically the estimation choice method for statisticians, due to having optimal theoretical properties. However, the MLE estimates approach their optimality properties as the sample size gets larger. Note that, the MOM estimates tend to be simpler than MLE estimates, but typically their statistical properties are not as desirable as the MLE estimates. MOM and PPCC are relatively generic methods that can be applied in many different distributions. For some distributions where the MLE does not work well, there are specialized methods that are specific to that distribution. There are variations of the MOM (specifically, probability weighted moments and L-moments) that try to improve on the statistical properties of MOM estimates.

4.2.4 Evaluation

After identifying the best fit model for the data and estimating its parameters (see previous two sections), statisticians typically may apply a handful of statistical goodness of fit tests to determine if the selected distributional model is in fact an appropriate distributional model for the data. Generally there are three basic categories of goodness of fit tests [10, 137]:

1. The first class is based on comparing the empirical cumulative distribution function (CDF) [66] (i.e., based on the data) to the theoretical CDF function. This includes, but is not limited to, the Kolmogorov-Smirnov (KS) test, the Anderson-Darling (AD) test and the Cramer Von Mises test. The KS test is the original test of this class. It is based on the maximum distance between the empirical and theoretical CDF. The Anderson-Darling and related variants weight differences in the tails more than differences in the center of the distribution. Another popular method is the chi-square goodness of fit. However, this is most useful when dealing with pre-binned data. Scientists and engineers are often more familiar with the chi-square. Note that AD and KS are more powerful for unbinned data.

2. The second class is based on comparing the differences between the empirical percent point function (PPF) [5] to the theoretical percent point function. This category includes the probability plot correlation coefficient test (PPCC) [62].

3. The third class is based on the likelihood function. As the name "maximum likelihood" implies, this module searches for the distribution that provides the maximum value of the likelihood function. Although the rankings can be based on just using the likelihood value, it is more common to use "information critieria" [77]. The original statistic of this kind was Akaike's Information Criterion (AIC), whereas the Bayesian Information Criterion (BIC) seems to be more commonly used. In any case, these various information critieria modify the likelihood score based on the sample size and the number of parameters being fit. The basic idea is to provide a penalty for including more parameters (i.e., bias the ranking to simpler models where simpler in this case means fewer parameters).

4.3 Weaknesses of the traditional methodology

Distributional modeling can be used in several contexts. Statistical tests typically depend on certain assumptions regarding the underlying distribution (e.g., many tests are based on the assumption of normality). Appropriate distributional models are also used to more accurately assess uncertainty and, in particular, to assess the uncertainty over the full range of the data. Although a poorly chosen distributional model may suffice for measuring and assessing the uncertainty of averages, this will not be the case for tails of the distribution. In many applications (e.g., reliability), accurate assessment of the tail behavior is more critical than the average.

Most of the existing software used to determine the best fit model include plotting graphs and interpreting them. This is often subjective and depends on the analyst's perspective and statistical knowledge which may result in unfit model choices or mis-interpretation of some data points as outliers. Moreover, for datasets with a large number of observations, assessing the goodness of fit presents an issue. As an analogy, when computing the confidence interval for the difference between two means, this interval becomes increasingly small as the sample size gets large. That is, very small differences (i.e., not significant in a practical sense), will test as statistically significant. Similarly, for small samples, practically significant differences may not be statistically significant. Furthermore, for the Anderson-Darling test, a similar problem can occur. That is, for very large sample sizes, a distribution that is in fact providing a reasonable fit may well reject the distribution. Generally, statisticians address this by estimating selected percentiles (based on 5,000 to 10,000 points) from the data and apply probability plots and Anderson Darling goodness of fit on these percentiles. Unfortunately, this is a knowledge that not every analyst is familiar with as it needed experience and practise to learn it.

4.4 Automated techniques

Obtaining an appropriate distributional model can be an exhaustive process that takes time, patience and requires previous knowledge of statistics and is, therefore, a difficult task for some analysts. Therefore, there are several attempts to automate the distributional modeling process by creating software and packages that let the user know the "best" candidate distribution that matches their data. Generally, most

of them use traditional statistical techniques to pre-process data and run some goodness of fit tests in order to rank and identify a good representation of the data. Similar tools and packages include: Dataplot [49], R code [171], fitdistrplus [52], expertFit[125] and easy-fit [177], etc. All of them have the same goal which is to help analysts regardless of their knowledge of statistics, pin down the best candidate distribution matching their data and avoid using the wrong distribution while saving them time. Nevertheless, a good statistical background is still required to interpret the numerical and graphical outputs in these tools.

4.5 Conclusion

Probability distribution fitting of an unknown stochastic process is an important preliminary step for any further analysis in science or engineering. However, as seen in this chapter, it requires some background in statistics, prior considerations of the process or phenomenon under study and familiarity with several distributions. Although, there are several statistical tools that try to automate this process, they still require the users to pre-screen and analyze their data via some graphical tools for example. However, this often requires human interpretation which is subjective and changes depending on the statistical experience of the analyst.

In this chapter, we learned that identifying the best model for a dataset is actually the most sensitive and essential step in distributional modeling (i.e., exploratory analysis step in Figure 4.3). Once a good model is selected, the following two steps, i.e., probability estimation and evaluation, become easy. However, we also learned that the exploratory analysis step is subject to errors and misinterpretations because it requires human interaction for analyzing graphs and the results of numerical tests. This led us to investigate the use of deep learning as an alternative solution to automate this step.

Chapter 5

Neural Networks for Classifying Probability Distributions

In the previous chapters we stated the importance of distributional modeling in several applications in both science and engineering. It is also an essential step in our methodology introduced in chapter 2. However, we also examined a non exhaustive list of tools and techniques typically used by expert statisticians to conduct distribution fitting and showed the amount of work and statistical knowledge this task requires, to accurately asses and characterize data (step 1 and 2 from Figure 4.3).

This section presents an alternative approach of conducting distribution fitting which doesn't require prior knowledge of statistical methods nor previous assumption on the available data. Instead, using Deep Learning (DL), the best candidate distribution is extracted from the output of neural networks that have been previously trained on a large suitable database in order to classify an array of observations into a matching distributional model. We find that our trained neural networks can perform this task comparably to using well known statistical estimators and goodness-of-fit tests such as the maximum likelihood estimation with an Anderson-Darling goodness of fit test. We published the results of this study in [109].

5.1 Introduction

This research focuses on the use of deep learning (DL) to assist in identifying the best distributional model among a fixed set of candidate distributions (step 2 from Figure 4.3) in order to help analysts who are not equipped with sufficient statistical background easily map a set of empirical observations obtained from an experiment to an appropriate distributional model. Deep learning is a sub-field of machine learning that applies neural network architectures to learn features of the object to be classified. It has become more popular in recent years due to its high accuracy, its capacity to deal with massive data and larger neural networks as well as its capability to deduce data features automatically. Although DL neural networks take a longer duration to train and usually require high performance servers with graphic processing units (GPU) which are expensive, it is still considered an efficient and effective approach for object classification [43].

The approach proposed in this chapter exploits the strengths of deep learning for classification of distributional models. As an initial prototype, we restrict ourselves to the case of continuous measurements where the data is not binned, censored or truncated. Moreover, we do not consider pathological distributions (e.g., Cauchy distribution). In this chapter, we train a feed forward neural network to recognize patterns in an input data set then predict the "best" candidate model from nine commonly used distributions that are widely encountered in science and engineering.

The distributions considered in this study are: uniform, normal, logistic, exponential, half normal, half logistic, gumbel max, gumbel min and double exponential. The idea is to use deep learning as a replacement of the exploratory analysis step (step 2 according to Figure 4.3) that statisticians follow in order to identify the model that best suits the data. Then once the neural networks have identified the "best" distributional model, the parameter estimation and the goodness of fit assessment still need to be applied using traditional statistics. We demonstrate the validity of this study by considering a limited list of distributions at the moment. However, the examined distributions still cover the most encountered models. Moreover, this chapter and our published results [109] are intended to serve as a proof of concept to show the viability of our method. In the future, we plan to increase the number of distributions to cover families of distributions (i.e., with one or more shape parameters) and other use cases.

Other researchers in the literature investigated the use of neural networks for conducting parameter estimation of probability density functions which corresponds to step 3 from Figure 4.3. Similar papers include [173], [130] and [147]. In another group of related works, the authors focus on conditional density estimation using artificial intelligence such as [115] and [203]. Additionally, [174] presented the best practices for conditional density estimation for finance applications using neural networks. Finally, the authors in [41] used an ensemble of mixture density networks to predict the probability density function of the surf height in order to know if it will fall within a given 'surfable' range. However, there is limited research involving the use of neural networks to tackle step 2 of Figure 4.3 and create a classifier for distribution models based on a set of independent empirical observations. In fact, this task is the most important since once it is completed and the distribution model is identified, it becomes extremely easy to estimate the parameters of the distribution and formulate the probability density function (PDF).

In this chapter, we explain our suggested approach and evaluation metrics in section 5.2. The results of our analysis are shown in section 5.3.2. We also discuss the limitations of this work in section 5.4 and finally preview our ongoing and future research in Section 5.5.

5.2 Approach

This section illustrates our empirical study that aims to identify the best candidate distribution given an input dataset. We focus on nine uni-variate commonly used distributions in order to provide a working proof of concept and demonstrate the viability of the idea without covering all possible scoops (e.g., pathological distribution, mixture distribution,...). The goal of this research is not to completely replace the traditional distribution fitting workflow but rather suggest an alternative to the exploratory analysis step. As previously stated, exploratory analysis is a crucial step in determining the best candidate distribution that characterizes the data, however, it is also subject to the analyst's own interpretation as it requires analyzing graphical tools and running some numerical tests which typically need a good statistical background and knowledge of several commonly used distributions.

We explain how we collected data for training and validation purposes of the neural networks in subsection 5.2.1. Then, we describe the neural networks models in subsection 5.2.2.

5.2.1 Collecting data for training

Prior to generating any data, a few questions were brought up in the investigation phase:

(i) Which distributions to consider?

(ii) Since there are endless values for the location, scale and shape factors, how to select the right values for training?

(iii) Since data comes in different sizes, what sample sizes are most appropriate?

(iv) Should a pre-processing or normalization method be applied to the training data prior to passing it to the neural networks?

For the first question, we restrict ourselves to the case of continuous measurements where the data is not binned, censored nor truncated. Moreover, we do not consider pathological distribution (e.g., Cauchy distribution). In this chapter, we train a feedforward neural network to recognize nine probability distributions: uniform, normal, logistic, exponential, half normal, half logistic, gumbel max, gumbel min and double exponential. These are the typical options that statisticians initially consider when screening the data as they are the most common.

Next, as previously explained in chapter 4, probability distributions are characterized by three types of parameters: a location, a scale and one or more shape parameters. The location parameter shifts the distribution left or right on the x-axis, while the scale parameter compresses or stretches the distribution on the y-axis (chapter 4 shows examples of this for the normal distribution). The relationship between the probability density function of a general form of the distribution (i.e., location and scale not equal to zero and one) and its standard form (i.e., location equals to zero scale equals to one) is:

$$f(x; a, b) = \frac{f(\frac{(x-a)}{b}; 0, 1)}{b}$$

where a and b are the location and scale parameters,

Any parameter that is not either a location or scale is considered a shape factor. Shape parameters allow a distribution to take a variety of different shapes. By shape, we mean properties such as the skewness and kurtosis (i.e., peakedness). We've already established that families of distributions are out of the scope of this work, therefore we didn't have to concern ourselves with the shape factor. Nevertheless, data doesn't always come in the standard form of the distribution (i.e. location = 0 and scale = 1) which means that there are endless possibilities for the location and scale values per distribution. Thus, the choice for the location and scale must be justified. We explained in chapter 4 that the shape of the distribution does not depend on the value of the location and scale parameters as they have no effect on its actual silhouette. The location and scale factors don't change the properties of any distribution (i.e., skewness, peakness, the tails, etc). Thus, we utilized the standard form of the distributions when collecting data for training the neural networks (answer to question (ii)).

Even though, probability distributions are typically defined in terms of the probability density function (pdf), there are a number of other probability functions used in applications (for more details please see [5]). Examples include:

1. Probability density functions;

2. Cumulative distribution function;
3. Percent Point Function;
4. Hazard Function;
5. Cumulative Hazard Function;
6. Survival Function;
7. Inverse Survival Function.

In this study, we used the kernel density plot (kdp) [104] as input to the neural networks classifiers. The kdp is a graphical estimate of the underlying probability density function and is considered a typical technique by statisticians to conduct exploratory analysis of datasets in order to identify the shape of the underlying distribution. We generated kdp for each set of random numbers with the Dataplot software [63] and used the Silverman recommendation for the smoothing parameter h [184]. The kernel density estimate, $f_n(x)$, of a set of n points from a density f is defined as:

$$f_n(x) = \frac{\sum_{j=1}^{n} K\{\frac{(x-X_j)}{h}\}}{nh}$$

where K is the kernel function and h is the smoothing parameter or window width. The Silverman algorithm uses a Gaussian kernel function. This down weights points smoothly as the distance from x increases. We used the Silverman's default recommendation for the h parameter:

$$0.9\min(s, \mathrm{IQ}/1.34)n^{-1/5}$$

with s, IQ, and n denoting the sample standard deviation, the sample interquartile range and the sample size, respectively.

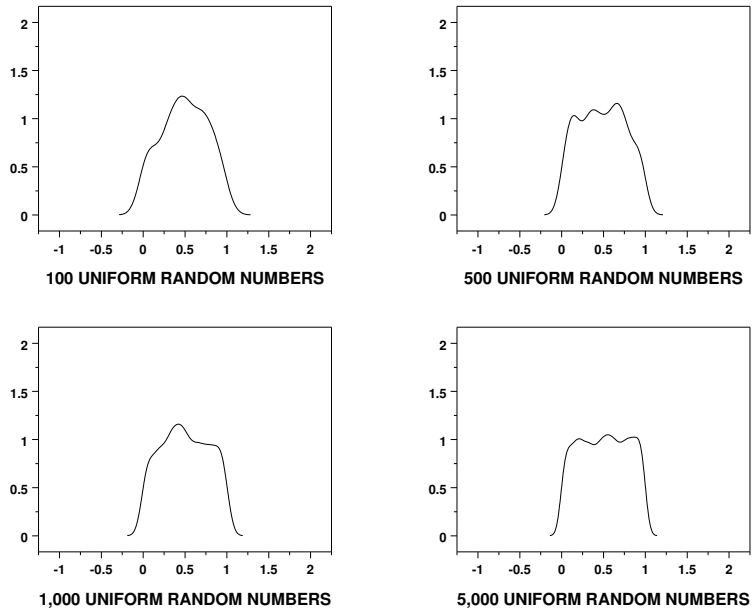

FIGURE 5.1: Uniform kdp based on the sample size

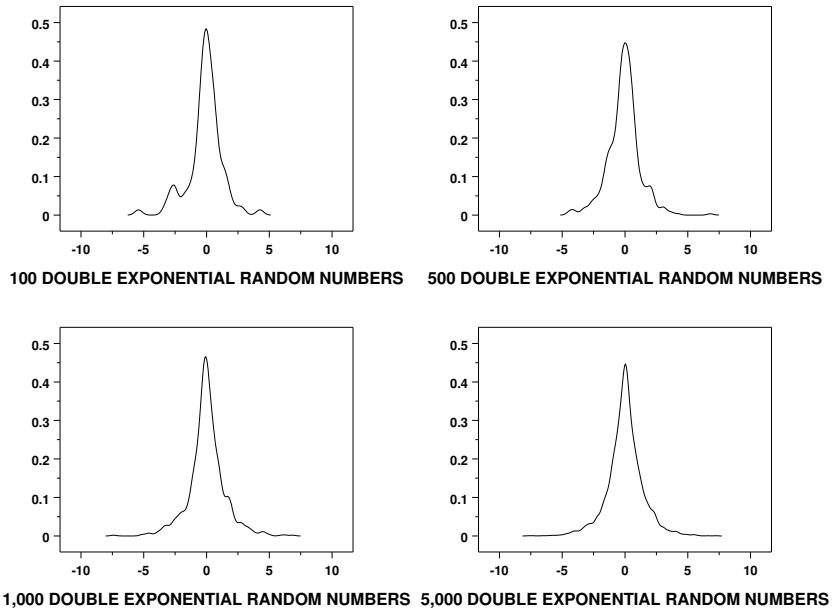

FIGURE 5.2: Double exponential kdp based on the sample size

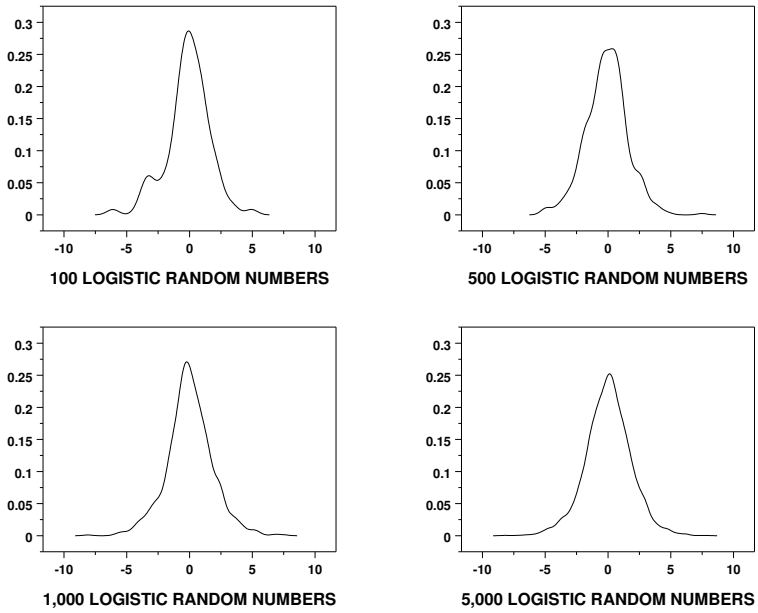

FIGURE 5.3: Logistic kdp based on the sample size

The choices of the sample size n to use per dataset was another concern in the beginning of this study. For this, we tested several sample sizes ranging from small (n = 30) to large (n = 10,000) as we tried to determine which ones were suitable. Eventually, we selected the following sizes 30, 50, 100, 250, 500, 750, 1,000 and 10,000 (answer to question (iii)). This choice may seem arbitrary, however it is based on the fact that the smoothness of the kernel density plot increases with the increase of the number of available data points. Thus, the chances that the neural networks accurately guesses the right distributional model would be higher. Figures 5.1, 5.2

and 5.3 show the impact of the sample size on the kernel density plot of the uniform, the double exponential and logistic distributions, respectively.

For each of the nine distributions considered in this study, 10,000 datasets were generated in the standard forms and at different sizes (30, 50, 100, 250, 500, 750, 1,000 and 10,000). The random numbers were generated with the Dataplot software [63] and a congruential-Fibonnaci [100] generator that was used with a different seed for each distribution/sample size configuration. For each set of the (distribution, sample size) pair, the kernel density plot was sampled to generate 256 points. These 256 points constitute the inputs for the neural networks as the y-axis coordinates of the kernel density plot with an implicit x-axis coordinates $X = 1, 2, ..., 256$.

5.2.2 Training the neural networks

An initial attempt at producing a single solid neural network model to classify an arbitrary number of data points (n) to a matching probability density function (pdf) has not yielded promising results especially when the sample size n is small. The intuition behind the misclassification could be interpreted as follows: our approach relies on building a kernel density estimator (kdp) from a set of independent empirical observations. This kdp tends to be noisy for small n and becomes increasingly smooth as n becomes larger. Thus, models trained on the larger sample sizes perform poorly on the smaller sample sizes and models trained on smaller sample sizes perform poorly for larger sample sizes.

To improve the performance, we consider 20+ models. The data collected is generated using eight sample sizes: n=30, n=50, n=100, n=250, n=500, n=750, n=1,000 and n=10,000 and each model is trained on a specific sample size range. The idea is to evaluate the models individually and collectively to deduce which ones work best for small, moderate and large sample sizes. Examples of the considered models include model 1: $n \in [30, 100]$, model 2: $n \in [100, 750]$, model 3: $n \in [750, 10000]$, etc. All the models have the same input and output layers. However, their hidden layers differ in size and width. The input layer has 256 units representing the Y-Axis coordinates for the kernel density plot whereas the output layer has 9 points which refers to the one hot encoding of the nine distributional models considered in this study. For each interval, we started with a very simple FeedForward neural network (FNN) that overfits. We then proceeded to handle the overfiting by tuning the FNN parameters to achieve the lowest loss and highest accuracy. We found that the following work best:

1. 20% of the training data was allocated for the validation;
2. All models use Softmax as the activation function for the output layer and Relu for the hidden layers;
3. The choice of the loss function was Categorical cross-entropy (CAT) for larger intervals and Mean squared error (MSE) for smaller intervals;
4. The ADAM optimizer was used when the loss function was set to CAT and RMSprop for MSE;
5. The learning rate is set to 10^{-6} or 10^{-5} in most cases;
6. The batch size is set to 200 for most models;
7. Each model was run for an average of 500 epochs;

8. The weights were initialized using the 'He uniform' distribution;
9. The bias was enabled in the hidden layers and disabled in the output layer;
10. The depth of each neural networks model was 40, while the width was either 512 or 1024 nodes per layer;
11. Early stopping was deployed;
12. Regularization and dropout were used.

The models were all implemented using Python and Keras with Tensorflow as a backend and the experiment was run on our testbed with 1 GPU and 40 CPU cores. Figures 5.4 to 5.7 show the accuracy and loss plots of the two best performing neural network models.

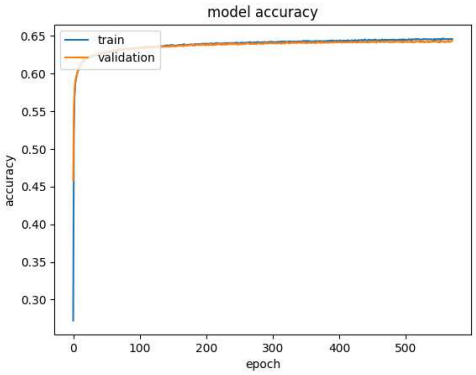

FIGURE 5.4: Accuracy plot for the model trained on smaller sizes

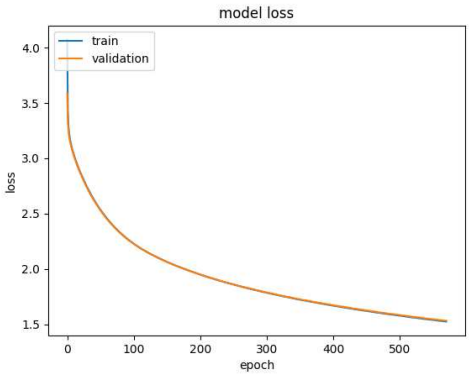

FIGURE 5.5: Loss plot for the model trained on smaller sizes

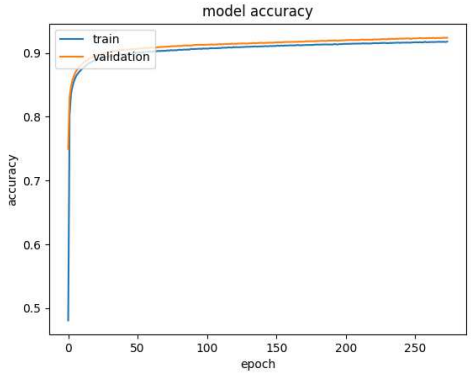

FIGURE 5.6: Accuracy plot for the model trained on larger sizes

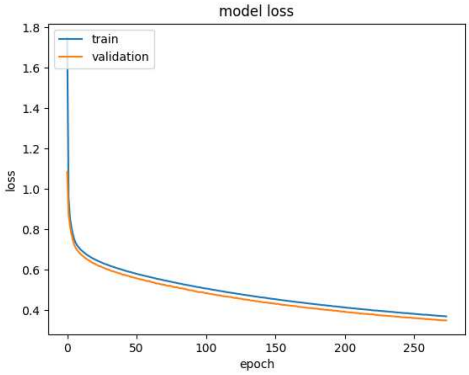

FIGURE 5.7: Loss plot for the model trained on larger sizes

5.3 Evaluation

After the training and validation steps of the neural networks were complete, additional datasets were used to test the best performing models. Furthermore, to

determining the viability and effectiveness of our approach, the models' accuracy results were also compared against the results of a conventional statistical approach as follows:

- The data is fit to each distribution using the maximum likelihood (MLE). The one exception is that the half-logistic distribution is fit using the method of moments.
- After estimating the parameters with the maximum likelihood, the distributions are ranked based on the Anderson-Darling (AD) goodness of fit statistic [188]. The AD test is a refinement of the Kolmogorov-Smirnov (KS) statistic that puts more weight in the tails of the distribution. The AD test is generally considered to have more power than the KS test (chapter 4).

  Given an ordered set of data points Y_i of size N and a cumulative distribution function F, the Anderson-Darling test statistic is defined as:

$$A^2 = -N - \sum_{i=1}^{N} \frac{(2i-1)}{N} [\ln F(Y_i) + \ln (1 - F(Y_{N+1-i}))]$$

There are a variety of estimation methods and goodness of fit statistics that could be used for this approach. However, the combination of MLE estimation and ranking by the AD goodness of fit test provides a reasonable benchmark for assessing the results of the neural networks. Moreover, it is important to mention that both approaches (neural networks and MLE-AD) were compared based on the same data.

5.3.1 Preparing the testing data

Typically, real world data might have location and scale values that are not in the standard form (i.e., location = 0 and scale = 1). For this reason, we generated random numbers for each distribution with different location and scale values. Specifically, datasets were generated with sample sizes of 50, 100, 250, 500, 750, 1,000 and 10,000. For each sample size, datasets were generated with location values: 20, 60 and 100 and scale values: 10, 30 and 50. This adds up to a total of nine different combinations of location and scale values per (distribution, sample size) pair. Then, we generated 1,000 datasets for each distribution/sample size/location/scale combination. Additionally, as with the training data, a kernel density plot was generated for each dataset with the same algorithm used for creating the kernel density plots in the training sets.

One question of interest is whether an appropriate normalization method can address the issues introduced by the location and scale parameters. We recall that the training and the validation sets were generated for the standard forms of the distributions (location parameter = 0, scale parameter = 1) while the testing data was generated with non-standard values of the location and scale parameters. The y-coordinates (height) of the kernel density plots are used as inputs to the neural networks, which gives an implicit x-coordinate scale of 1 to 256. Since the location parameter mainly shifts the kdp right or left, using an implicit x-axis scale of 1 to 256 for both the training and the testing data should minimize the effects of the location parameter. However, the scale parameter stretches the kdp across the y-axis which ultimately changes the height of the kernel density plot. Therefore, there is a need to transform the kernel density heights so that the testing data can be more effectively compared to the training data.

To determine which normalization technique is best suited for this case, we experimented with several transformation algorithms on both the training, the validation and the testing data sets. But, only two yielded promising results:

1. The U-score, also referred to as the Min-Max normalization. The u-score algorithm transforms the kernel density heights to a (0,1) scale according to the following mathematical formulation:

$$u\_score = \frac{x - min(x)}{max(x)) - min(x)}$$

where x is the original value, u_score is the normalized value, $min(x)$ and $max(x)$ are respectively the minimum and maximum values of each dataset x.

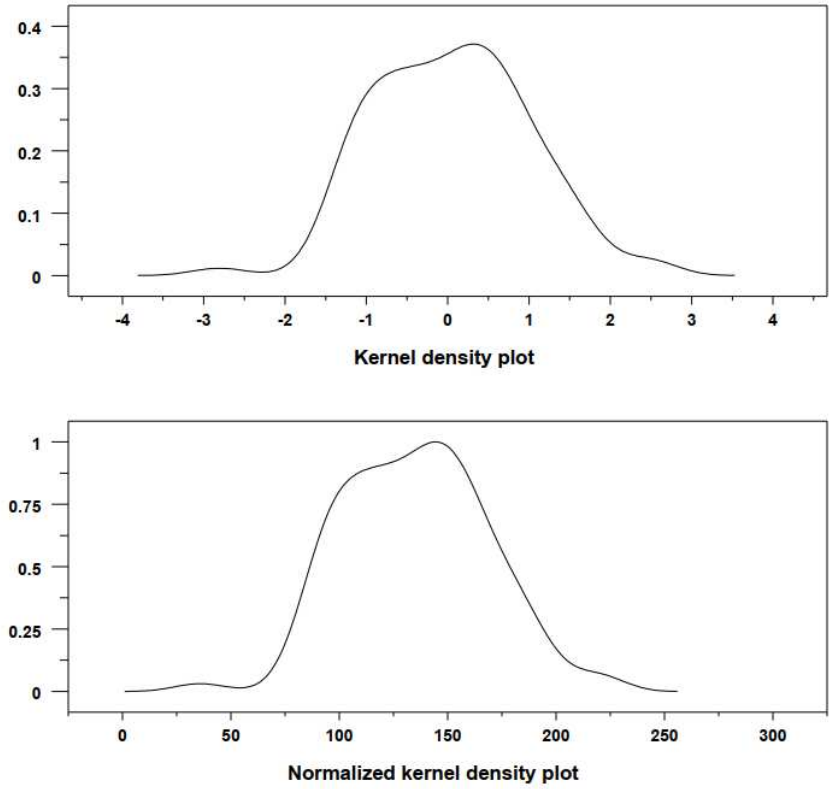

FIGURE 5.8: U-score normalization

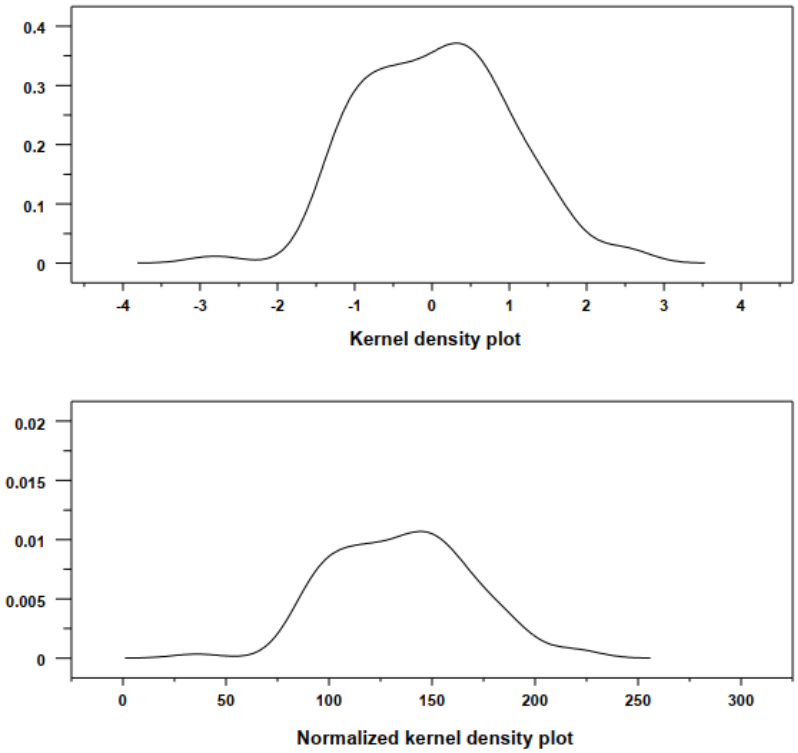

FIGURE 5.9: kernel density normalization

This normalization preserves the shape of the original distribution and does not meaningfully change the silhouette of the kernel density plot nor does it reduce the importance of outliers. Figure 5.8 shows the kernel density plot of 100 normal random numbers before and after applying this normalization.

2. The kernel density normalization transforms the kernel density heights to integrate to 1 on the 1 to 256 x-coordinate scale:

$$k\_score = \frac{x}{\sum_{i=1}^{256} x_i}$$

where x is the original value and k_score is the normalized value.

Figure 5.9 shows the kernel density plot of 100 normal random numbers before and after applying the kernel density normalization.

After careful considerations, we found that these two normalization techniques are non-distorting of the shape of the kernel density plot and preserve the form of the probability distribution regardless of the location and scale values.

5.3.2 Results

Our primary metric of success was the percentage of times that the correct distribution was accurately identified by a neural networks model. For the mis-classified cases, we also identify which distributions were chosen instead of the correct ones. The following factors are examined while analyzing the results:

Factor 1: There are two different normalization algorithms considered: the u-score and the kernel density normalization (subsection 5.3.1);

Factor 2: There were eight different sample sizes used for the testing datasets. We grouped these into three categories: "small", that is datasets with 30 or 50 or 100 observations; "moderate", that is datasets with 100, 250, 500 or 750 observations; and "large", that is datasets with 750, 1,000 or 10,000 observations. Note that these categories contain overlaps in order to create three intervals: small $[30, 100]$, moderate $[100, 750]$ and large $[750, 10000]$. These intervals are useful because real world data is not confined to these particular eight sizes;

Factor 3: This study considered nine distinct distributions. These distributions allow for location and scale parameters, but none of them have shape parameters;

Factor 4: There were 20+ NN models considered;

Factor 5: There were nine combinations of location/scale parameters for each (distribution, sample size) pair.

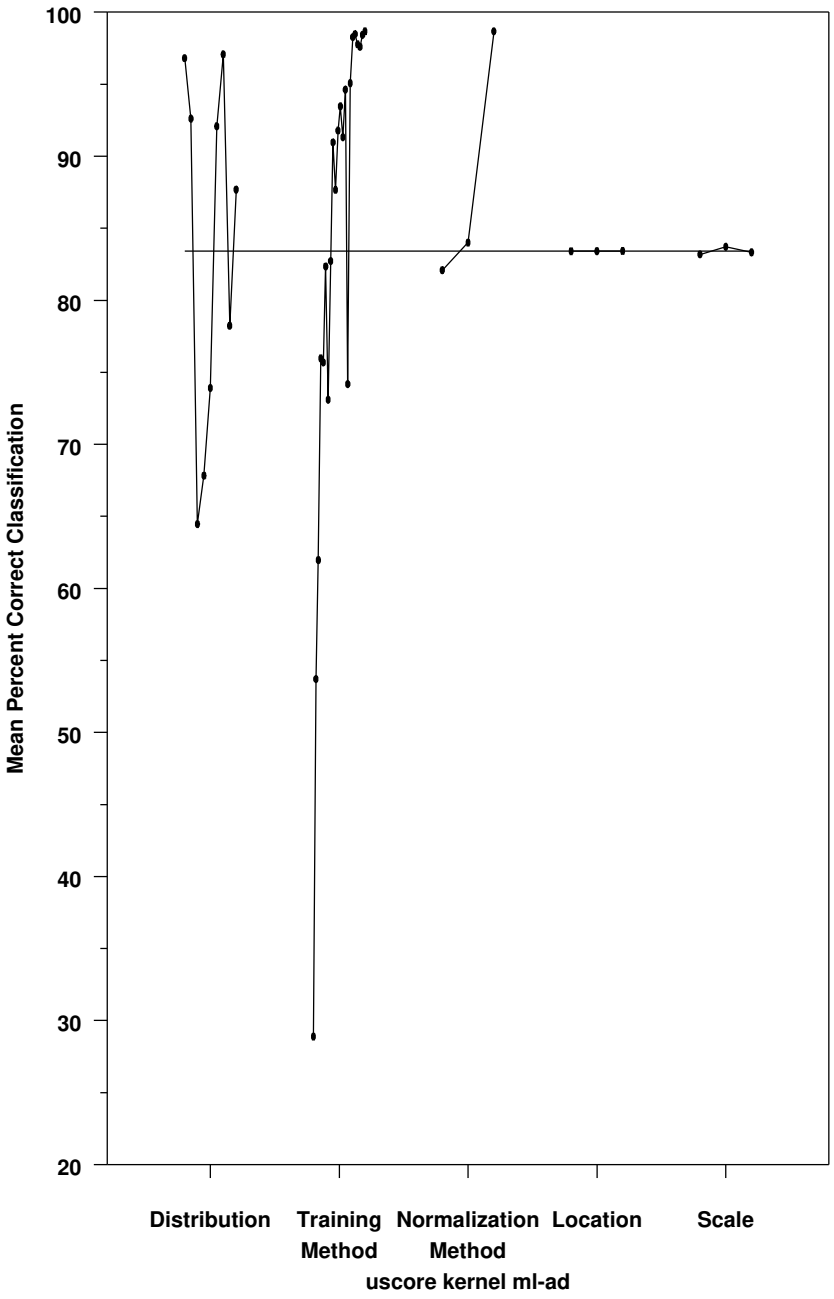

FIGURE 5.10: DEX mean plot for large sample sizes - all NN models

As a first step, we generated Design of Experiments (DEX) mean plots [61] as shown in Figures 5.10 to 5.15. The DEX mean plot is a useful graphical tool for showing the most important factors from an experiment. This plot typically answers the following two questions:

1. Which factors are clearly important and which factors are clearly or borderline not important?

2. What is the ranking list of the important factors?

In a DEX mean plot, the horizontal axis represents all the considered factors, whereas the vertical axis indicates the mean of the response variable for each level of the factors. The means for a single factor are connected by a straight line which is used to measure their importance (i.e., the longer the vertical line the higher the importance). Refer to [2] for more information on this type of plots.

We generated separate DEX plots for the large (Figure 5.10), moderate (Figure 5.11) and small (Figure 5.12) sample sizes for all of the 20+ models investigated. For these DEX mean plots, the y-axis represents the mean percent correct classification of the distributions and the x-axis represents the five variables in question:

1. Probability distributions;[1]
2. The trained neural networks model (out of the 20+);
3. The normalization method (u-score, k-score);
4. The location parameter values;
5. The scale parameter values.

From these DEX plots, we notice that the response variables for both the location and scale factors almost form horizontal lines which leads us to believe that they have very little effect on the accuracy of the classification of the probability distributions. However, we can't successfully identify the best performing model from these plots. Therefore, we selected the most effective NN models based on their testing accuracy rates and plotted a subset DEX graphs for each sample size category (Figures 5.13, 5.14 and 5.15). Some initial conclusions from these plots include:

[1] uniform, 2: normal, 3: logistic, 4: exponential, 5: double-exponential, 6: half-normal, 7: half-logistic, 8: gumbel-min, 9: gumbel-max

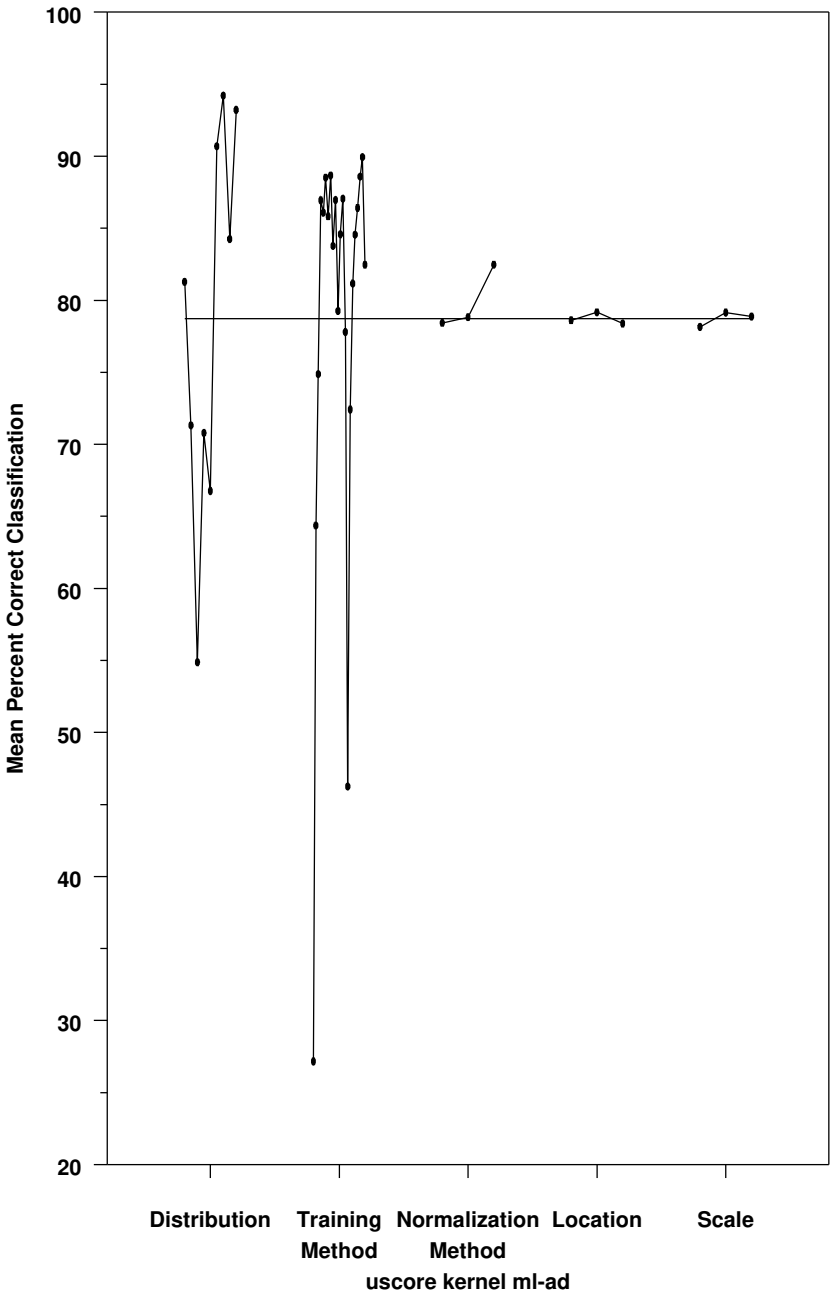

FIGURE 5.11: DEX mean plot for moderate sample sizes - all NN models

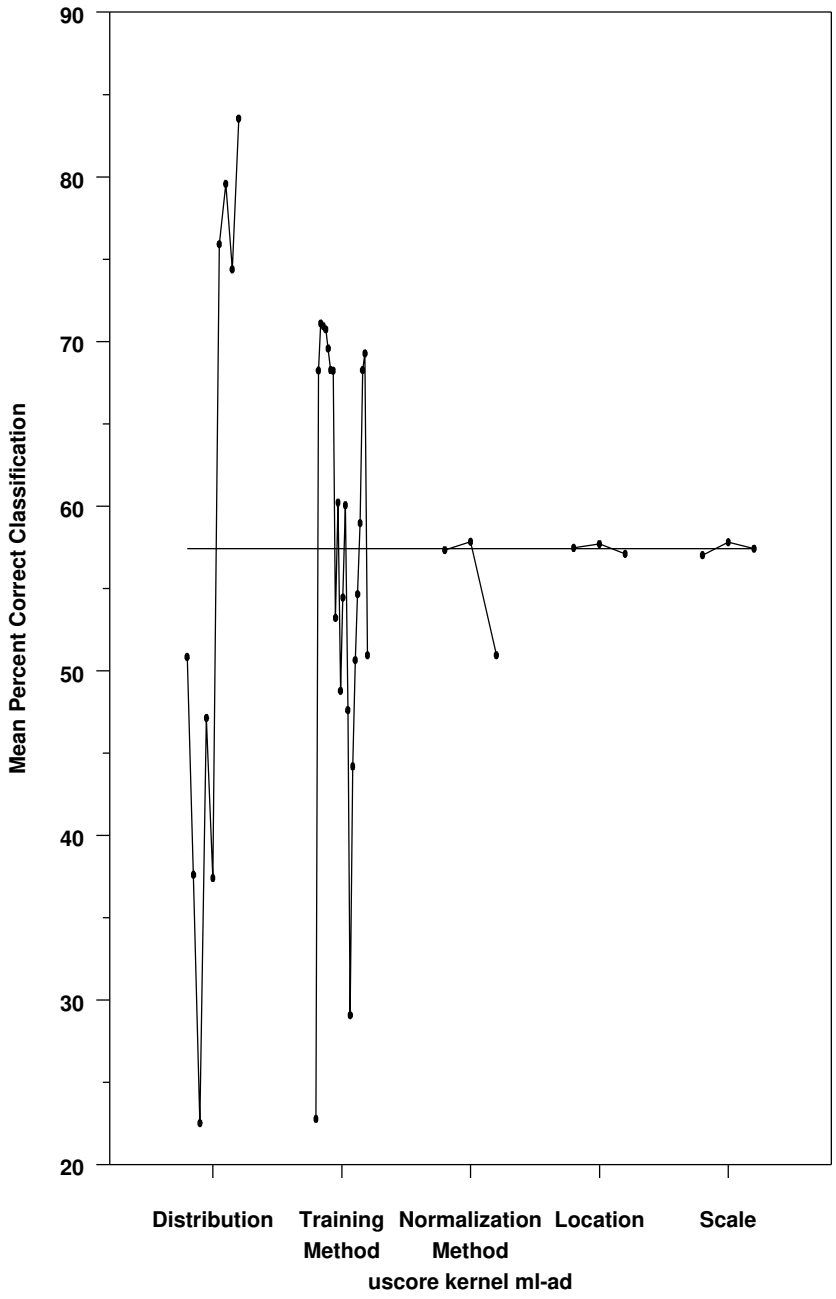

FIGURE 5.12: DEX mean plot for small sample sizes - all NN models

- When we look at the location/scale factors for the plots that only include the best performing training models, we confirm that the effect is indeed negligible. For this reason, in subsequent analysis, the data for all nine location/scale combinations are aggregated into a single value.
- The u-score and kernel density normalization methods have similar performance.

- For each sample size category, there is significant variability in the performance of the different training models. The best training models are different for the three sample size categories, but for a given sample size category there are several training models that have similar performances. According to Figures 5.13, 5.14 and 5.15, we highlight the following best three models: [2]:

  1. For moderate and large sample sizes, we choose the training model '100-250-500-750-1,000-10,000' (($N \in [100, 10{,}000]$) and the kernel normalization. Tables 5.1 and 5.2 show the confusion matrix for this model compared against MLE-AD for the moderate and the large categories (rounded to two decimal places) and Figures 5.6-5.7 indicate the accuracy and loss plots for this neural networks model, respectively.
  2. For a small sample size, we choose the training model '30-50-100' (($N \in [30, 100]$) and the u-score normalization. Table 5.3 shows the confusion matrix for this model compared against MLE-AD (rounded to two decimal places) and Figures 5.4-5.5 indicate the accuracy and loss plots for this neural networks model, respectively.

- There are performance differences between the distributions. Specifically, the half-logistic and and the logistic have significantly poorer performance than the other distributions. This is not surprising as these have similar shapes to the half-normal and normal distributions, respectively.

- As expected, performance improves as the sample size increases. According to the DEX mean plots, for the small category, the overall performance was approximately 70%, for the moderate category the overall performance was close to 85%, and for the large category the overall performance was about 98%. The latter means that with a large number of observations, our NN can identify "the correct" distribution.

[2] uniform, 2: normal, 3: logistic, 4: exponential, 5: double-exponential, 6: half-normal, 7: half-logistic, 8: gumbel-min, 9: gumbel-max

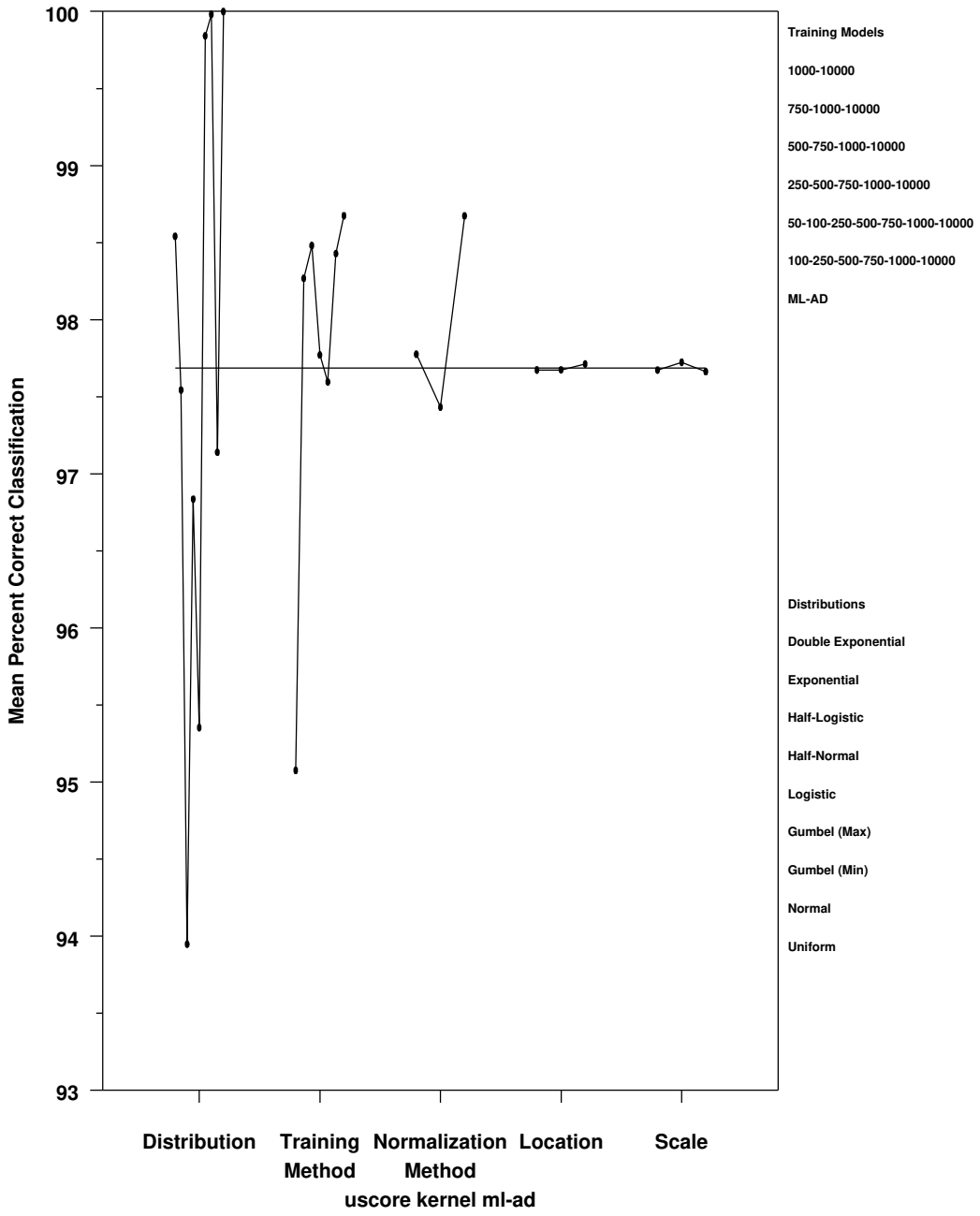

FIGURE 5.13: DEX mean plot for large sample sizes - best performing NN models

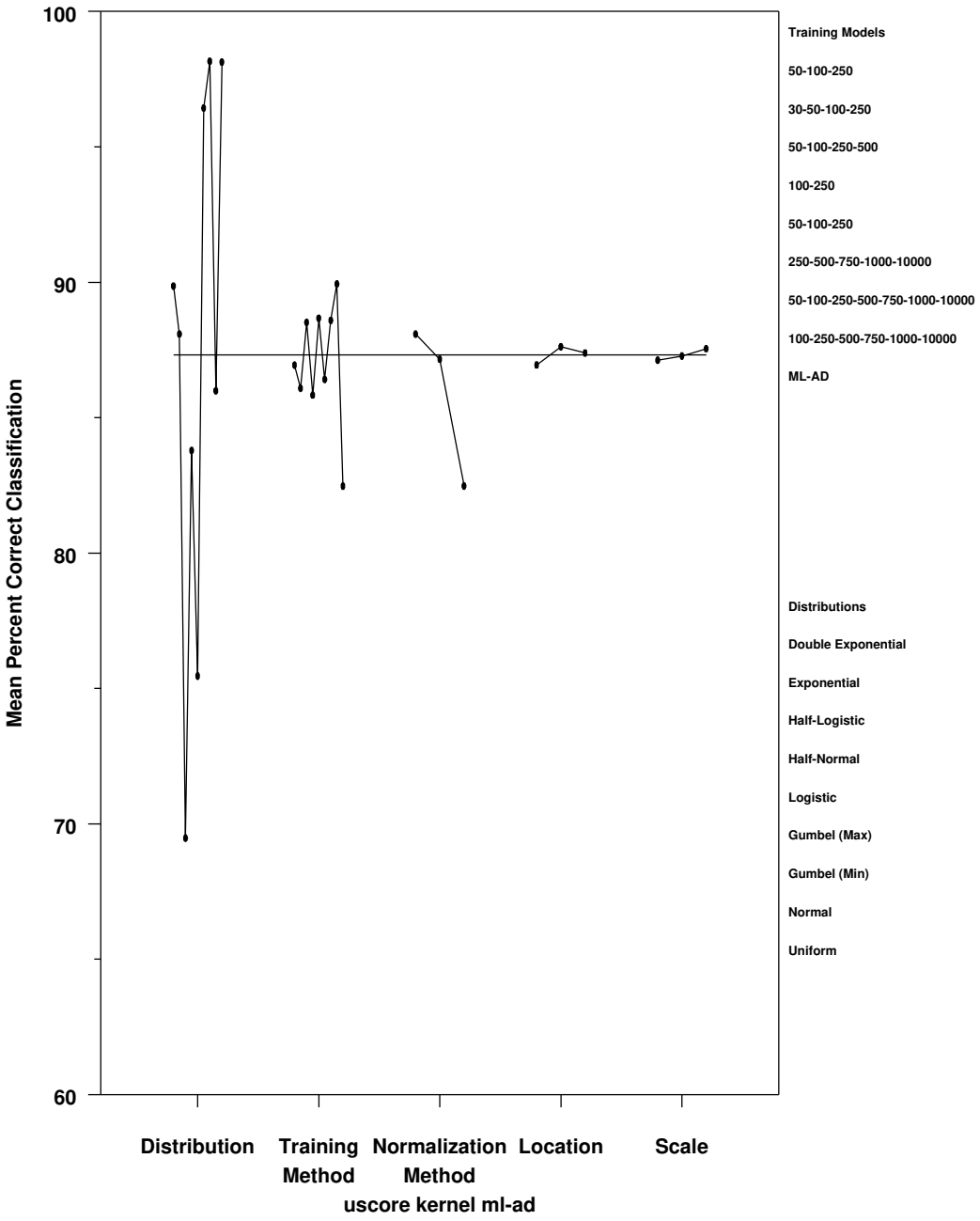

FIGURE 5.14: DEX mean plot for moderate sample sizes - best performing NN models

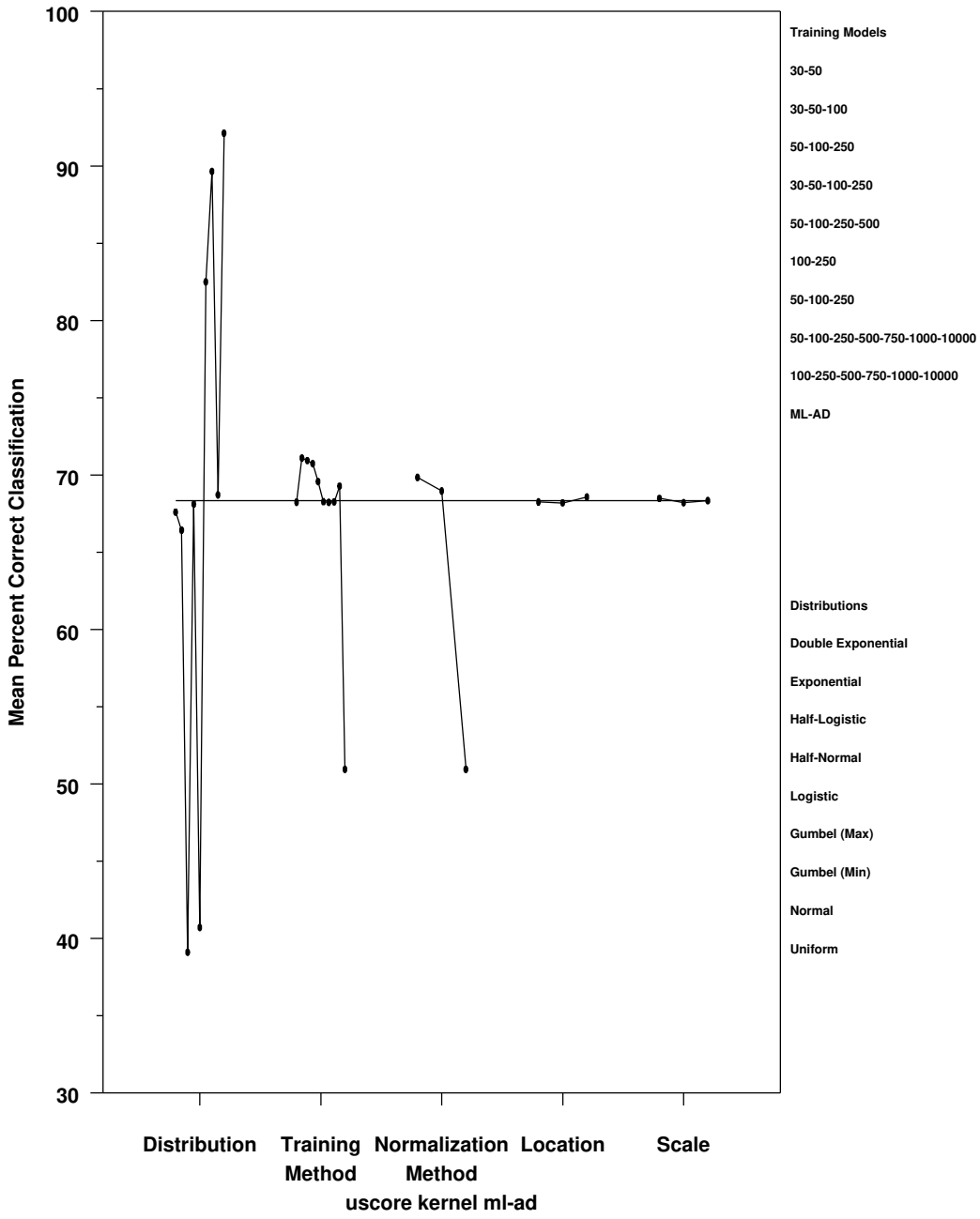

FIGURE 5.15: DEX mean plot for small sample sizes - best performing NN models

Furthermore, we also generated a tabulation chart in figure 5.16. This figure simply shows the response variable (i.e., correct classification percentage) for the selected few subset models.

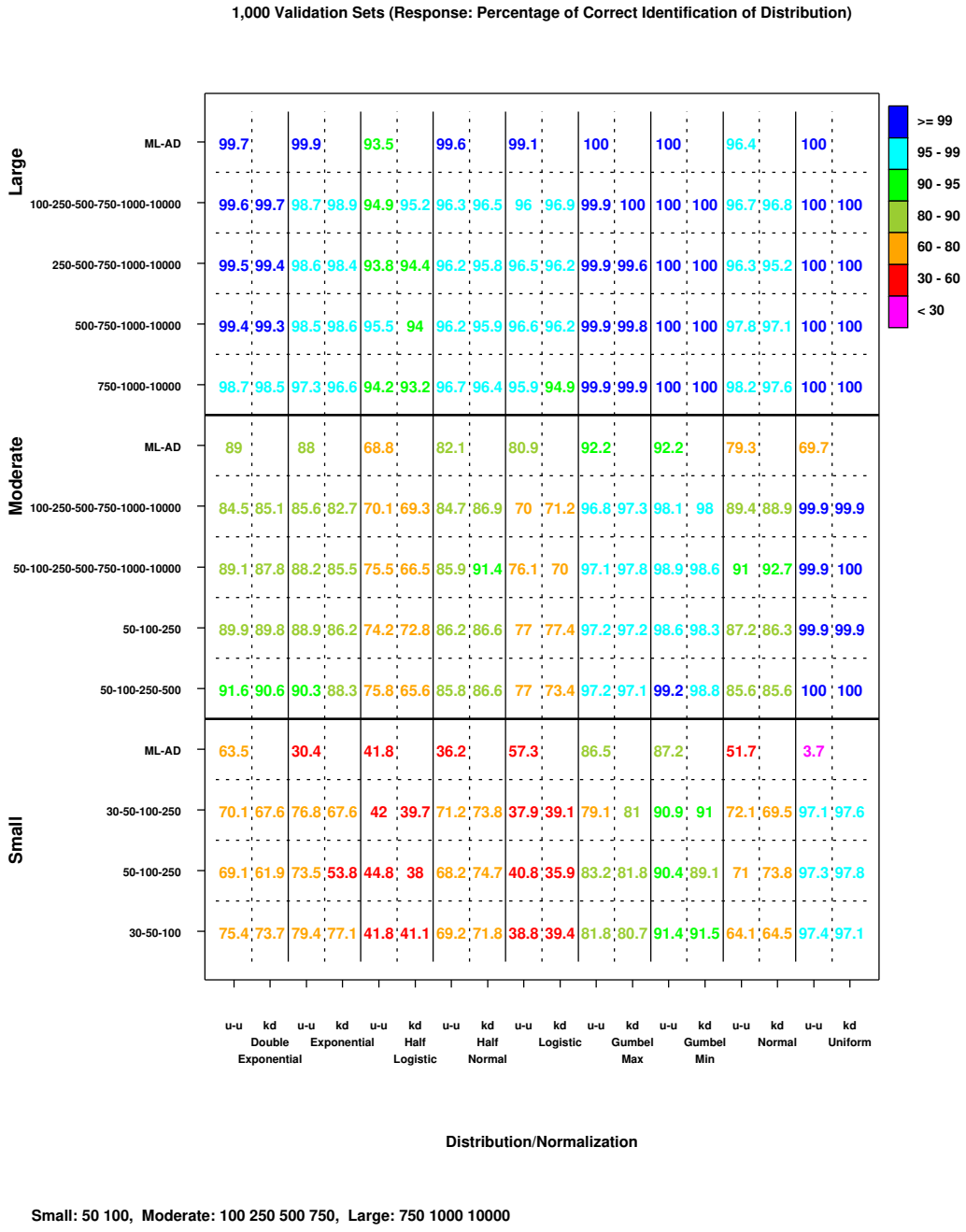

FIGURE 5.16: Tabulation chart

For evaluation purposes, we also provide confusion matrices for MLE-AD in the same tables as the output of the neural networks per sample size category (tables 5.1, 5.2 and 5.3). Theses tables indicate that our neural networks perform comparably to MLE-AD and give better performance in a majority, but not all, of the cases than the MLE-AD method. In fact:

True Distribution	Approach	Selected distribution								
		1	2	3	4	5	6	7	8	9
Uniform	NN	100	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	MLE-AD	99.99	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Normal	NN	0.00	96.77	3.20	0.00	0.00	0.00	0.00	0.01	0.02
	MLE-AD	0.00	96.38	3.62	0.00	0.00	0.00	0.00	0.00	0.00
Logistic	NN	0.00	1.87	96.94	0.00	1.13	0.00	0.00	0.03	0.03
	MLE-AD	0.00	0.79	99.08	0.00	0.12	0.00	0.00	0.00	0.01
Exponential	NN	0.00	0.00	0.00	98.93	0.00	0.00	1.07	0.00	0.00
	MLE-AD	0.00	0.00	0.00	99.93	0.00	0.00	0.06	0.01	0.00
Double Exponential	NN	0.00	0.00	0.30	0.00	99.70	0.00	0.00	0.00	0.00
	MLE-AD	0.00	0.00	0.34	0.00	99.64	0.00	0.00	0.01	0.00
Half Normal	NN	0.00	0.00	0.00	0.00	0.00	96.45	3.55	0.00	0.00
	MLE-AD	0.00	0.00	0.00	0.00	0.00	99.57	0.34	0.00	0.09
Half Logistic	NN	0.00	0.00	0.00	2.32	0.00	2.47	95.21	0.00	0.00
	MLE-AD	0.00	0.00	0.00	0.81	0.00	5.70	93.48	0.00	0.01
Gumbel Min	NN	0.00	0.00	0.00	0.00	0.00	0.00	0.00	100	0.00
	MLE-AD	0.00	0.00	0.00	0.00	0.01	0.00	0.00	99.99	0.00
Gumbel Max	NN	0.00	0.00	0.00	0.01	0.00	0.00	0.01	0.00	99.98
	MLE-AD	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	99.99

TABLE 5.1: Confusion matrix for the large category: Neural Networks (NN) vs maximum likelihood/Anderson-Darling (MLE-AD)

True Distribution	Approach	Selected distribution								
		1	2	3	4	5	6	7	8	9
Uniform	NN	99.95	0.03	0.00	0.00	0.00	0.00	0.00	0.02	0.01
	MLE-AD	73.78	15.48	0.07	0.00	0.00	0.94	0.00	5.30	4.43
Normal	NN	0.10	91.01	7.08	0.00	0.34	0.00	0.00	0.74	0.72
	MLE-AD	0.00	79.27	18.55	0.00	0.38	0.00	0.00	0.93	0.86
Logistic	NN	0.01	16.42	77.15	0.00	4.44	0.00	0.00	0.96	1.03
	MLE-AD	0.00	8.71	85.67	0.00	4.52	0.00	0.00	0.54	0.55
Exponential	NN	0.00	0.00	0.00	86.62	0.00	1.30	12.07	0.00	0.00
	MLE-AD	0.00	0.00	0.00	87.98	0.00	0.43	10.89	0.00	0.70
Double Exponential	NN	0.00	0.53	10.19	0.00	88.66	0.00	0.00	0.32	0.31
	MLE-AD	0.00	0.09	10.57	0.00	89.04	0.00	0.00	0.13	0.17
Half Normal	NN	0.03	0.00	0.00	0.29	0.00	88.85	9.24	0.00	1.58
	MLE-AD	0.00	0.24	0.10	0.00	0.00	87.50	3.00	0.00	9.15
Half Logistic	NN	0.00	0.00	0.00	8.55	0.00	15.93	74.96	0.00	0.57
	MLE-AD	0.00	0.01	0.01	5.18	0.00	20.69	68.84	0.00	5.27
Gumbel Min	NN	0.03	1.02	0.39	0.00	0.07	0.00	0.00	98.49	0.00
	MLE-AD	0.00	0.59	1.60	0.00	0.20	0.00	0.00	97.62	0.00
Gumbel Max	NN	0.02	0.85	0.27	0.03	0.06	0.57	0.26	0.00	97.94
	MLE-AD	0.00	0.54	1.46	0.00	0.20	0.13	0.01	0.00	97.66

TABLE 5.2: Confusion matrix for the moderate category: Neural Networks (NN) vs maximum likelihood/Anderson-Darling (MLE-AD)

- For the small category, NN outperforms MLE-AD for 6 out of 9 distributions and they perform essentially the same for the half-logistic distribution;
- For the moderate category, NN outperforms MLE-AD for 6 out of the 9 distributions and they perform essentially the same for the Gumbel max distribution;
- For the large category, ML-AD performs slightly better for 3 distributions, NN performs slightly better for one distribution and for the remaining 5 distributions they perform essentially the same.

5.4 Limitation

In this study we proposed the use of deep learning to build a classifier for distributional modeling. This classifier takes as input a set of data points and provides a

True Distribution	Approach	Selected distribution								
		1	2	3	4	5	6	7	8	9
Uniform	NN	**97.44**	0.89	0.00	0.01	0.00	0.73	0.02	0.58	0.33
	MLE-AD	**3.72**	52.00	0.64	0.00	0.00	2.68	0.01	21.72	19.24
Normal	NN	1.99	**64.13**	17.94	0.00	4.04	0.46	0.00	5.89	5.54
	MLE-AD	0.00	**51.72**	31.96	0.00	3.07	0.00	0.00	6.81	6.44
Logistic	NN	0.37	28.42	**38.84**	0.00	19.99	0.20	0.00	6.14	6.03
	MLE-AD	0.00	19.74	**57.32**	0.00	13.28	0.00	0.00	4.81	4.86
Exponential	NN	0.02	0.00	0.00	**79.29**	0.00	5.96	14.58	0.00	0.16
	MLE-AD	0.00	0.11	0.09	**30.44**	0.00	1.58	51.80	0.00	15.98
Double Exponential	NN	0.03	4.41	14.17	0.01	**75.39**	0.03	0.01	2.96	2.98
	MLE-AD	0.00	1.51	30.37	0.00	**63.49**	0.00	0.00	2.18	2.45
Half Normal	NN	1.04	0.22	0.01	4.67	0.01	**69.24**	19.06	0.00	5.76
	MLE-AD	0.00	2.80	1.22	0.00	0.08	**36.36**	13.61	0.02	45.91
Half Logistic	NN	0.13	0.01	0.01	24.99	0.00	29.99	**41.84**	0.00	3.03
	MLE-AD	0.00	0.49	0.39	3.79	0.03	19.34	**41.84**	0.01	34.10
Gumbel Min	NN	0.70	4.82	1.65	0.00	1.43	0.00	0.00	**91.39**	0.01
	MLE-AD	0.00	4.29	6.81	0.00	1.67	0.00	0.00	**87.21**	0.03
Gumbel Max	NN	0.68	4.57	1.58	0.21	1.40	6.55	3.14	0.02	**81.84**
	MLE-AD	0.00	4.07	6.44	0.00	1.74	0.69	0.46	0.05	**86.54**

TABLE 5.3: Confusion matrix for the small category: Neural Networks (NN) vs maximum likelihood/Anderson-Darling (MLE-AD)

distribution label that matches one of nine most common distributions.

This approach is not a complete replacement of the traditional statistical workflow that statisticians follow to analyze and fit the data. However, it is an alternative to step 2 from figure 4.3 (i.e., exploratory analysis) which usually requires a good background of statistical knowledge as well as a familiarity with several distributions to be able to recognize a good potential distribution from a set of empirical observations. This is typically done via histograms or kernel density plots to help pin down the basic shape of the underlying distribution and find properties such as the skewness and the presence of multiple modes in the data. Moreover, this paper considers uni-variate non-censored and non-truncated data and doesn't consider families of distributions (with one or more shape parameters) nor noisy data (which is generally a mixture distributions).

In this research, We considered a limited number of distributions that correspond to the most commonly used models that are widely encountered. The reason behind our decision is that we hope to provide an initial working prototype that can prove the viability and applicability of our methodology.

5.5 Future work

In future work, we will extend the training set beyond the nine currently supported distributions. In particular, this will include commonly used families of distributions such as the weibull, lognormal and gamma distributions. These families can generate a variety of shapes based on the value of their shape factors. For this reason, we plan to incorporate the ability to make more specific classifications (e.g., distinguish between a weibull or a lognormal distribution) and compare this to approaches such as the likelihood ratio test [56], [55].

Furthermore, we developed a tool that automates the distributional fitting process for uncensored and unbinned uni-variate data which deploys our trained neural networks to identify the best candidate model from the distributions presented in this study. This tool takes empirical observation of any size, computes the kernel

density estimation on behalf of the user, then runs the corresponding neural networks classifier to predict the best fit model. We named this tool DeepFit and we will discuss its architecture and design in chapter 6.

Additionally, DeepFit incorporates a handful of statistical tests to estimate the parameters of the fitted distribution and assess its goodness of fit. We compare the predictions to the results of the Anderson Darling, the Kolmogorov-Smirnov and the probability plot correlation coefficient tests as well as the information criteria (AIC, BIC).

Moreover, we include in the tool an interactive module to help the users screen their dataset, pre-process it by selecting a normalization technique before starting the neural networks classifier. This module will contain a step by step guide to help users identify and eventually remove outliers prior to running the neural networks classification. This step is very important because identifying bad data in the sense of being erroneous (e.g., data is mis-coded or there is an assignable cause for why the observation is in error) could improve the clarity of the kernel density plot, hence improve the accuracy of the prediction. However, statistical classification of an observation as an outlier is dependent on the underlying distribution of the data, which is what we are trying to determine, so simply being an "extreme" observation is not a sufficient justification for removing it. In the future, we plan to enhance the tool with more statistical tests for outliers identification.

5.6 Conclusion

In this study, we investigated the use of neural networks for distributional models classification as an alternative to the exploratory analysis step (Figure 4.3). Given a set of independent empirical observations obtained from an unknown process or phenomenon, we showed that a neural networks classifier is capable of identifying which distributional model is best suited for the input data. This study is intended to serve as a proof of concept to demonstrate the viability of our method, and show that it is possible to train neural networks to guess the 'best' probability distribution that characterizes a dataset. Even though, the number of distributions examined in this research is limited, they still cover the most encountered models in science and engineering. We are currently working to increase the number of distributions classified by our neural networks to support distributions with one or more shape parameters (e.g., weibul, lognormal,..) and other potential use cases.

We chose two neural networks models depending on the sample size and apply a suitable normalization technique (kernel density normalization or u-score normalization) then run the points through the neural networks to predict the "best fit" distribution. We validated the results by comparing them to a traditional statistical approach: parameter estimation by maximum likelihood with subsequent goodness of fit ranking by Anderson-Darling (MLE-AD). We showed that our trained neural networks outperform MLE-AD in a majority of cases.

Chapter 6

DeepFit

In the previous chapters, we showed the importance of distribution fitting in science and engineering in general. We also explained that, determining a suitable distributional model requires some background in statistics and familiarity with several probability distributions, skills that some analysts may not be equipped with. As such, this chapter presents DeepFit, a tool that uses the neural networks models, which we previously trained on a large database of commonly used distributions, in addition to traditional statistical tests to automate the distributional modeling process. DeepFit is based on the two best performing neural networks that we presented in chapter 5 and published in [109]-[107].

6.1 Introduction

Determining an appropriate probability distribution for a uni-variate dataset is a critical first step as it can guide the subsequent statistical analysis and impact the statistical model checking verification results. This step typically requires significant statistical knowledge and familiarity with the properties of a number of probability distributions, in order to fit the parameters of the identified model and assess its goodness of fit. Some scientists have attempted automating this task by creating tools that rely on traditional statistical assessments to pre-process data and run some goodness of fit tests to be able to rank and identify a good representation of the data. Similar tools and packages include Dataplot [49], Fitdistrplus [52], ExpertFit[125] and Easy-fit [177].

In this chapter, we present DeepFit, a software that is based on a combination of deep learning and the traditional statistical approach of conducting distributional modeling. First, a neural networks (NN) model that was previously trained on a large suitable database is used to identify the best candidate distribution from an input data. Then statistical techniques such as maximum likelihood are used to estimate the parameters of the selected model. Moreover, DeepFit applies several goodness of fit tests to evaluate and confirm the NN classifications. Furthermore, DeepFit users can also choose to, completely, bypass the NN classifier and use the implemented goodness of fit tests to simply rank the probability distributions from the best fit to the least. The details of the NN classifiers can be found in chapter 5.

Unlike existing tools, DeepFit does not require performing any parameter estimation in the distribution identification stage. Moreover, DeepFit allows the use of the standard statistical tests, the NN classifiers or both. This is particularly interesting in the case where the traditional statistical approach performs poorly (e.g., for datasets with larger/smaller numbers of observations).

6.2 Architecture

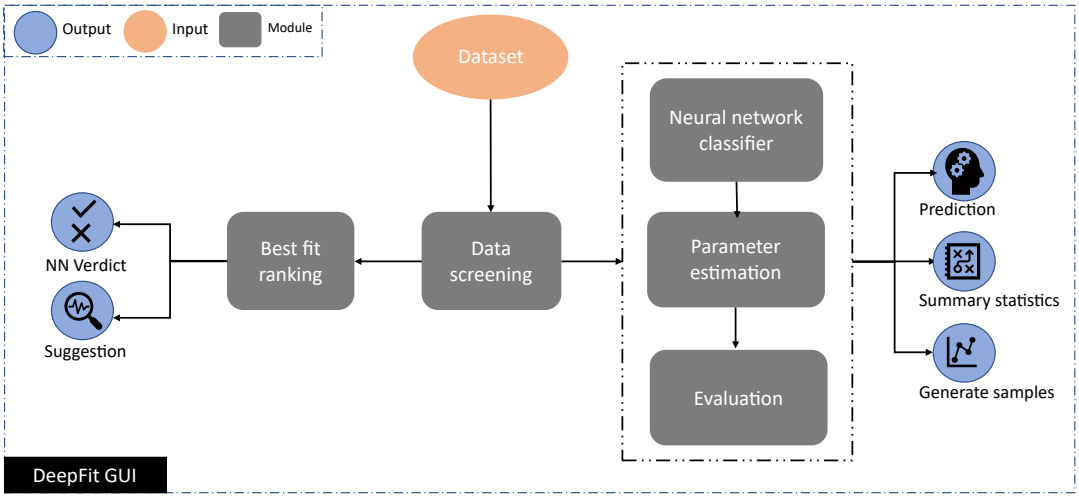

FIGURE 6.1: DeepFit architecture

DeepFit is a tool for automating the distribution fitting process for uncensored and unbinned uni-variate data. It relies on neural networks in order to identify the 'best' candidate model from a set of commonly used distributions.[1] Then, it applies traditional statistical techniques such as the maximum likelihood and many goodness of fit tests to estimate the parameters of the selected distribution and assesses its appropriateness. The purpose of the tool is to help engineers with limited statistical background save time and more easily determine an appropriate distributional model. DeepFit has four modules (Figure 6.1):

1. Data screening;
2. Neural networks classification;
3. Parameter estimation;
4. Evaluation;
5. Best Fit ranking.

In the next subsections, we will go over each module in details.

6.2.1 Data screening

The first step in using DeepFit is to pre-process and validate the input data for the neural networks classifier.

A number of graphs are provided to help the analyst determine various characteristics of the data (e.g., if the data is symmetric, if the data is skewed and in which direction is the skewness, if there are any extreme observations). These graphs can also help to identify outliers. In this context, the purpose of identifying outliers is simply to help the analyst determine whether an observation is erroneous (e.g., is it mis-coded?) and not to perform formal outlier analysis. Formal outlier analysis

[1] uniform, 2: normal, 3: logistic, 4: exponential, 5: double-exponential, 6: half-normal, 7: half-logistic, 8: gumbel-min, 9: gumbel-max

is based on assuming that the underlying distribution is known (most outlier tests are based on assuming the data is normally distributed). However, DeepFit assumes that the underlying distribution is unknown and in fact the purpose of the tool is to determine an appropriate distributional model. An "extreme" point may indicate a bad data value or it may be a reflection of the underlying distribution of the data, so it is recommended that an observation be removed only if it can reasonably be determined to be erroneous. It should be noted that "noisy" data is an indication that the observations do not come from a single common distribution. This type of data may be more appropriately modeled with a mixture distribution which is beyond the scope of DeepFit tool at the time of this writing. However, further use cases and more distributions will be supported soon.

This first module of DeepFit, screens the data by using the 4-plot method [59]. This method consists of generating four types of plots, designed to check whether the data are independent with each other and draw from a common distribution with fixed location and fixed scale values. Datasets that do not satisfy these assumptions should not be modeled with a single distributional model. The analyst is engaged in understanding their data while locating and removing any mis-coded points that could be considered as outliers. Note that, simply being an "extreme" observation is not a sufficient justification for removing a value from the input set. DeepFit generates four graphs corresponding to the 4-plot method that James J Filliben [97] developed [59]. The rationale is that the data should be random (i.e., independent) draws from a common distribution with a fixed location and fixed scale. The four plots are:

1. The run sequence plot which is used to identify whether the fixed location and fixed scale assumptions are reasonable. In addition, it can identify if there are trends in the data which would indicate a lack of independence. This plot, generally, answers three questions: Does the "location" or "scale" shift? Are there any trends in the data over time? and Are there any "outliers"? Note that this plot should be performed on the unsorted data.

2. The lag plot that is specifically testing for first order autocorrelation (a dataset that shows significant autocorrelation is not independent). This plot should basically look like a random blob. Patterns in the data are an indication that the data is not independent. Note that these first two plots should only be generated if the data is available in the order that it is collected. If the data is pre-sorted, then these plots will show very strong dependence (e.g., the lag plot will basically be a straight line).

3. The third plot which is designed to show the distribution of the data. This can be either a histogram or a kernel density plot. If this plot is bell-shaped, the underlying distribution is symmetric and perhaps approximately normal (chapter 4).

4. The normal probability plot that though it is usually used to check for normality, in the context of screening the data it can be useful for identifying outlying points.

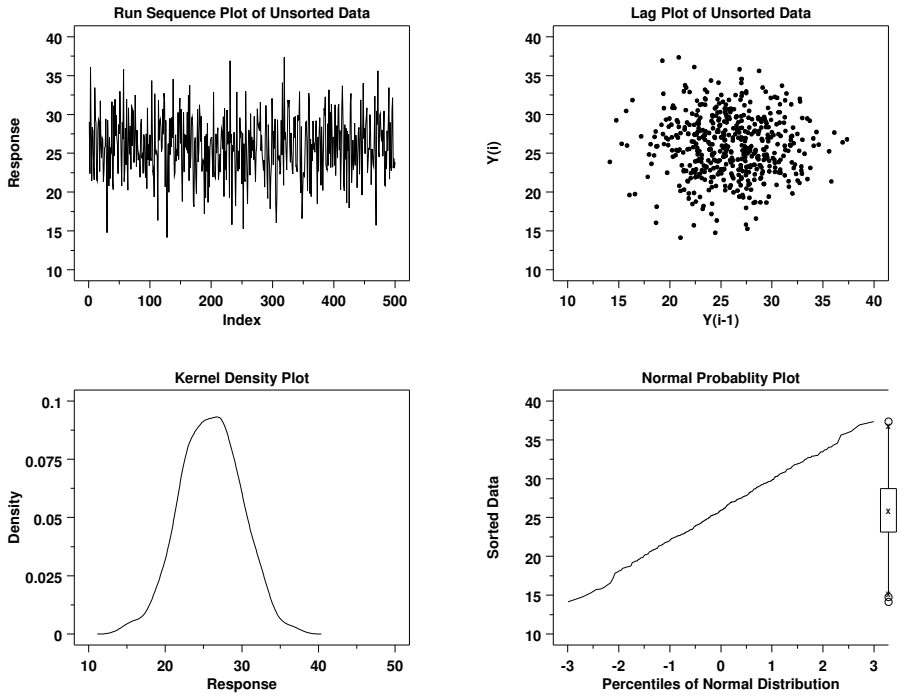

FIGURE 6.2: 4-plot of 500 random normal points.

This module identifies some underlying assumptions (e.g., normality, shape, scale and location, existence of outliers) about the input set and the process from which the measurements were collected. DeepFit only advocates removing an observation from the set if it can be determined that the observation is in fact a bad data point as opposed to simply being an outlier relative to a normal distribution. Once the analysts are satisfied with the data, they can proceed to the next module which uses the neural networks classifiers to fit the data.

Figure 6.2 shows an example of the 4-plot method applied to 500 normally distributed data points. Using these plots, we see no obvious patterns nor trends on the lag plot and the run sequence plot. This indicates that the data is independent and comes from a random process. Additionally, from the normal probability plot we learn that there are no visible outliers to remove. Moreover, from the kernel density plot (kdp), we can already interpret that the data is normally distributed since the kdp has a bell like shape and is symmetric. But since we assume that the analysts are unfamiliar with the shape of many probability distributions, they can proceed with the next step and let our trained NN classifiers predict the 'correct' model (chapter 5 explains the details of the neural networks used in DeepFit).

We also show another example of data sampled from the gumbel max distribution in Figure 6.3. Similar conclusions from figure 6.2 about the data independence still apply to this figure as well. We can also confidently confirm the absence of any extreme points from the normal probability plot. However, we can't clearly identify the shape of the distribution from the kernel density plot except that it's an upper tailed distribution. Therefore, the analyst is advised to use our neural networks classifier to predict the 'best' model for their data. This figure and dataset is a great use case that proves the usefulness of our tool.

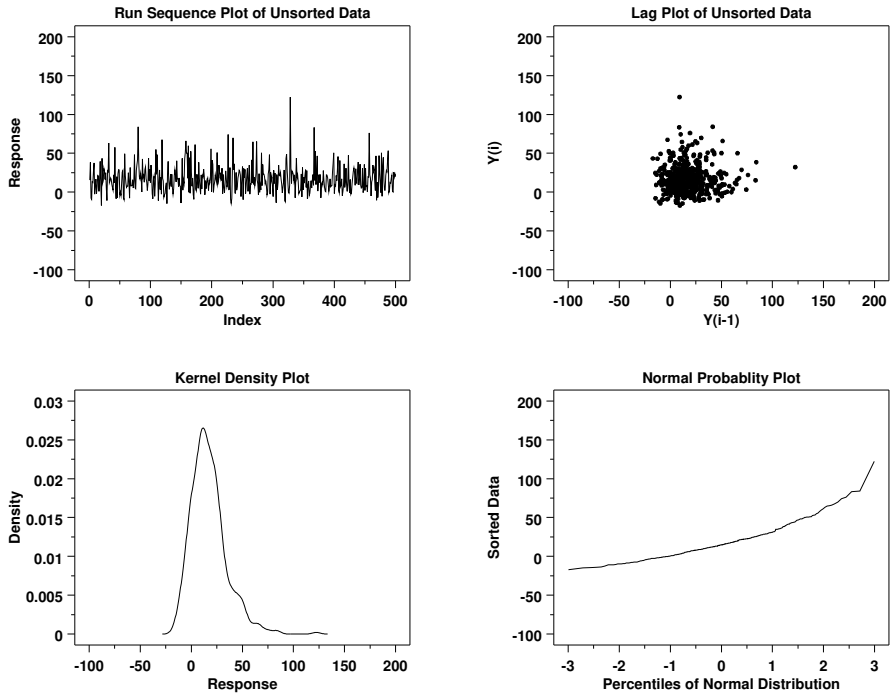

FIGURE 6.3: 4-plot of 500 random gumbel max points.

6.2.2 Neural Network classification

In this module, an initial transformation is applied to the input dataset, after the screening has been conducted in the first module and prior to using the neural networks. This transformation consists of either the u-score normalization or the kernel density normalization explained in chapter 5. Figures 6.4 and 6.5, respectively, show the impact of both normalization techniques on a dataset that was generated using 5000 random numbers from the logistic distribution.

The u-score algorithm transforms the kernel density heights to a (0,1) scale, whereas the kernel density normalization transforms the kernel density heights to integrate to 1 on the 1 to 256 x-coordinate scale. Nevertheless, as we stated in the previous chapter (i.e., chapter 5), these two normalization methods are non-distorting of the shape of the kernel density plot and preserve the form of the underlying probability distribution regardless of the location and scale values. We also noticed, in chapter 5, that neither of these techniques has significant impact on the accuracy of the NN classification. For this reason, the analyst doesn't need to worry about choosing the right method but is still encouraged to try both.

Additionally, DeepFit implements the two best performing NN models previously trained: one for smaller samples and one for larger samples. Therefore, depending on the size of the input data, DeepFit selects the corresponding NN classifier and uses it to select the 'right' candidate model from the list of supported distributions.[2]

Note that we are continuously adding more distributions to the tool in order to support families of distributions and other use cases. The details of how we trained the neural networks can be found in chapter 5.

[2]uniform, normal, logistic, exponential, half normal, half logistic, gumbel max, gumbel min and double exponential

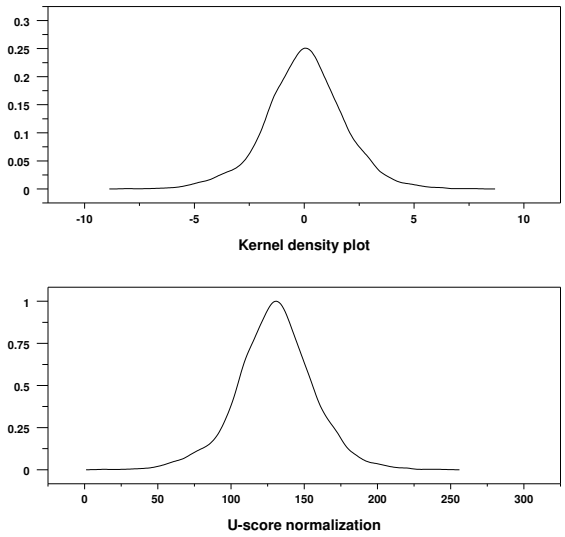

FIGURE 6.4: Impact of the u-score normalization

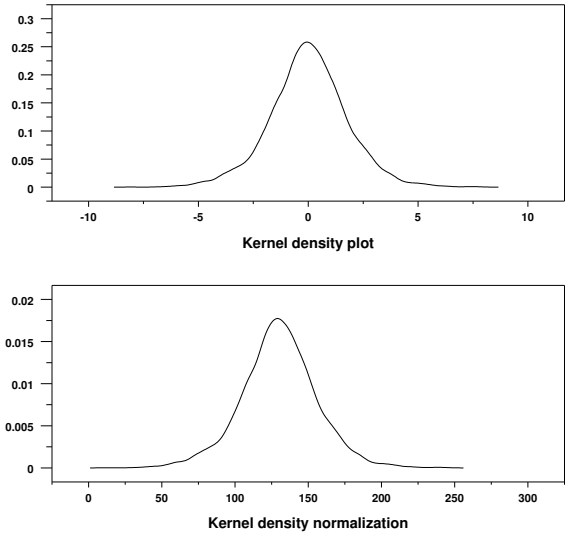

FIGURE 6.5: Impact of the kernel density normalization

6.2.3 Parameter estimation

Once the data has been pre-processed and the neural network classifier has identified the "best" distributional model, the parameters of the distribution are estimated in this module using traditional statistics and more specifically the maximum likelihood algorithm (MLE) explained in chapter 4 [60].

In this parameter estimation module, we present the analyst with different output sections:

Basic statistics

This first section prints out some basic summary statistics for the entered data. This includes the sample minimum, maximimum, mean, standard deviation, range, Skewness and kurtosis [3].

Parameter estimates

This next section computes and prints out the parameter estimates of the fitted distribution i.e., the location and scale values as well as the AIC, BIC and log-likelihood [77]. The generic formula for the log-likelihood LL is:

$$LL = \sum_{i=1}^{N} Ln(f(x))$$

That is, for each observation x in the dataset, a probability density function $f(x)$ for that observation is computed with the specified location/scale values. Then, the log-likelihood is obtained by summing all their logs. In some cases, we used the algorithms implemented in dataplot [49] when available to calculate these parameters as they include some simplifications for specific distributions. Once the log-likelihood value is computed, BIC and AIC can be determined as follows:

$$BIC = -2 * LL + p * Ln(N)$$

$$AIC = -2 * LL + 2 * p$$

where LL is the log-likelihood value, N is the sample size and p is the number of parameters. For the currently supported distributions, p is equal to 2 (i.e., location and scale).

Confidence intervals

The confidence intervals for the location and scale parameters are computed for multiple confidence values of α:

$$\alpha = 1 - \frac{c}{100}$$

where c is the confidence level (i.e., 90%, 95%, 99%). So, for a 95% confidence level, α is usually given as 0.05, for a 90% confidence level α is given as 0.10 and for a 99% confidence interval α is given as 0.01.

In DeepFit, the confidence intervals are computed via two methods: the non-parametric bootstrap algorithm or using analytical methods which are based on the the percent point function [5] of the chi-square [7] or the T-distributions [6]. As an exception, the confidence intervals for the logistic distribution are computed using assigned tables. The bootstrap can be used to calculate the confidence intervals for the estimated parameters of the distribution (location and scale) as well as their standard errors. The latter is not currently supported at the time of this writing but we will incorporate it in the future. The primary reason for including the bootstrap in DeepFit is that sometimes analytical methods are too difficult or intractable to compute. And the second reason is that the bootstrap can sometimes be more accurate when the underlying assumptions on the data are not met. Additionally, in DeepFit, we consider that if the analytical and bootstrap results are reasonably consistent,

this is an indication that the selected distribution is, potentially, a good match for the data.

The basic bootstrap algorithm is as follows (n is the number of observations for the dataset):

Step 1: Generate a sample of size n from the original observations with replacement. This idea of "with" replacement is key which means that the observations from the original sample may occur multiple times or not at all in the bootstrap sample. This means that, as an example, if the original sample size is 500, we are generating an index value from 1 to 500 (i.e., the index value is the order by which the original observations were collected) and we are generating 500 such index numbers. Note that, each index is randomly selected from 1 to 500, so the 500^{th} index values are not unique. We are currently using the Mersenne Twister as the default random numbers generators in DeepFit [142].

Step 2: Compute the parameter estimates from the new sample (i.e., location and scale). We are using the same algorithm used to compute the location and scale for the original data, that is the maximum likelihood.

Step 3: Repeat the above two steps 10,000 times and obtain 10,000 scale and location values. The standard errors for the location and scale are, simply, the standard deviations of the newly computed location and scale values. Whereas, to determine the confidence intervals at different α, symmetric intervals are used. For example, at $\alpha = 0.05$ (i.e., 95% confidence level), we take the 2.5% and 97.5% percent points function values of these samples. The percent point function is another probability function used to define a probability distribution. It is commonly referred to as the inverse distribution function (see [5] for more details on this function).

This bootstrap algorithm was applied to the following distributions: gumbel max, gumbel min, half logistic and half normal whereas, for the normal, logistic, exponential and double exponential, we used the T distribution percent point function and the chi-square percent point function to determine the confidence intervals. Note that, DeepFit doesn't implement the bootstrap method for the uniform distribution. The reason for this is that the estimates for the lower and upper limits are based on the sample minimum and sample maximum. Since the bootstrap samples from the original dataset, the minimum and maximum in the bootstrap samples will never be smaller (or larger for the upper limit) than the original sample. Therefore, the bootstrap will not generate a reasonable lower bound for the confidence limit for the lower limit and likewise a reasonable upper bound for the confidence limit for the upper limit. Hence, we deployed the algorithm in these references [132, 151] for the uniform distribution instead.

6.2.4 Evaluation

This module includes traditional statistical goodness of fit techniques to determine if the distributional model suggested by the neural networks is in fact an appropriate distributional model for the data. In chapter 4, we have examined the three class categories for the goodness of fit test (GoF). That is

1. The first class is based on comparing the empirical cumulative distribution function (CDF) [5] (i.e., based on the data) to the theoretical CDF function;

2. The second class is based on comparing the differences between the empirical percent point function (PPF) to the theoretical percent point function;

3. The third class is based on the likelihood function.

In DeepFit. we implemented the Anderson-darling and the Kolmogorov Smirnov GoF tests from the first class category. Since DeepFit primarily focuses on unbinned data at this time, we implemented the AD and KS tests because they are more powerful for this type of observations. From the second class category, we implemented the probability plot correlation coefficient test (PPCC). Finally, from the last class category, we implemented the Akaike's Information Criterion (AIC) and the the Bayesian Information Criterion (BIC). We refer the reader to chapter 4 for more information on these tests.

Note that we are using these goodness of fit statistics for two distinct purposes:

Goal 1: Once the neural networks prediction of the 'best' fit distribution is complete, in order to assess the accuracy of the fitted distribution.

Goal 2: Without going through the neural networks classification, the GoF tests are used to simply rank the supported distributions from the most probable fit to the least probable fit. For this we compute the value of the statistic for each of AD, KS and PPCC and provide a sorted list of distribution ranks.

Nevertheless, two additional concerns remain for consideration when using these GoF tests. First, when computing critical values, there is a distinction between the case where the parameters are "known" and the case where the parameters are estimated from the data. We are really only interested in the case where the parameters are estimated from the data. In particular, one claim for the KS is that the critical values are the same regardless of the underlying distribution. However, this is based on the assumption that the parameters are known. Second, for some methods, there are tables of critical values. Specifically, critical values for AD have been generated for a number of common distributions. Alternatively, the critical values can be computed via simulation.

6.2.5 Best fit ranking

When the analyst chooses to completely skip the neural networks classification and strictly use traditional statistics to determine which distribution is better fitting for the data, DeepFit offers this module which relies mainly on three goodness of fit tests examined in the previous module: Anderson Darling (AD), Kolmogorov Smirnov (KS) or the probability plot confidence coefficient (PPCC). This option allows to simply rank all the currently supported distributions from the "best" fit to the "least". First, the analyst is prompt with selecting the goodness of fit test of choice. Then, DeepFit relies on the computed values of the test statistic for all of the distributions to establish a ranking. The ranking algorithm follows these steps:

Step 1: Select the method for the parameter estimation. That is, for KS and AD the parameters are computed using the maximum likelihood algorithm whereas for the PPCC, the estimates are based on the PPCC value from the probability plot;

Step 2: Select a Goodness of fit test and compute its value of the test statistic (AD, KS or PPCC);

Step 3: Rank all the distributions based on their values of the test statistic. For the KS and AD, a lower value is better ranked. Whereas, for the PPCC, a higher value is preferred;

Step 4: Show the estimated location and scale values next to the established ranking, for each distribution.

The output of this module is similar to the output generated using the dataplot software [49]. Dataplot uses the "BEST DISTRIBUTIONAL FIT" command to display the ranking of the supported distributions (see [29] for examples of dataplot).

6.3 Tool Assessment

DeepFit was evaluated on synthetic data [109] and real world data and successfully modeled several commonly used distributions. In this section, we use one example of real measurements obtained from a published study on Heat Flow Meter Calibration & Stability Analysis [106] to demonstrate the functionalities of DeepFit.

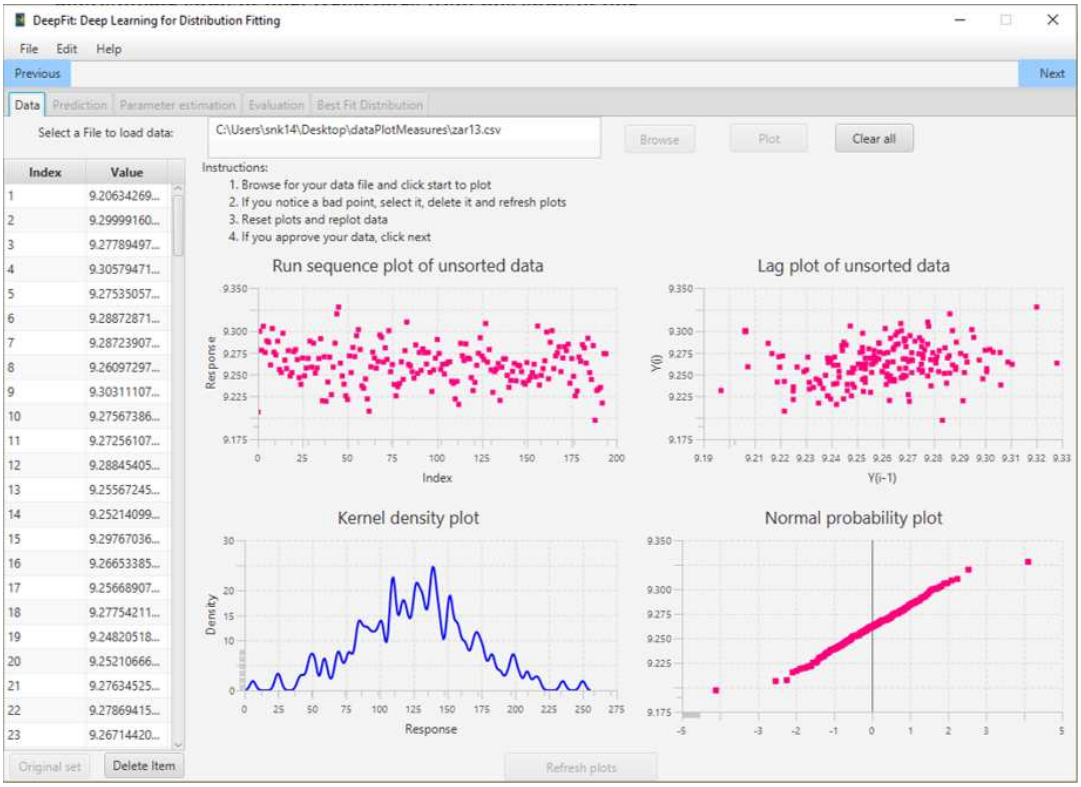

FIGURE 6.6: Module 1

Figure 6.6 shows the first module of the tool, in which the 4-plot method is deployed to screen the data for randomness, test for data normality and check the existence of any extreme points that might potentially be considered as outliers.

When uploading the Heat Flow Meter Calibration & Stability Analysis data to DeepFit, we first notice that the kernel density plot is symmetric and that the normal probability plot is linear. This suggests potential normality in the this set (i.e., the underlying distribution might be the normal pdf). Furthermore, no visible trends or structures are seen on the lag and the run sequence plot which indicates that the

data is from a random process. Note that, we intentionally colored the kernel density plot in a color that's different than the other three plots. This is because the analyst is prompted to carefully inspect the kdp prior to passing it to the the neural network classifiers to make sure that the shape is clear and not distorted.

Next, the pre-screened data is normalized by selecting one of the scaling methods discussed previously (i.e., u-score or k-score) then ran through the neural network model to predict the 'right' candidate distribution for the data. Figure 6.7 shows that DeepFit plots the original kdp of the data as well as the normalized kdp. Moreover, this figure indicates that using the k-score, the NN classifier was able to identify the underlying distribution as the normal pdf which matches our intuition from the first module (i.e., Figure 6.6).

The next step, as shown in Figure 6.8 is to compute the parameters of the distribution. That is, the location, scale estimates and their associated confidence intervals in addition to some basic summary statistics. The analyst is also prompted with the option to enter the number of observations N in order to generate N random numbers from the specified distribution. These numbers can be saved into a local file in a comma separated values (csv) format. The tool also overlays the fitted probability distribution plot on top of the kernel density plot of the original data. This feature allows the user to graphically assess the validity of the fit. Figure 6.8 shows that the stacked kernel density plots of the original data and 1000 random samples from the NN estimation, are very similar in shape indicating that the prediction is somehow accurate.

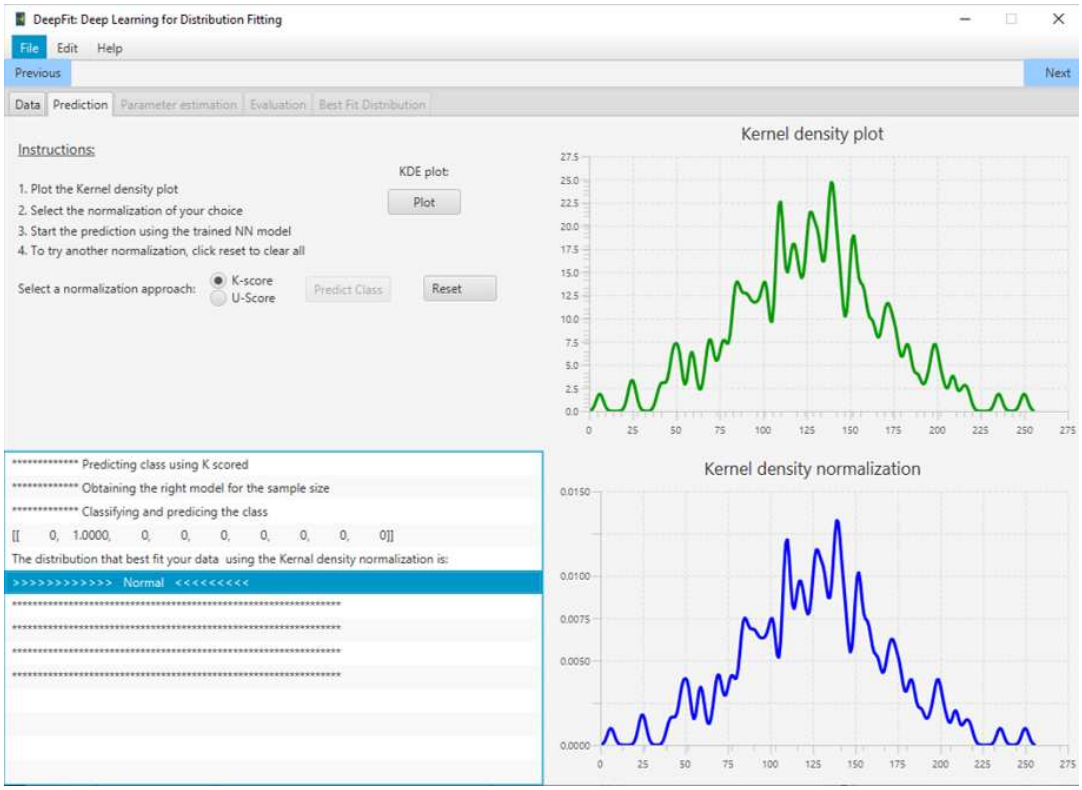

FIGURE 6.7: Module 2

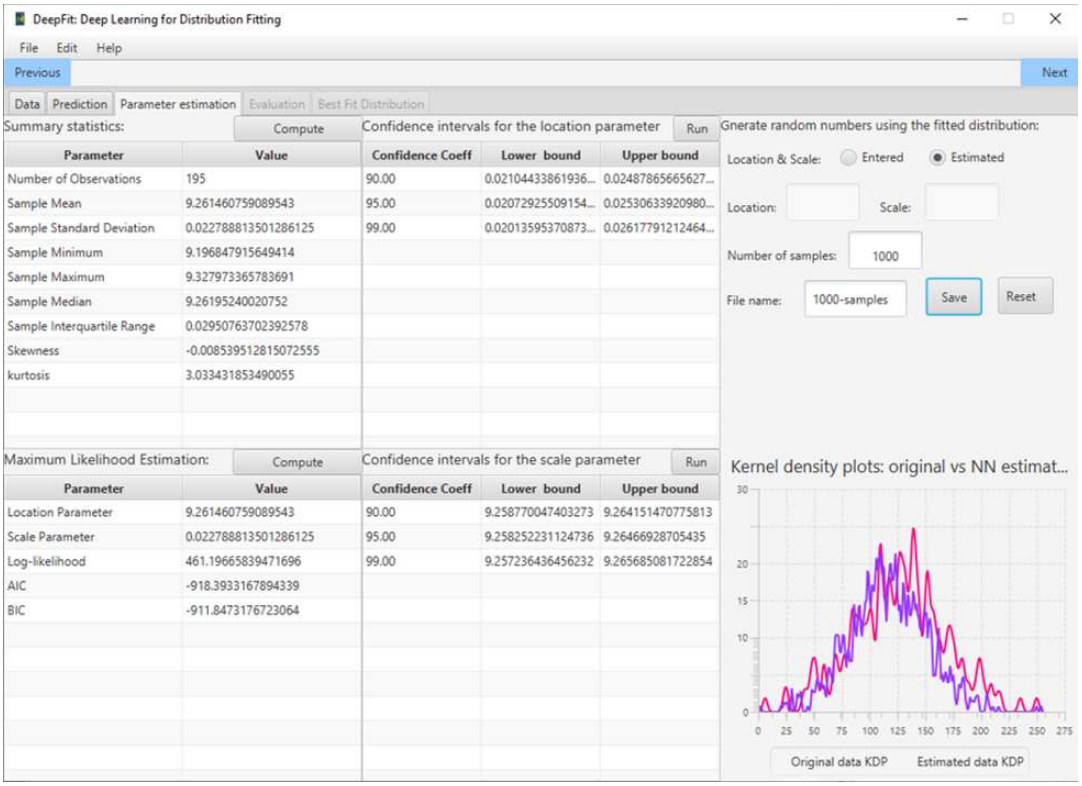

FIGURE 6.8: Module 3

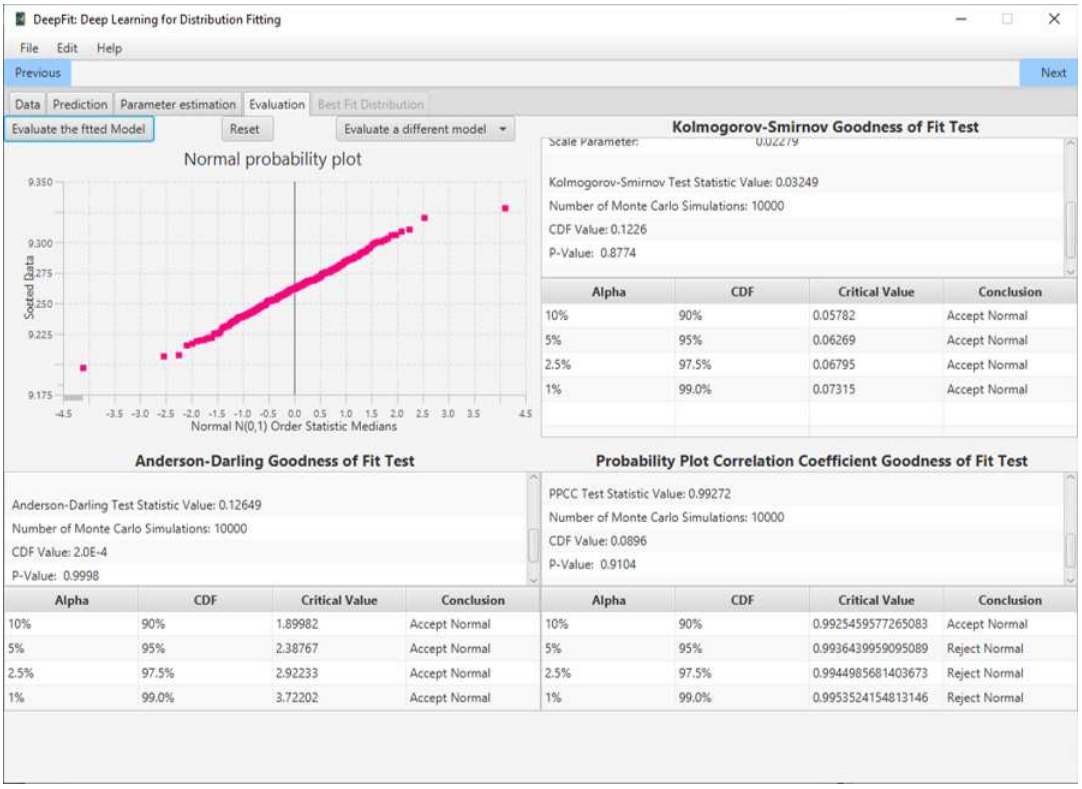

FIGURE 6.9: Module 4 - testing the NN model

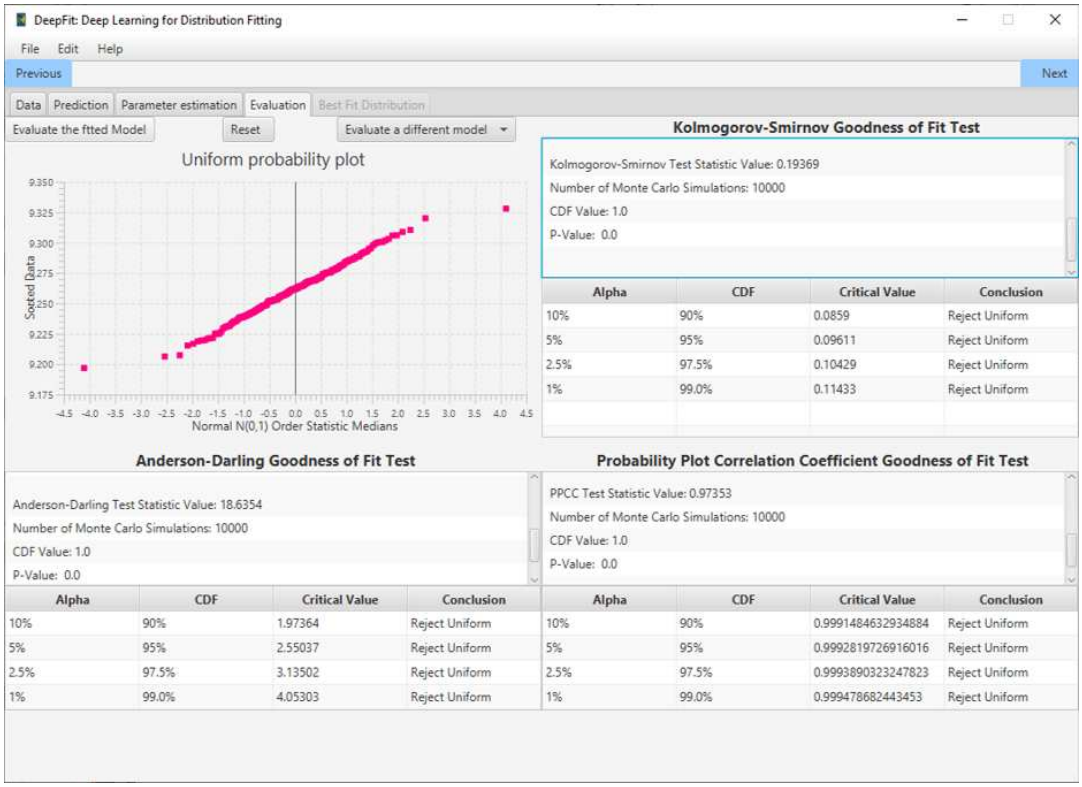

FIGURE 6.10: Module 4 - testing a different probability distribution

For further assessments, the analyst might choose to run several goodness of fit tests, included in DeepFit to confirm that the hypothesis about the underlying distribution is indeed correct. In Figure 6.9, we evaluate the data against the NN fit (i.e., the normal distribution). Figure 6.10 demonstrates how the analyst can test a different distribution other than the NN selected pdf. In this figure, we checked the data against the uniform distribution which got eventually rejected by all four GoF tests as expected. Note that, for each tested pdf, we complement the goodness of fit tests with a graphical test (we currently use the probability plot).

In certain cases, the analyst might notice that different goodness of fit tests can reach different conclusions or that for the same test different conclusions are obtained according to the value of α. This is not uncommon. In fact, the first case can be interpreted as each test evaluating specific features of the data. The AD, KS and PPCC can be sensitive to different types of departures from the hypothesized distributions (e.g., AD .est is specifically designed to be more sensitive to differences in the tails than the KS test). The second case, where opposing conclusions are obtained withing the same GoF test, depending on the value of α, is also common and happens often. The reasoning behind this is that, when we say something is "statistically significant", we are really saying "statistically significant at a specific confidence level" (when the level is not specified it is assumed to be 95% by default). The p-value (or alternatively, the cdf value) gives the specific cut-off for the statistical significance. For the AD and KS, which are upper-tailed tests (i.e., smaller values of the test statistic are better), the 90% level is actually a stricter test than the 95% level and the 95% level is a stricter test than the 99% test. Whereas, the PPCC test is a lower tailed test, hence, the directions get reversed (i.e., larger values of the test statistic are preferred). Similarly, the 99% level is actually a stricter test than the 95% level and the 95% level is a stricter test than the 90% level. However, in terms of users interpretations, having different conclusions at 90%, 95% and 99% generally indicates

that the fit is probably "borderline". The intended usage of the distributional model drives the practical interpretation as to what significance level is appropriate and what action to take.

Figure 6.11 shows the final feature provided by DeepFit, which is the option to bypass the neural networks and simply run a few goodness of fit tests. In this context, we apply it to the Heat Flow Meter Calibration & Stability Analysis data. Note that, there are two downside of strictly using the goodness of fit tests rather than our proposed workflow (which combines the neural networks and traditional statistics). For a small number of observations, the GoF tests can have low power (i.e., it takes large departures from the hypothesized distribution to reject) and for large sample sizes the tests can be overly sensitive (i.e., small departures from the hypothesized distribution that have little practical significance can reject). In DeepFit, we resolve these sample size concerns by complementing the neural networks prediction with a variety of formal tests as well as probability plots. Because in some scenarios, graphical tools are more illuminating than analytical methods.

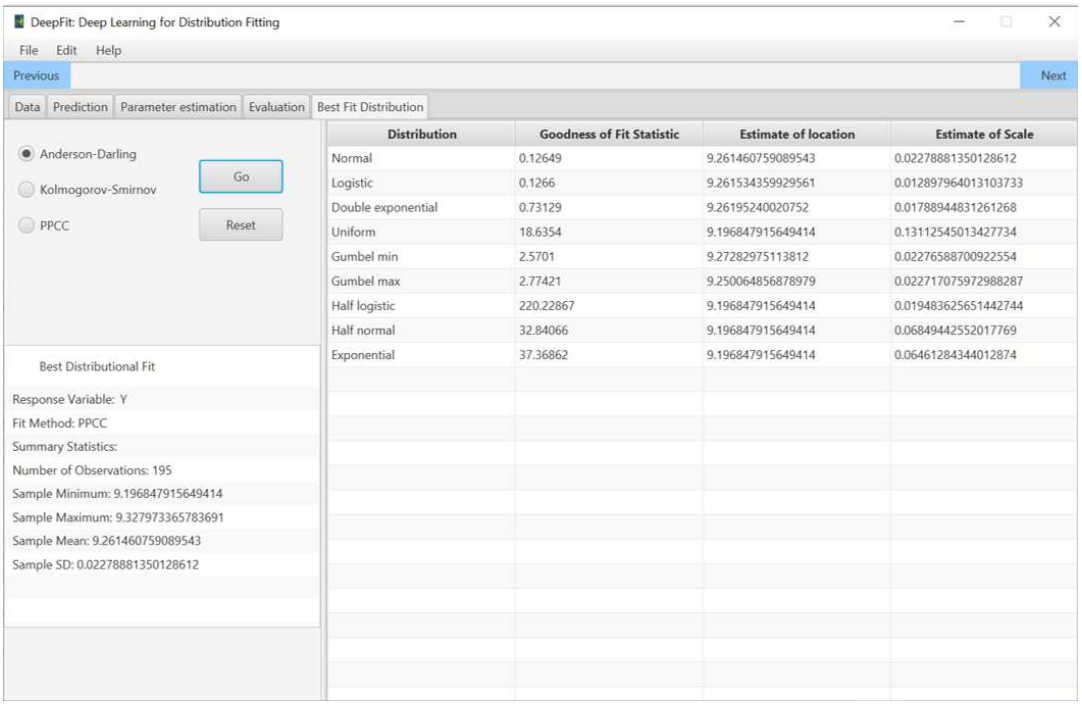

FIGURE 6.11: Module 5

6.4 Conclusion

In this chapter, we propose DeepFit, a combined effort between neural networks and statistical techniques for data analysis and distribution fitting. DeepFit provides a preliminary step of data screening to remove bad data points in the sense of being miscoded. Then uses one of the two neural network classifiers, that were previously trained on a large database of commonly used distributions, to select the 'best' candidate model given a set of empirical observations. Moreover, the tool incorporates a variety of traditional statistics designed to compute the parameters of the selected distribution as well as assess it's appropriateness and accuracy.

In the future, we plan to extend the number of supported distributions to also include families of distributions, such as the weibull, lognormal, Gamma distributions and other use cases as well as incorporate the ability to make more specific classifications (e.g., distinguish between weibull or lognormal) and compare this to approaches such as the likelihood ratio test [56], [55].

DeepFit has the advantage of being used as a standalone tool for scientists and engineers or it could be integrated into the workflow of the methodology, which we proposed in this thesis, for modeling and evaluating the performance of networking systems via statistical model checking (SMC). In the next final chapter, we explain how we combine DeepFit with SMC and conclude this thesis with perspectives and future work.

Chapter 7

Conclusion

The demand for faster performance, increased accessibility, mobility and secure communications has driven significant advancements in Internet architectures, protocols and applications. As such, scientists are constantly investigating methods to secure networks against malicious attacks, unintentional bugs and errors in addition to maintaining a good quality of service in network speed and performance. Unfortunately, it has been observed that their attempts of evaluating network performance and quality of service value more a culture of running code, heuristics and engineering judgements rather than a culture of sound proofs and rigorous verification methods. The accuracy obtained by these methods varies greatly, hence, loss of real insights into how the network acts can lead to unforeseen performance issues that degrade the quality of experience and service. Nevertheless, benchmarking and simulation based testing are still considered the default option for network analysis and supervision in the networking community, in spite of failing to achieve high accuracy results and obtaining trustworthy analysis.

In this thesis, we presented a methodology which is based on a framework for modeling systems called BIP and a formal verification approach that relies on SMC. SMC takes a stochastic model written in the BIP formalism and a property to verify. The stochastic model is generally obtained by modeling the functional behavior of a system, augmented with probabilistic variables, which are updated via probability distributions (PD). A PD is, typically, obtained by collecting and analyzing measurements from the system's execution using traditional statistical tests to select the best fit distribution (i.e., distribution fitting). We demonstrated the benefits of our methodology in addition to its feasibility and ease of use via an example of a software forwarder, of a novel Internet architecture, to which we maximized the throughput rate.

Additionally, we showed that distribution fitting is crucial for the correct assessment via SMC and it's an important preliminary step in science and engineering, in general. However, this task requires a good statistical background and familiarity with several distributions which is beyond the expertise of some analysts. Therefore, we built DeepFit, a tool that combines traditional statistical tests and deep learning to automate the distribution fitting process. DeepFit is then integrated into the workflow of our methodology for rigorous modeling and performance assessment of networking systems as seen in Figure 7.1.

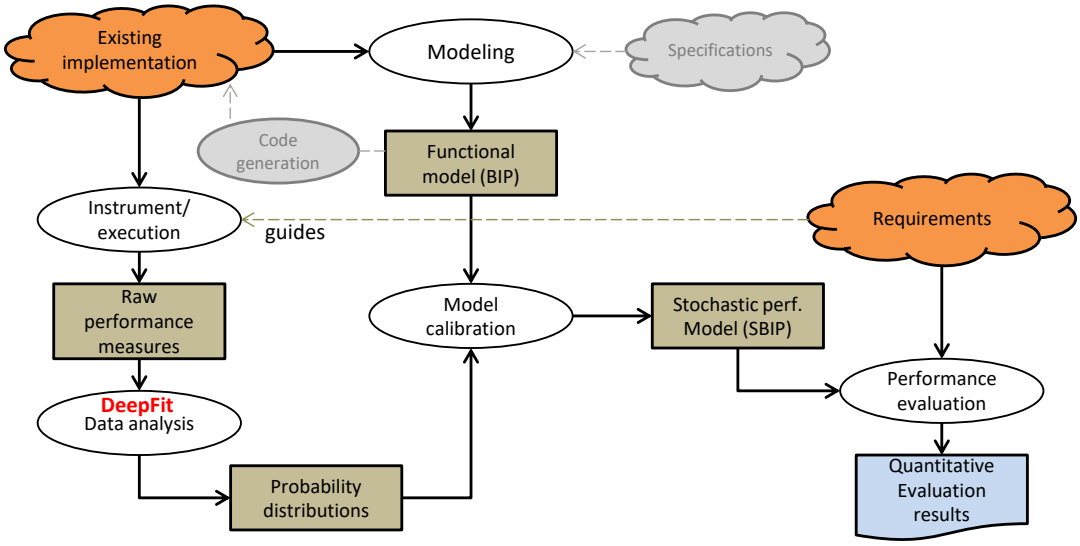

FIGURE 7.1: Methodology

Bibliography

[1] *1. Introduction — Theorem Proving in Lean 3.23.0 documentation*. Jan. 2021. URL: https://leanprover.github.io/theorem_proving_in_lean/introduction.html.

[2] *1.3.3.12. DEX Mean Plot*. 2003. URL: http://atomic.phys.uni-sofia.bg/local/nist-e-handbook/e-handbook/eda/section3/eda33c.htm.

[3] *1.3.5.11. Measures of Skewness and Kurtosis*. Jan. 2018. URL: https://www.itl.nist.gov/div898/handbook/eda/section3/eda35b.htm.

[4] *1.3.5.13. Runs Test for Detecting Non-randomness*. Jan. 2018. URL: https://www.itl.nist.gov/div898/handbook/eda/section3/eda35d.htm.

[5] *1.3.6.2. Related Distributions*. Jan. 2018. URL: https://www.itl.nist.gov/div898/handbook/eda/section3/eda362.htm.

[6] *1.3.6.6.4. t Distribution*. URL: https://www.itl.nist.gov/div898/handbook/eda/section3/eda3664.htm.

[7] *1.3.6.6.6. Chi-Square Distribution*. URL: https://www.itl.nist.gov/div898/handbook/eda/section3/eda3666.htm.

[8] *1.5 Performance — Computer Networks: A Systems Approach Version 6.2-dev documentation*. URL: https://book.systemsapproach.org/foundation/performance.html.

[9] *6.4.4.8.1. Box-Ljung Test*. Jan. 2018. URL: https://www.itl.nist.gov/div898/handbook/pmc/section4/pmc4481.htm.

[10] *8.2.3.2. Goodness of fit tests*. URL: https://www.itl.nist.gov/div898/handbook/apr/section2/apr232.htm.

[11] *A Brief Introduction to Named Data Networking - Named Data Networking (NDN)*. URL: https://named-data.net/publications/ndn18.

[12] *A Simple Network Management Protocol (SNMP)*. URL: https://datatracker.ietf.org/doc/html/rfc1157.

[13] *About - OpenDaylight*. URL: https://www.opendaylight.org/about.

[14] *About Internet Society | Internet Society*. URL: https://www.internetsociety.org/about-internet-society.

[15] Ajin Abraham. "Chapter 3 - Static Analysis". In: *Automated Security Analysis of Android and iOS Applications with Mobile Security Framework*. Syngress, Jan. 2016, pp. 7–14. ISBN: 978-0-12-804718-7. DOI: 10.1016/B978-0-12-804718-7.00003-5.

[16] Husain Aljazzar et al. "Safety Analysis of an Airbag System Using Probabilistic FMEA and Probabilistic Counterexamples". In: *2009 Sixth International Conference on the Quantitative Evaluation of Systems*. 2009, pp. 299–308. DOI: 10.1109/QEST.2009.8.

[17] Sahel Alouneh et al. "A comprehensive study and analysis on SAT-solvers: advances, usages and achievements". In: *Artif. Intell. Rev.* 52.4 (Dec. 2019), pp. 2575–2601. ISSN: 1573-7462. DOI: 10.1007/s10462-018-9628-0.

[18] Leopoldo Angrisani and Claudio Narduzzi. "Measurements for Networking: An Overview". In: *2008 IEEE Instrumentation and Measurement Technology Conference*. 2008, pp. 1328–1333. DOI: 10.1109/IMTC.2008.4547248.

[19] D. Antos, V. Rehák, and J. Korenek. "Hardware Router's Lookup Machine and its Formal Verification". In: 2004.

[20] *ARPA Becomes DARPA*. URL: https://www.darpa.mil/about-us/timeline/arpa-name-change.

[21] P. Bagade, A. Banerjee, and S. K. S. Gupta. "Validation, Verification, and Formal Methods for Cyber-Physical Systems". In: *Cyber-Physical Systems*. Cambridge, MA, USA: Academic Press, Jan. 2017, pp. 175–191. ISBN: 978-0-12-803801-7. DOI: 10.1016/B978-0-12-803801-7.00012-2.

[22] Christel Baier et al. "Symbolic model checking for probabilistic processes". In: *Automata, Languages and Programming*. Berlin, Germany: Springer, June 2005, pp. 430–440. ISBN: 978-3-540-63165-1. DOI: 10.1007/3-540-63165-8_199.

[23] A. Basu et al. "Verification of an AFDX Infrastructure using Simulations and Probabilities". In: *Runtime Verification, RV'10*. Vol. 6418. LNCS. Springer, 2010.

[24] Ananda Basu, Marius Bozga, and Joseph Sifakis. "Modeling Heterogeneous Real-time Components in BIP". In: *Proceedings of the Fourth IEEE International Conference on Software Engineering and Formal Methods*. SEFM'06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 3–12. ISBN: 0-7695-2678-0.

[25] Ananda Basu et al. "Rigorous Component-Based System Design Using the BIP Framework". In: *IEEE Software* 28.3 (May 2011), pp. 41–48. DOI: 10.1109/MS.2011.27.

[26] Ananda Basu et al. "Verification of an AFDX Infrastructure Using Simulations and Probabilities". In: *Runtime Verification*. Berlin, Germany: Springer, Nov. 2010, pp. 330–344. ISBN: 978-3-642-16611-2. DOI: 10.1007/978-3-642-16612-9_25.

[27] Ananda Basu et al. "Verification of an AFDX Infrastructure Using Simulations and Probabilities". In: *Runtime Verification*. Berlin, Germany: Springer, Nov. 2010, pp. 330–344. ISBN: 978-3-642-16611-2. DOI: 10.1007/978-3-642-16612-9_25.

[28] Pankaj Berde et al. "ONOS: towards an open, distributed SDN OS". In: *HotSDN '14: Proceedings of the third workshop on Hot topics in software defined networking*. New York, NY, USA: Association for Computing Machinery, Aug. 2014, pp. 1–6. ISBN: 978-1-45032989-7. DOI: 10.1145/2620728.2620744.

[29] *BEST DISTRIBUTIONAL FIT*. Nov. 2020. URL: https://www.itl.nist.gov/div898/software/dataplot/refman1/auxillar/bestfit.htm.

[30] K. Bhargavan et al. "Verisim: formal analysis of network simulations". In: *IEEE Trans. Software Eng.* 28.2 (Aug. 2002), pp. 129–145. ISSN: 1939-3520. DOI: 10.1109/32.988495.

[31] Karthikeyan Bhargavan, Davor Obradovic, and Carl A. Gunter. "Formal verification of standards for distance vector routing protocols". In: *J. ACM* 49.4 (July 2002), pp. 538–576. ISSN: 0004-5411. DOI: 10.1145/581771.581775.

[32] Armin Biere et al. "Linear Encodings of Bounded LTL Model Checking". In: *arXiv* (Nov. 2006). DOI: 10.2168/LMCS-2(5:5)2006. eprint: cs/0611029.

[33] *BIP Tools*. Sept. 2015. URL: https://www-verimag.imag.fr/BIP-Tools-93.

[34] Jasmin Christian Blanchette, Lukas Bulwahn, and Tobias Nipkow. "Automatic Proof and Disproof in Isabelle/HOL". In: *Frontiers of Combining Systems*. Berlin, Germany: Springer, Oct. 2011, pp. 12–27. ISBN: 978-3-642-24363-9. DOI: 10.1007/978-3-642-24364-6_2.

[35] G. Bochmann and C. Sunshine. "Formal Methods in Communication Protocol Design". In: *IEEE Trans. Commun.* 28.4 (Apr. 1980), pp. 624–631. ISSN: 1558-0857. DOI: 10.1109/TCOM.1980.1094685.

[36] Dominique Borrione et al. "A Formal Approach to the Verification of Networks on Chip". In: *J. Embedded Systems* 2009.1 (Mar. 2009), pp. 548324–14. ISSN: 1687-3963. DOI: 10.1155/2009/548324.

[37] Tom van den Broek and Julien Schmaltz. "Towards a formally verified network-on-chip". In: *2009 Formal Methods in Computer-Aided Design*. IEEE, Nov. 2009, pp. 184–187. ISBN: 978-1-4244-4966-8. DOI: 10.1109/FMCAD.2009.5351124.

[38] *Brown, B. (2019). Cisco, UCLA & more launch Named Data Networking Consortium. [online] Network World.* URL: https://www.networkworld.com/article/2602109/ucla-cisco-more-join-forces-to-replace-tcpip.html.

[39] Peter Bulychev et al. "UPPAAL-SMC: Statistical Model Checking for Priced Timed Automata". In: *arXiv* (July 2012). DOI: 10.4204/EPTCS.85.1. eprint: 1207.1272.

[40] Doron Bustan, Sasha Rubin, and Moshe Y. Vardi. "Verifying ω-Regular Properties of Markov Chains". In: *Computer Aided Verification*. Berlin, Germany: Springer, July 2004, pp. 189–201. ISBN: 978-3-540-22342-9. DOI: 10.1007/978-3-540-27813-9_15.

[41] Michael Carney et al. "Predicting probability distributions for surf height using an ensemble of mixture density networks". In: *Proceedings of the 22nd international conference on Machine learning* (2005).

[42] Hao Chen, Drew Dean, and David Wagner. "Model checking one million lines of C code". In: *In Proceedings of the 11th Annual Network and Distributed System Security Symposium (NDSS*. 2004, pp. 171–185.

[43] Jen-Tzung Chien. "Chapter 7 - Deep Neural Network". In: *Source Separation and Machine Learning*. Cambridge, MA, USA: Academic Press, Jan. 2019, pp. 259–320. ISBN: 978-0-12-817796-9. DOI: 10.1016/B978-0-12-804566-4.00019-X.

[44] Frank Ciesinski and Marcus Größer. *On Probabilistic Computation Tree Logic*. 2002.

[45] E. M. Clarke, E. A. Emerson, and A. P. Sistla. "Automatic verification of finite-state concurrent systems using temporal logic specifications". In: *ACM Trans. Program. Lang. Syst.* 8.2 (Apr. 1986), pp. 244–263. ISSN: 0164-0925. DOI: 10.1145/5397.5399.

[46] Gerald D. Cole, California Univ Los Angeles School Of Engineering Science, and Applied. "Computer Network Measurements: Techniques and Experiments". In: *DTIC* (Oct. 1971). URL: https://apps.dtic.mil/sti/citations/AD0739344.

[47] Eric Conrad, Seth Misenar, and Joshua Feldman. "Chapter 4 - Domain 4: Communication and network security". In: *Eleventh Hour CISSP® (Third Edition)*. Syngress, Jan. 2017, pp. 95–116. ISBN: 978-0-12-811248-9. DOI: 10.1016/B978-0-12-811248-9.00004-8.

[48] Costas Courcoubetis and Mihalis Yannakakis. "The complexity of probabilistic verification". In: *J. ACM* 42.4 (July 1995), pp. 857–907. ISSN: 0004-5411. DOI: 10.1145/210332.210339.

[49] *Dataplot homepage*. URL: https://www.itl.nist.gov/div898/software/dataplot/homepage.htm.

[50] Mrinal Kanti Debbarma et al. "Performance analysis of network monitoring tool through automated software engineering approach". In: *2015 International Conference on Signal Processing and Communication Engineering Systems*. 2015, pp. 402–406. DOI: 10.1109/SPACES.2015.7058294.

[51] *Defense Advanced Research Projects Agency*. URL: https://www.darpa.mil.

[52] M Delignette-Muller and C Dutang. "fitdistrplus: An R Package for Fitting Distributions". In: *Journal of Statistical Software* 64.4 (2015), pp. 1–34.

[53] *DOMAIN NAMES*. URL: https://datatracker.ietf.org/doc/html/rfc1034.

[54] *draft-asaeda-icnrg-contrace-02*. URL: https://datatracker.ietf.org/doc/html/draft-asaeda-icnrg-contrace-02.

[55] Robert Dumonceaux and Charles E. Antle. "Discrimination Between the Log-Normal and the Weibull Distributions". In: *Technometrics* 15.4 (1973), pp. 923–926.

[56] Robert Dumonceaux, Charles E. Antle, and Gerald Haas. "Likelihood Ratio Test for discrimination between two models with unknown scale and location parameters". In: *Technometrics* 15.1 (1973), p. 19.

[57] Nick Feamster, Jennifer Rexford, and Ellen Zegura. "The road to SDN: an intellectual history of programmable networks". In: *SIGCOMM Comput. Commun. Rev.* 44.2 (Apr. 2014), pp. 87–98. ISSN: 0146-4833. DOI: 10.1145/2602204.2602219.

[58] *FILE TRANSFER PROTOCOL (FTP)*. URL: https://datatracker.ietf.org/doc/html/rfc959.

[59] James J. Filliben. *4-Plot*. National Institute of Standards and Technology, 2003. URL: %7Bhttps://www.itl.nist.gov/div898/handbook/eda/section3/4plot.htm%7D.

[60] James J. Filliben. *Maximum Likelihood*. National Institute of Standards and Technology, 2003. URL: %7Bhttps://www.itl.nist.gov/div898/handbook/eda/section3/eda3652.htm%7D.

[61] James J. Filliben. *Mean Plot*. National Institute of Standards and Technology, 2003. URL: %7Bhttp://web.archive.org/web/20180217195200/http://www.itl.nist.gov/div898/handbook/eda/section3/dexmeanp.htm%7D.

[62] James J. Filliben. "The Probability Plot Correlation Coefficient Test for Normality". In: (1975).

[63] James J. Filliben and Alan N. Heckert. *Dataplot*. National Institute of Standards and Technology, 1978. URL: %7Bhttp://web.archive.org/web/20190819195854/https://www.itl.nist.gov/div898/software/dataplot/%7D.

[64] *Floodlight Controller - Project Floodlight*. URL: https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/overview.

[65] M. Fujita, P. C. McGeer, and J. C.-Y. Yang. "Multi-Terminal Binary Decision Diagrams: An Efficient Data Structure for Matrix Representation". In: *Formal Methods in System Design* 10.2 (Apr. 1997), pp. 149–169. ISSN: 1572-8102. DOI: 10.1023/A:1008647823331.

[66] Fabrizio Gabbiani and Steven J. Cox. "CHAPTER 11 - Probability and Random Variables". In: *Mathematics for Neuroscientists*. Cambridge, MA, USA: Academic Press, Jan. 2010, pp. 155–173. ISBN: 978-0-12-374882-9. DOI: 10.1016/B978-0-12-374882-9.00011-3.

[67] Chavoosh Ghasemi, Hamed Yousefi, and Beichuan Zhang. "Internet-Scale Video Streaming over NDN". In: *arXiv* (Aug. 2020). eprint: 2008.02752. URL: https://arxiv.org/abs/2008.02752v1.

[68] Jerry Glowniak. "History, structure, and function of the Internet". In: *Semin. Nucl. Med.* 28.2 (Apr. 1998), pp. 135–144. ISSN: 0001-2998. DOI: 10.1016/S0001-2998(98)80003-2.

[69] B. Goode. "Voice over Internet protocol (VoIP)". In: *Proceedings of the IEEE* 90.9 (2002), pp. 1495–1517. DOI: 10.1109/JPROC.2002.802005.

[70] Paul Göransson, Chuck Black, and Timothy Culver. "Chapter 12 - SDN Applications". In: *Software Defined Networks (Second Edition)*. Morgan Kaufmann, Jan. 2017, pp. 271–301. ISBN: 978-0-12-804555-8. DOI: 10.1016/B978-0-12-804555-8.00012-0.

[71] Konstantinos Gotsis, Sotirios Goudos, and J.N. Sahalos. "A Test Lab for the Performance Analysis of TCP Over Ethernet LAN on Windows Operating System". In: *Education, IEEE Transactions on* 48 (June 2005), pp. 318–328. DOI: 10.1109/TE.2004.842897.

[72] Khalid Halba and Charif Mahmoudi. "In-Vehicle Software Defined Networking: An Enabler for Data Interoperability". In: ICISDM '18. Lakeland, FL, USA: Association for Computing Machinery, 2018, pp. 93–97. ISBN: 9781450363549. DOI: 10.1145/3206098.3206105. URL: https://doi.org/10.1145/3206098.3206105.

[73] Yong-Qi Han et al. "Research of Network Monitoring Based on SNMP". In: *2013 Third International Conference on Instrumentation, Measurement, Computer, Communication and Control*. 2013, pp. 411–414. DOI: 10.1109/IMCCC.2013.94.

[74] Andreas Hanemann et al. "A study on network performance metrics and their composition". In: *Campus-Wide Information Systems* (Aug. 2006). DOI: 10.1108/10650740610704135.

[75] John Harrison. "HOL Light: An Overview". In: *Theorem Proving in Higher Order Logics*. Berlin, Germany: Springer, Aug. 2009, pp. 60–66. ISBN: 978-3-642-03358-2. DOI: 10.1007/978-3-642-03359-9_4.

[76] Hossein Hassani and Mohammad Reza Yeganegi. "Selecting optimal lag order in Ljung–Box test". In: *Physica A* 541 (Mar. 2020), p. 123700. ISSN: 0378-4371. DOI: 10.1016/j.physa.2019.123700.

[77] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. New York, NY, USA: Springer, New York, NY, 2009. DOI: 10.1007/978-0-387-84858-7.

[78] John Heath et al. "Probabilistic model checking of complex biological pathways". In: *Theoret. Comput. Sci.* 391.3 (Feb. 2008), pp. 239–257. ISSN: 0304-3975. DOI: 10.1016/j.tcs.2007.11.013.

[79] T. A. Henzinger et al. "Symbolic Model Checking for Real-Time Systems". In: *Inform. And Comput.* 111.2 (June 1994), pp. 193–244. ISSN: 0890-5401. DOI: 10.1006/inco.1994.1045.

[80] T. A. Henzinger et al. "Symbolic model checking for real-time systems". In: *[1992] Proceedings of the Seventh Annual IEEE Symposium on Logic in Computer Science*. IEEE, June 1992, pp. 394–406. ISBN: 978-0-8186-2735. DOI: 10.1109/LICS.1992.185551.

[81] T. Hérault et al. "Approximate Probabilistic Model Checking". In: *International Conference on Verification, Model Checking, and Abstract Interpretation, VMCAI'04*. Jan. 2004, pp. 73–84.

[82] Thomas Hérault et al. "Approximate Probabilistic Model Checking". In: *Verification, Model Checking, and Abstract Interpretation*. Berlin, Germany: Springer, Jan. 2004, pp. 73–84. ISBN: 978-3-540-20803-7. DOI: 10.1007/978-3-540-24622-0_8.

[83] Holger Hermanns, Björn Wachter, and Lijun Zhang. "Probabilistic CEGAR". In: *Computer Aided Verification*. Berlin, Germany: Springer, July 2008, pp. 162–175. ISBN: 978-3-540-70543-7. DOI: 10.1007/978-3-540-70545-1_16.

[84] Wassily Hoeffding. "Probability Inequalities for Sums of Bounded Random Variables". In: *J. Am. Stat. Assoc.* 58.301 (Mar. 1963), pp. 13–30. ISSN: 0162-1459. URL: http://www.jstor.org/stable/2282952.

[85] Katharina Hofer-Schmitz and Branka Stojanović. "Towards formal verification of IoT protocols: A Review". In: *Comput. Networks* 174 (June 2020), p. 107233. ISSN: 1389-1286. DOI: 10.1016/j.comnet.2020.107233.

[86] *Hypertext Transfer Protocol – HTTP/1.1*. URL: https://datatracker.ietf.org/doc/html/rfc2616.

[87] *ICN Ping Protocol Specification*. 2017. URL: https://tools.ietf.org/id/draft-mastorakis-icnrg-icnping-02.html.

[88] *IEEE 802.11, The Working Group Setting the Standards for Wireless LANs*. URL: https://www.ieee802.org/11.

[89] *IEEE 802.3-2018 - IEEE Standard for Ethernet*. URL: https://standards.ieee.org/standard/802_3-2018.html.

[90] *Internet - Foundation of the Internet*. URL: https://www.britannica.com/technology/Internet/Foundation-of-the-Internet.

[91] *Internet Corporation for Assigned Names and Numbers (ICANN)*. URL: https://www.icann.org.

[92] *Internet Engineering Task Force (IETF)*. URL: https://www.ietf.org/about.

[93] *Internet Protocol*. URL: https://datatracker.ietf.org/doc/html/rfc791.

[94] Md. Tariqul Islam, Nazrul Islam, and Md. Al Refat. "Node to Node Performance Evaluation through RYU SDN Controller". In: *Wireless Pers. Commun.* 112.1 (May 2020), pp. 555–570. ISSN: 1572-834X. DOI: 10.1007/s11277-020-07060-4.

[95] Van Jacobson et al. *Networking Named Content*. 2009. ISBN: 9781605586366. URL: https://named-data.net/wp-content/uploads/Jacob.pdf.

[96] Michel Jambu. "Chapter 3 - 1-D Statistical Data Analysis". In: *Exploratory and Multivariate Data Analysis*. Cambridge, MA, USA: Academic Press, Jan. 1991, pp. 27–62. ISBN: 978-0-12-380090-9. DOI: 10.1016/B978-0-08-092367-3.50007-1.

[97] *James J Filliben*. URL: https://www.nist.gov/people/james-j-filliben.

[98] David N. Jansen et al. "How fast and fat is your probabilistic model checker? an experimental performance comparison". In: *HVC'07: Proceedings of the 3rd international Haifa verification conference on Hardware and software: verification and testing*. Berlin, Germany: Springer-Verlag, Oct. 2007, pp. 69–85. ISBN: 978-354077964. DOI: 10.5555/1787497.1787510.

[99] J. Jubin and J.D. Tornow. "The DARPA packet radio network protocols". In: *Proceedings of the IEEE* 75.1 (1987), pp. 21–32. DOI: 10.1109/PROC.1987.13702.

[100] David Kahaner et al. *Numerical Methods and Software*. Upper Saddle River, NJ, USA: Prentice Hall, 1988. ISBN: 978-0-13627258-8. URL: https://books.google.com/books?id=jipEAQAAIAAJ&newbks=0&hl=en&source=newbks_fb.

[101] Joost-Pieter Katoen et al. "The ins and outs of the probabilistic model checker MRMC". In: *Perform. Eval.* 68.2 (Feb. 2011), pp. 90–104. ISSN: 0166-5316. DOI: 10.1016/j.peva.2010.04.001.

[102] Joost-Pieter Katoen et al. "Three-valued abstraction for probabilistic systems". In: *Journal of Logic and Algebraic Programming* 81.4 (May 2012), pp. 356–389. ISSN: 1567-8326. DOI: 10.1016/j.jlap.2012.03.007.

[103] Christoph Kern and Mark R. Greenstreet. "Formal verification in hardware design: a survey". In: *ACM Trans. Des. Autom. Electron. Syst.* 4.2 (Apr. 1999), pp. 123–193. ISSN: 1084-4309. DOI: 10.1145/307988.307989.

[104] *Kernel Density Plot*. URL: https://www.itl.nist.gov/div898/software/dataplot/refman1/auxillar/kernplot.htm.

[105] Muhammad Jawad Khokhar, Thibaut Ehlinger, and Chadi Barakat. "From Network Traffic Measurements to QoE for Internet Video". In: *2019 IFIP Networking Conference (IFIP Networking)*. 2019, pp. 1–9. DOI: 10.23919/IFIPNetworking.2019.8816854.

[106] Siham Khoussi. "Some real data for testing". In: *GitHub repository* (2021).

[107] Siham Khoussi et al. "A neural networks-based methodology for fitting data to probability distributions". In: *2021 IEEE/ACS 18th International Conference on Computer Systems and Applications (AICCSA)*. 2021, pp. 1–7. DOI: 10.1109/AICCSA53542.2021.9686821.

[108] Siham Khoussi et al. "NDN-trace: a path tracing utility for named data networking". In: *ICN '17: Proceedings of the 4th ACM Conference on Information-Centric Networking*. New York, NY, USA: Association for Computing Machinery, Sept. 2017, pp. 116–122. ISBN: 978-1-45035122-5. DOI: 10.1145/3125719.3125738.

[109] Siham Khoussi et al. "Neural networks for classifying probability distributions". In: *NIST* (2021). DOI: 10.6028/NIST.TN.2152. URL: https://nvlpubs.nist.gov/nistpubs/TechnicalNotes/NIST.TN.2152.pdf.

[110] Siham Khoussi et al. "Performance evaluation of a NDN forwarder using statistical model checking". In: *CoRR* abs/1905.01607 (2019). arXiv: 1905.01607. URL: http://arxiv.org/abs/1905.01607.

[111] Siham Khoussi et al. “Performance Evaluation of the NDN Data Plane Using Statistical Model Checking”. In: *Automated Technology for Verification and Analysis*. Ed. by Yu-Fang Chen, Chih-Hong Cheng, and Javier Esparza. Cham: Springer International Publishing, 2019, pp. 534–550.

[112] Siham Khoussi et al. “Performance Evaluation of the NDN Data Plane Using Statistical Model Checking”. In: *Automated Technology for Verification and Analysis*. Cham, Switzerland: Springer, Oct. 2019, pp. 534–550. ISBN: 978-3-030-31783-6. DOI: 10.1007/978-3-030-31784-3_31.

[113] Shinji Kikuchi and Yasuhide Matsumoto. “Performance Modeling of Concurrent Live Migration Operations in Cloud Computing Systems Using PRISM Probabilistic Model Checker”. In: *2011 IEEE 4th International Conference on Cloud Computing*. IEEE, pp. 4–9. DOI: 10.1109/CLOUD.2011.48.

[114] P.T. Kirstein. “Early experiences with the Arpanet and Internet in the United Kingdom”. In: *IEEE Annals of the History of Computing* 21.1 (1999), pp. 38–44. DOI: 10.1109/85.759368.

[115] G. V. Kobyz and A. V. Zamyatin. “Conditional probability density estimation using artificial neural network”. In: (2015), pp. 441–445.

[116] Teemu Koponen et al. “A data-oriented (and beyond) network architecture”. In: *SIGCOMM Comput. Commun. Rev.* 37.4 (Aug. 2007), pp. 181–192. ISSN: 0146-4833. DOI: 10.1145/1282427.1282402.

[117] Heiko Koziolek, Bastian Schlich, and Carlos Bilich. “A Large-Scale Industrial Case Study on Architecture-Based Software Reliability Analysis”. In: *2010 IEEE 21st International Symposium on Software Reliability Engineering*. 2010, pp. 279–288.

[118] Valeriyi Kuzmin et al. “Method of Probability Distribution Fitting for Statistical Data with Small Sample Size”. In: *2020 10th International Conference on Advanced Computer Information Technologies (ACIT)*. IEEE, Sept. 2020, pp. 221–224. DOI: 10.1109/ACIT49673.2020.9208842.

[119] Marta Kwiatkowska, Gethin Norman, and David Parker. “Symmetry Reduction for Probabilistic Model Checking”. In: *Computer Aided Verification*. Berlin, Germany: Springer, Aug. 2006, pp. 234–248. ISBN: 978-3-540-37406-0. DOI: 10.1007/11817963_23.

[120] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. *PRISM 4.0: Verification of Probabilistic Real-time Systems*. Vol. 6806. July 2011. ISBN: 978-3-642-22109-5. DOI: 10.1007/978-3-642-22110-1_47.

[121] Marta Z. Kwiatkowska, Gethin Norman, and Jeremy Sproston. “Probabilistic Model Checking of the IEEE 802.11 Wireless Local Area Network Protocol”. In: *PAPM-PROBMIV ’02: Proceedings of the Second Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification*. Berlin, Germany: Springer-Verlag, July 2002, pp. 169–187. ISBN: 978-354043913. DOI: 10.5555/645777.668436.

[122] *Labelled Transition System - an overview | ScienceDirect Topics*. URL: https://www.sciencedirect.com/topics/mathematics/labelled-transition-system.

[123] Sophie Laplante et al. *Probabilistic Abstraction for Model Checking: An Approach Based on Property Testing*. IEEE Computer Society, July 2002. ISBN: 978-0-7695-1483. DOI: 10.1109/LICS.2002.1029815.

[124] Richard Lassaigne and Sylvain Peyronnet. "Approximate Verification of Probabilistic Systems". In: *PAPM-PROBMIV '02: Proceedings of the Second Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification*. Berlin, Germany: Springer-Verlag, July 2002, pp. 213–214. ISBN: 978-354043913. DOI: 10.5555/645777.668441.

[125] A. M. Law and M. G. McComas. "How the ExpertFit distribution-fitting software can make your simulation models more valid". In: *Proceedings of the Winter Simulation Conference* 1 (2011), pp. 199–204.

[126] Axel Legay, Benoıt Delahaye, and Saddek Bensalem. "Statistical Model Checking: An Overview". In: *Runtime Verification*. Berlin, Germany: Springer, Nov. 2010, pp. 122–135. ISBN: 978-3-642-16611-2. DOI: 10.1007/978-3-642-16612-9_11.

[127] Alexios Lekidis, Marius Bozga, and Saddek Bensalem. "Model-based validation of CANopen systems". In: May 2014. DOI: 10.1109/WFCS.2014.6837602.

[128] Alexios Lekidis et al. "A model-based design flow for CAN-based systems". In: Nov. 2013.

[129] Teng Liang, Ju Pan, and Beichuan Zhang. "NDNizing existing applications: research issues and experiences". In: *ICN '18: Proceedings of the 5th ACM Conference on Information-Centric Networking*. New York, NY, USA: Association for Computing Machinery, Sept. 2018, pp. 172–183. ISBN: 978-1-45035959-7. DOI: 10.1145/3267955.3267969.

[130] Aristidis Likas. "Probability density estimation using artificial neural networks". In: *Computer Physics Communications* 135 (2001), pp. 167–175.

[131] Stephen Lukasik. "Why the Arpanet Was Built". In: *IEEE Annals of the History of Computing* 33.3 (2011), pp. 4–21. DOI: 10.1109/MAHC.2010.11.

[132] N Hastings M Evans and B Peacock. "Statistical Distributions, Third Edition". In: *Measurement Science and Technology* 12.1 (Dec. 2000), pp. 117–117. DOI: 10.1088/0957-0233/12/1/702. URL: https://doi.org/10.1088/0957-0233/12/1/702.

[133] Rajib Maity. "Probability Distributions and Their Applications". In: *Statistical Methods in Hydrology and Hydroclimatology*. Singapore: Springer, May 2018, pp. 93–143. ISBN: 978-981-10-8778-3. DOI: 10.1007/978-981-10-8779-0_4.

[134] Anggi Mardiyono, Walidatush Sholihah, and Faisal Hakim. "Mobile-based Network Monitoring System Using Zabbix and Telegram". In: *2020 3rd International Conference on Computer and Informatics Engineering (IC2IE)*. 2020, pp. 473–477. DOI: 10.1109/IC2IE50715.2020.9274582.

[135] Dan Marinescu. *Cloud Computing*. Morgan Kaufmann, Nov. 2017. ISBN: 978-0-12812810-7. URL: https://www.elsevier.com/books/cloud-computing/marinescu/978-0-12-812810-7.

[136] Baphumelele Masikisiki, Siyabulela Dyakalashe, and Mfundo Shakes Scott. "Network monitoring system for network equipment availability and performance reporting". In: *2017 IST-Africa Week Conference (IST-Africa)*. 2017, pp. 1–12. DOI: 10.23919/ISTAFRICA.2017.8102339.

[137] A. Maydeu-Olivares and C. Garcıa-Forero. "Goodness-of-Fit Testing". In: *International Encyclopedia of Education (Third Edition)*. Walthm, MA, USA: Elsevier, Jan. 2010, pp. 190–196. ISBN: 978-0-08-044894-7. DOI: 10.1016/B978-0-08-044894-7.01333-6.

[138] Catherine Meadows. "Applying Formal Methods to the Analysis of a Key Management Protocol". In: *J. Comput. Secur.* 1.1 (Jan. 1992), pp. 5–35. ISSN: 0926-227X. DOI: 10.5555/2699855.2699857.

[139] Braham Lotfi Mediouni et al. " 2.0: Statistical Model Checking Stochastic Real-Time Systems". In: *Automated Technology for Verification and Analysis - 16th International Symposium, ATVA 2018, Los Angeles, CA, USA, October 7-10, 2018, Proceedings*. 2018, pp. 536–542.

[140] Jan Medved et al. "OpenDaylight: Towards a Model-Driven SDN Controller architecture". In: *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*. 2014, pp. 1–6. DOI: 10.1109/WoWMoM.2014.6918985.

[141] David O. Meltzer and Peter C. Smith. "Theoretical Issues Relevant to the Economic Evaluation of Health Technologies". In: *Handbook of Health Economics*. Vol. 2. Walthm, MA, USA: Elsevier, Jan. 2011, pp. 433–469. DOI: 10.1016/B978-0-444-53592-4.00007-4.

[142] *Mersenne Twister - an overview | ScienceDirect Topics*. June 2021. URL: https://www.sciencedirect.com/topics/computer-science/mersenne-twister.

[143] Devang Mistry et al. "Network traffic measurement and analysis". In: *2016 IEEE Long Island Systems, Applications and Technology Conference (LISAT)*. 2016, pp. 1–7. DOI: 10.1109/LISAT.2016.7494141.

[144] Decebal Constantin Mocanu et al. "Network performance assessment with Quality of experience benchmarks". In: *10th International Conference on Network and Service Management (CNSM) and Workshop*. 2014, pp. 332–335. DOI: 10.1109/CNSM.2014.7014187.

[145] Marie-Jose Montpetit, Serge Fdida, and Jianping Wang. "Future Internet: Architectures and Protocols". In: *IEEE Communications Magazine* 57.7 (2019), pp. 12–12. DOI: 10.1109/MCOM.2019.8767072.

[146] David E. Morgan et al. "A computer network monitoring system". In: *IEEE Transactions on Software Engineering* SE-1.3 (1975), pp. 299–311. DOI: 10.1109/TSE.1975.6312855.

[147] Yoshihiro Nakamura and Osamu Hasegawa. "Non parametric density estimation based on self-organizing incremental neural network for large noisy data". In: *IEEE Transactions on Neural Networks and Learning Systems* 28.1 (2017), pp. 8–17.

[148] *Named Data Networking - Named Data Networking (NDN)*. URL: https://named-data.net/publications/named_data_networking_ccr.

[149] *Named Data Networking Project*. USA, Oct. 2010. URL: http://named-data.net/techreport/TR001ndn-proj.pdf.

[150] *ndn-tools*. URL: https://github.com/named-data/ndn-tools.

[151] VASERMANIS NECHVAL NECHVAL and MAKEEV. *Constructing shortest-length confidence intervals*. URL: https://www.researchgate.net/profile/Konstantin-Nechval/publication/267986471_CONSTRUCTING_SHORTEST-LENGTH_CONFIDENCE_INTERVALS/links/55c37b3008aeb97567400f29/CONSTRUCTING-SHORTEST-LENGTH-CONFIDENCE-INTERVALS.pdf.

[152] *NFD Developer's Guide*. Tech. rep. URL: http://named-data.net/techreports.html.

[153] *NLSR - Named Data Link State Routing Protocol — Named Data Link State Routing Protocol (NLSR) 0.6.0-8-g3781c7e documentation*. URL: https://named-data.net/doc/NLSR/current.

[154] Gethin Norman et al. "Using probabilistic model checking for dynamic power management". In: *Form. Asp. Comp.* 17.2 (Aug. 2005), pp. 160–176. ISSN: 1433-299X. DOI: 10.1007/s00165-005-0062-0.

[155] Ayoub Nouri. *BIP-SMC : A Statistical Model Checking Engine for the BIP framework*. Dec. 2017. URL: https://www-verimag.imag.fr/BIP-SMC-A-Statistical-Model-Checking.html.

[156] Ayoub Nouri. "Rigorous System-level Modeling and Performance Evaluation for Embedded System Design". PhD thesis. Grenoble, France: Université Grenoble Alpes, Apr. 2015. URL: https://hal.inria.fr/tel-01148690.

[157] Ayoub Nouri. "Rigorous System-level Modeling and Performance Evaluation for Embedded System Design." PhD thesis. Grenoble Alpes University, France, 2015.

[158] Ayoub Nouri et al. "*ASTROLABE*: A Rigorous Approach for System-Level Performance Modeling and Analysis". In: *ACM Trans. Embedded Comput. Syst.* 15.2 (2016), 31:1–31:26.

[159] Ayoub Nouri et al. "Building Faithful High-level Models and Performance Evaluation of Manycore Embedded Systems". In: *MEMOCODE*. Lausanne, Switzerland, Oct. 2014. DOI: 10.1109/MEMCOD.2014.6961864. URL: https://hal.inria.fr/hal-01087671.

[160] Ayoub Nouri et al. "Performance Evaluation of Stochastic Real-Time Systems with the SBIP Framework". In: *International Journal of Critical Computer-Based Systems* 8 (Jan. 2018). DOI: 10.1504/IJCCBS.2018.10017703.

[161] Ayoub Nouri et al. "Performance evaluation of stochastic real-time systems with the SBIP framework". In: *International Journal of Critical Computer-Based Systems* 8.3-4 (2018), pp. 340–370.

[162] Ayoub Nouri et al. "Statistical Model Checking QoS Properties of Systems with SBIP". In: *Int. J. Softw. Tools Technol. Transf. (STTT)* 17.2 (Apr. 2015), pp. 171–185. ISSN: 1433-2779.

[163] *NSF Announces Future Internet Architecture Awards*. URL: https://www.nsf.gov/news/news_summ.jsp?cntn_id=117611.

[164] *NSF Future Internet Architecture Project*. June 2016. URL: http://www.nets-fia.net.

[165] *PACKET SATELLITE TECHNOLOGY REFERENCE SOURCES*. URL: https://datatracker.ietf.org/doc/html/rfc829.

[166] Larry L. Peterson and Bruce S. Davie. "1 - Foundation". In: *Computer Networks (Fifth Edition)*. Ed. by Larry L. Peterson and Bruce S. Davie. Fifth Edition. The Morgan Kaufmann Series in Networking. Boston: Morgan Kaufmann, 2012, pp. 1–69. ISBN: 978-0-12-385059-1. DOI: https://doi.org/10.1016/B978-0-12-385059-1.00001-6. URL: https://www.sciencedirect.com/science/article/pii/B9780123850591000016.

[167] Amir Pnueli, Jessie Xu, and Lenore D. Zuck. "Liveness with (0, 1, infty)-Counter Abstraction". In: *CAV '02: Proceedings of the 14th International Conference on Computer Aided Verification*. Berlin, Germany: Springer-Verlag, July 2002, pp. 107–122. ISBN: 978-354043997. DOI: 10.5555/647771.734286.

[168] Amir Pnueli and Lenore D. Zuck. "Probabilistic verification". In: *Inform. And Comput.* 103.1 (Mar. 1993), pp. 1–29. ISSN: 0890-5401. DOI: 10.1006/inco.1993.1012.

[169] *Probability Distribution Function - an overview | ScienceDirect Topics*. URL: https://www.sciencedirect.com/topics/mathematics/probability-distribution-function.

[170] Junaid Qadir and Osman Hasan. "Applying Formal Methods to Networking: Theory, Techniques, and Applications". In: *IEEE Communications Surveys Tutorials* 17.1 (2015), pp. 256–291. DOI: 10.1109/COMST.2014.2345792.

[171] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2013. URL: http://www.R-project.org/.

[172] L.R. Rabiner. "A tutorial on hidden Markov models and selected applications in speech recognition". In: *Proceedings of the IEEE* 77.2 (1989), pp. 257–286. DOI: 10.1109/5.18626.

[173] Leonardo Reyneri, Valentina Colla, and Marco Vannucci. "Estimate of a probability density function through neural networks". In: 6691.1 (2011), pp. 57–64.

[174] Jonas Rothfuss et al. "Conditional Density Estimation with Neural Networks: Best Practices and Benchmarks". In: (2019). URL: %7Bhttp://arxiv.org/abs/1903.00954%7D.

[175] J. Rutten et al. *Mathematical Techniques for Analyzing Concurrent and Probabilistic Systems*, P. Panangaden and F. van Breugel (eds.) Vol. 23. CRM Monograph Series. American Mathematical Society, 2004.

[176] Caitlin Sadowski et al. "Lessons from Building Static Analysis Tools at Google". In: *Communications of the ACM (CACM)* 61 Issue 4 (2018), pp. 58–66. URL: https://dl.acm.org/citation.cfm?id=3188720.

[177] K. Schittkowski. "EASY-FIT: a software system for data fitting in dynamical systems". In: *Structural and Multidisciplinary Optimization* 23 (2002), pp. 153–169.

[178] *SDN Applications | Elsevier Enhanced Reader*. DOI: 10.1016/B978-0-12-804555-8.00012-0.

[179] *SDN Series Part Three: NOX, the Original OpenFlow Controller - The New Stack*. URL: https://thenewstack.io/sdn-series-part-iii-nox-the-original-openflow-controller.

[180] Erik Seligman, Tom Schubert, and M. V. Achutha Kiran Kumar. "Chapter 1 - Formal verification: From dreams to reality". In: *Formal Verification*. Morgan Kaufmann, Jan. 2015, pp. 1–22. ISBN: 978-0-12-800727-3. DOI: 10.1016/B978-0-12-800727-3.00001-0.

[181] Koushik Sen, Mahesh Viswanathan, and Gul Agha. "Statistical Model Checking of Black-Box Probabilistic Systems". In: *Computer Aided Verification*. Berlin, Germany: Springer, July 2004, pp. 202–215. ISBN: 978-3-540-22342-9. DOI: 10.1007/978-3-540-27813-9_16.

[182] Anita Shinde and S. M. Chaware. "Content Centric Networks (CCN): A Survey". In: *2018 2nd International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), 2018 2nd International Conference on*. IEEE, Aug. 2018, pp. 595–598. DOI: 10.1109/I-SMAC.2018.8653769.

[183] Vitaly Shmatikov. "Probabilistic analysis of an anonymity system". In: *J. Comput. Secur.* 12.3,4 (May 2004), pp. 355–377. ISSN: 0926-227X. DOI: 10.5555/1297352.1297359.

[184] B. W. Silverman. *Kernel Density Estimation Using the Fast Fourier Transform*. 1982. URL: https://rss.onlinelibrary.wiley.com/doi/epdf/10.2307/2347084.

[185] *Simple Mail Transfer Protocol*. URL: https://datatracker.ietf.org/doc/html/rfc5321.

[186] Oktay Simsek and Marco Pospiech. "A network performance measurement tool". In: *2013 5th IEEE International Conference on Broadband Network Multimedia Technology*. 2013, pp. 45–48. DOI: 10.1109/ICBNMT.2013.6823912.

[187] *Single Sample Acceptance Plan*. URL: https://www.itl.nist.gov/div898/software/dataplot/refman1/auxillar/singsamp.htm.

[188] M. A. Stephens. "EDF Statistics for Goodness of Fit and Some Comparisons". In: *Journal of the American Statistical Association* 69.347 (1974), p. 730.

[189] *The Evolution of Packet Switching*. URL: https://web.archive.org/web/20160324033133/http://www.packet.cc/files/ev-packet-sw.html.

[190] *The evolution of Software Defined Networking*. URL: https://www.redhat.com/en/blog/evolution-software-defined-networking.

[191] F. A. Tobagi, R. Binder, and B. Leiner. "Packet radio and satellite networks". In: *IEEE Commun. Mag.* 22 (Nov. 1984), pp. 24–40. ISSN: 0163-6804. URL: https://ui.adsabs.harvard.edu/abs/1984IComM..22...24T/abstract.

[192] Claudio Topdcic and Joachim Kaiser. "The SATNET Monitoring System". In: *MILCOM 1985 - IEEE Military Communications Conference*. Vol. 2. 1985, pp. 468–476. DOI: 10.1109/MILCOM.1985.4795070.

[193] *Traceroute Using an IP Option*. URL: https://datatracker.ietf.org/doc/html/rfc1393.

[194] *TRANSMISSION CONTROL PROTOCOL*. URL: https://datatracker.ietf.org/doc/html/rfc793.

[195] *Tutorial BIP2 2015.04 (RC7) documentation*. Apr. 2015. URL: https://www-verimag.imag.fr/TOOLS/DCS/bip/doc/latest/html/tutorial.html#hello-world.

[196] Kazuaki Ueda et al. "Demo: Dynamic adaptive streaming over NDN using explicit congestion feedback". In: *2017 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*. 2017, pp. 1–2. DOI: 10.1109/LANMAN.2017.7972182.

[197] "Understanding and Choosing the Right Probability Distributions". In: *Advanced Analytical Models*. Chichester, England, UK: John Wiley & Sons, Ltd, Oct. 2015, pp. 899–917. DOI: 10.1002/9781119197096.app03.

[198] *User Datagram Protocol*. URL: https://datatracker.ietf.org/doc/html/rfc768.

[199] Moshe Y. Vardi. "Automatic verification of probabilistic concurrent finite state programs". In: *SFCS '85: Proceedings of the 26th Annual Symposium on Foundations of Computer Science*. USA: IEEE Computer Society, Oct. 1985, pp. 327–338. ISBN: 978-081860844. DOI: 10.1109/SFCS.1985.12.

[200] A. Wald. "Sequential Tests of Statistical Hypotheses". In: *Breakthroughs in Statistics: Foundations and Basic Theory*. New York, NY, USA: Springer, New York, NY, 1992, pp. 256–298. ISBN: 978-0-387-94037-3. DOI: 10.1007/978-1-4612-0919-5_18.

[201] Anduo Wang et al. "Formally Verifiable Networking". In: (2009). URL: https://www.semanticscholar.org/paper/Formally-Verifiable-Networking-Wang-Jia/4a3c5ff896c27697c19dedcfa01f10cb6698ce9e.

[202] Wambura Wasira. *Formal Verification Methods*. Feb. 2020. DOI: 10.1007/3-540-.

[203] Karl Pichotta Wesley Tansey and James G. Scott. "Better Conditional Density Estimation for Neural Networks". In: (2016). URL: %7Bhttps://arxiv.org/abs/1606.02321%7D.

[204] *What are Autonomous System Numbers (ASN) for Internet? | ThousandEyes*. URL: https://www.thousandeyes.com/learning/glossary/as-autonomous-system.

[205] *What is a Probability Distribution*. Jan. 2018. URL: https://www.itl.nist.gov/div898/handbook/eda/section3/eda361.htm.

[206] G. Xylomenos et al. "A Survey of Information-Centric Networking Research". In: *IEEE Communications Surveys Tutorials* 16.2 (2014), pp. 1024–1049.

[207] H. L. S. Younes. "Verification and Planning for Stochastic Processes with Asynchronous Events". PhD thesis. Carnegie Mellon, 2005.

[208] Håkan L. S. Younes and R. Simmons. "Verification and planning for stochastic processes with asynchronous events". In: 2004.

[209] Håkan L. S. Younes and Reid G. Simmons. "Probabilistic Verification of Discrete Event Systems Using Acceptance Sampling". In: *Computer Aided Verification*. Berlin, Germany: Springer, Sept. 2002, pp. 223–235. ISBN: 978-3-540-43997-4. DOI: 10.1007/3-540-45657-0_17.

[210] Hakan Lorens Samir Younes and Reid G. Simmons. "Verification and Planning for Stochastic Processes with Asynchronous Events". AAI3159989. PhD thesis. USA, 2004. ISBN: 0496934759.

[211] Artem Yushkovskiy. "Comparison of Two Theorem Provers: Isabelle/HOL and Coq". In: *arXiv* (Aug. 2018). eprint: 1808.09701. URL: https://arxiv.org/abs/1808.09701v2.

[212] Haitao Zhang. *NDNFit: An Open mHealth Application Built on Named Data Networking*. 2018. URL: https://escholarship.org/uc/item/8h8950n3.

[213] Lixia Zhang et al. "Named Data Networking". In: *SIGCOMM Comput. Commun. Rev.* 44.3 (July 2014), pp. 66–73. ISSN: 0146-4833.