A Deep Neural Network based Multi-Label Classifier for SLA Violation Prediction in a Latency Sensitive NFV Application

Nikita Jalodia, Mohit Taneja, *Member, IEEE*, Alan Davy, *Senior Member, IEEE*

Recent advancements in the domain of Network Function Virtualization (NFV), and rollout of next-generation networks have necessitated the requirement for the upkeep of latency-critical application architectures in future networks and communications. While Cloud service providers recognize the evolving mission-critical requirements in latency sensitive verticals such as autonomous driving, multimedia, gaming, telecommunications, and virtual reality, there is a wide gap to bridge the Quality of Service (QoS) constraints for the end-user experience. Most latency-critical services are over-provisioned on all fronts to offer reliability, which is inefficient towards scalability in the long run. To address this, we propose a strategy to model frequent violations on the application level as a multi-output target to enable more complex decision-making in the management of virtualised communication networks. In this work, we utilize data from a real-world deployment to configure and draft a realistic set of Service Level Objectives (SLOs) for a voice based NFV application, and develop a deep neural network based multi-label classification methodology to identify and predict multiple categories of SLO breaches associated with an application state. With this, we aim to gain granular SLA and SLO violation insights, enabling us to study and mitigate their impact and inform precision in drafting proactive scaling policies. We further compare the performance against a set of multi-label compatible machine learning classifiers, and address class imbalance in a multi-label setup. We perform a comprehensive evaluation to assess the performance on example-based, label-based and ranking-based measures, and demonstrate the suitability of deep learning in such a use-case.

***Index Terms*—Network function virtualization, machine learning, deep learning, neural networks, classification algorithms, multi-label classification, prediction methods, quality of service, service level agreements, quality of experience, supervised learning, artificial neural networks, multi-layer neural network, naive bayes methods, random forests, decision trees, boosting, support vector machines, imbalanced classification, probabilistic classification**

I. INTRODUCTION

THE next generation of networks hold a vision to expand communications from the scale of billions in the world's population to a virtually limitless scale of inter-connectivity between humans, machines, and things. As a result, we are facing a paradigm of exponential growth in enhanced services and applications, network traffic, and consumers. The global mobile traffic is expected to reach 5016 Exabytes (EB) per month in 2030 [1], which is an explosive surge as compared to 51 EB per month in 2020 [2]. Both supporting and driving this demand, the next generation of communication networks continue to be driven by a fundamental restructuring in the way that the networks and services are deployed and delivered. Network programmability and softwarisation are the key drivers of this change, and are delivered via the concepts of Software Defined Networking (SDN) and Network Function Virtualization (NFV) [3]. These continue to play a pivotal role in the vision of 6G, forming the backbone of flexible and intelligent networks [4]. SDN abstracts the underlying network while NFV introduces softwarisation and decouples network functions from the underlying hardware, overall creating a hardware agnostic virtualized environment for network applications [5]. This shift has opened the market to a wider movement towards virtualised applications and services in key verticals such as automatic vehicles, smart grid, virtual reality (VR), internet of things (IoT), industry 4.0, etc., and also includes verticals that previously relied solely on specialised hardware. A key example of such a sector is the telecommunications industry, which is driven by one of the oldest and most complex operational and business support systems to date [4].

Traditionally, with its specialised infrastructure, the telecoms realm has evolved towards a highly reliant service, with carrier-grade offerings guaranteeing a five-nines standard of availability [6]. However, with the emergence of such agile and flexible paradigms as enabled with the coupling of SDN and NFV, we are seeing an emergence of a new era of applications driven by the vision of low latency and high reliability [4]. 5G's usage scenario of ultra-reliable low-latency communications (URLLC) is further expected to extend in scope to a high-throughput ubiquitous global connectivity at scale, driving all major verticals towards a change [4]. As a result of such a shift, the Cloud infrastructure is no longer host to just web based application services, but is also being extended for the next-generation of requirements that fuel these futuristic application verticals [3]. A key aspect to driving such a change is in how the Cloud reacts to such a latency-critical demand, and in being precisely proactive over time [7].

While static threshold based scaling is still the most dominant scaling policy in use for most systems on the Internet,

This paragraph of the first footnote will contain the date on which you submitted your paper for review.

This work has emanated from research conducted with the financial support of Science Foundation Ireland (SFI) and is co-funded under the European Regional Development Fund under Grant Number 13/RC/2077.

N. Jalodia is with the Emerging Networks Lab Research Division, Walton Institute of Information and Communication Systems Science, Waterford Institute of Technology, Waterford, X91WR86, Ireland. She is also affiliated with CONNECT- Centre for Future Networks and Communications, Dublin, Ireland. (*Corresponding Author; e-mail:* nikita.jalodia@waltoninstitute.ie).

M. Taneja is with the Department of Accountancy and Economics, School of Business at Waterford Institute of Technology, Waterford, X91 TX03, Ireland.

A. Davy is with the Department of Computing and Mathematics, Waterford Institute of Technology, Waterford, X91K0EK, Ireland. He is also affiliated with CONNECT, and the Walton Institute of Information and Communication Systems Science, Waterford Institute of Technology, Waterford, X91WR86, Ireland

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/OJCOMS.2021.3122844, IEEE Open Journal of the Communications Society

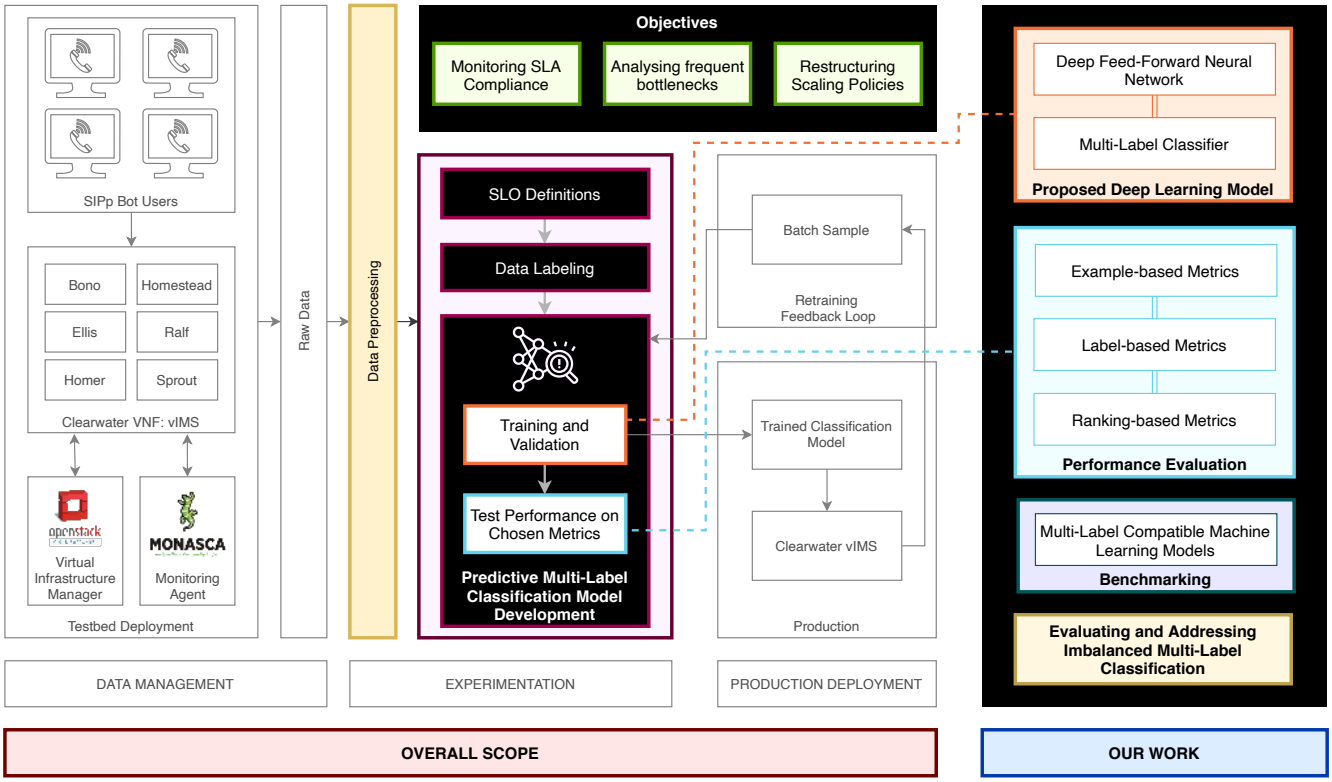


Fig. 1: Overview of system architecture, objectives, and scope.

some Cloud service providers also offer dynamic scaling mechanisms [8]. These have more flexibility on the choice of threshold based on a pre-defined range, and real time network traffic. While such policies offer better adaptation to meeting desired QoS levels, they still do not match the elevated service requirements facing the ebb and flow in network traffic and demand. Acknowledging these shortcomings, some Cloud service providers now also provide an upgrade in the form of predictive scaling policies [9], [10], [11]. These are based towards analysing the traffic and key high-level system usage metrics over time, and increasing the system resources during regularly anticipated patterns of high incoming traffic.

While service operators come up with new scaling policies to match the demand facing the current generation of Cloud based application services, these are still a long way to go towards supporting latency-critical applications with high availability values. Since the requirements of real-time applications such as voice and multimedia communication differ from the traditional web-based applications that the Cloud supports, so do the Service Level Agreements (SLAs). A momentary increase in latency and jitter in a voice application has an immediate influence on the end-user Quality of Experience (QoE), while not quite so in conventional web applications [12]. A key challenge highlighted in the conceptualization of 6G is to impose stringent end-to-end QoS requirements within heterogeneous services [4]. To this end, 5G deployments include the proposition of network slicing, clustering applications with similar demands in an appropriate Cloud environment [5]. This ensures the placement of latency-critical URLLC applications in a high availability slice, where resources are suitably provisioned to ensure reliability. However, 6G expects an improvement o reliability by atleast two orders of magnitude, i.e. from a five-nines standard to a seven-nines standard [4].

Further, efficiency and reliability are competing elements within an SLA, marking a trade-off between system usage and requirements [13]. While network operators and Cloud operators may resort to over-provisioning to match the high requirements for these latency-critical applications in a high availability network slice, such practices are inefficient in the long run [7]. 6G expects a 10 to 100 times improvement on the energy efficiency as compared to 5G [4], which is a challenge in itself. Over time, we need to be mindful of the significant carbon footprint of Cloud data-centres too, so there is more to it than just the end user experience here [3].

In this work, we take the example of a latency-critical NFV application, and draft realistic Service Level Objectives (SLOs) in a way that provide more granular insights into the violations occurring in operational settings. Such insights would help towards improving the formulation of resource provisioning policies in a way that is targeted towards the precise objectives that match service requirements of the target application use-case, rather than a blanket over-provisioning of all categories of system resources. Contrary to simplistic classification methodologies that predict a single label that categorises whether the SLA is violated at the application level, we formulate our work as a multi-label classification problem. This methodology involves training models to associate a sample of input data features with a set of labels from a bigger set of disjoint labels [14], [15], thus helping us to model

individual SLO violations associated to the application's state that contribute to an SLA violation overall. We further provide a detailed analysis on how to manage the challenges that such a system presents, including class imbalance with minority classes. We test the performance against a cohort of machine learning solutions, and present a methodical analysis towards the development and effective use of a deep learning classifier for such objectives.

To the best of our knowledge, this is the first approach in the area that applies multi-label classification towards such objectives, and formulates a methodology that combines realistic SLO definitions to predict precise QoS violations for such a latency-sensitive use-case. The key contributions are summarised as follows:

- We work with a real-world deployment of a latency critical NFV application with two months' worth of raw network telemetry data sampled every 30 seconds, and use that as the basis for all our policy formation and learning models. An overview of the system and scope is provided in Figure 1.
- We break down the SLA into a set of realistic SLO definitions for such a latency critical use-case in an operational setting. While SLA and SLO definitions are application specific, we form these measures to be as realistic as possible to capture the dynamics of a real-world deployment.
- As opposed to a single-label binary or multi-class classification objective, we associate and model a multi-label classifier to effectively predict each SLO violation that an application state is associated with. Over time, this helps us to study and mitigate frequent application and use-case specific bottlenecks, and also in predicting a more granular state of the application's behaviour as it faces a drop in QoS, and a violation in SLA.
- We test the performance of the developed model against a wide set of compatible machine learning methodologies, and provide a justified reasoning to the deployment of a deep neural network model in such a setup.
- We methodically address the challenges that come up with training such a model at scale, i.e. the associated problems of varying degrees of class imbalance in a multi-label setup.
- We evaluate the performance on a wide range of metrics that include example based, label based and ranking based measures, and provide an all-round evaluation of each learning model benchmarked.

The rest of the paper has been structured as follows: §II describes the background and related work, §III describes the Clearwater NFV application, and defines the SLA and SLOs drafted for the purpose of violation prediction. §IV provides an overview of the unique characteristics of a multi-label classification methodology, the mathematical formulation of the problem statement, the machine learning algorithms applied, and the definitions of the various metrics used for an all-round evaluation. §V addresses the prevailing issue of class imbalance in the multi-label context, and presents the methodologies we use to overcome this issue with a deep neural network model. Thereafter, §VI expands on the details of the experimental setup, §VII evaluates the results obtained through the various models, and §VIII presents the conclusion and future work.

II. Related Work

A transition towards complete softwarization of networks brings in the requirement to adopt more complex models to guarantee QoS and reliability [16]. This is because of an impending evolution in not just the way networks are composed and managed, but also renewed application architectures, corresponding QoS and SLA management techniques, and optimization and automation to cope with the added complexity [17].

Authors in [16], [18] study the impact of virtualization in fault management, and the added challenges that the distinct yet complementary paradigms of SDN and NFV bring in such a setup. Authors in [19] highlight the shift from traditionally tracking the QoS of a single service to that of service compositions in networks, and use a genetic algorithm to optimize the application reliability in the 5G network case. Further, authors in [20] quantitatively model and assess availability from a core network perspective for an end-to-end NFV enabled service. Reliability block diagrams and stochastic reward nets based approaches have also been leveraged for providing an optimal configuration of an NFV based SFC for telecommunications standards availability modeling [21], [6]. An in-depth survey [22] on the autonomic provisioning and QoS management for SDN-based networks highlights the need for more in-depth machine learning models that target and improve policy-based QoS management, and remark that assuring end-user QoE continues to be an open research area.

Much of the work done so far addresses QoS with characterizing and anticipating traffic patterns, and a combination of reactive and proactive scaling policies. Significant progress has been made in the context of forecasting and clustering anticipated network traffic [23], [24], using machine learning to classify network traffic in NFV [25], [26], and related resource allocation [7], [13].

However, post appropriate provisioning following anticipated and identified traffic patterns, there is not a lot of work that directly addresses the remaining SLA bottlenecks from an application perspective. Automated SLA management for use-cases deployed on softwarized networks has been highlighted to be a critical requirement for next generation networks [27], [12]. A theoretical SLA management framework that maps high-level requirements to low-level resource attributes is presented in [28], where the authors highlight the additional challenges that 5G and future architectures present. Authors in [29], [30] present a cognitive management architecture for these softwarized networks, and discuss the importance of machine learning techniques in such complete end-to-end management control loops. Existing work on SLA and SLO violation prediction approaches it as a single label output classification [31]— either identifying an overall SLA violation with a binary classification, or identifying a defined SLO breach with multi-class classification [32]. A proof of

concept for SLA enforcement in programmable networks in a Cloud-based environment is presented in [33], where the authors work towards identifying an SLO breach with a multi-class decision tree classification methodology. However, in a realistic scenario, there is a pressing need for the incorporation of multi-output models as we move towards more complex decision-making [15]. As future networks as well as deployed services gain complexity, it is impractical to define and consider an SLO as a mutually exclusive single-output target. There is no existing work in the area of SLA management that leverages advanced classification methodologies for a multi-output prediction target, identifying and predicting multiple categories of SLO breaches as applicable to study their impact.

To fill this gap, we propose the use of multi-label classification methodology for a multi-output SLO violation prediction in NFV environments. Multi-label classification is a branch of predictive classification models that involves training models to associate a sample of input data features with more than one class labels [14]. While the primary motivation for such models draws from the domain of text categorization, image and multimedia object interpretation, music information retrieval, movie genre classification, automated video annotation, etc., other fields such as biology and functional genomics have also leveraged multi-label classification models to address challenging research problems [34]. While initial approaches focused on machine learning based methods to handle multi-label problems [35], [36], [37], [38], there has been a recent rise in the application of several neural network architectures to address the complexity and of varied use-cases [15], [39], [40], [41], [42], [43]. Associating structured data with multiple semantic information at once holds tremendous potential in the future as we advance towards solving more complex decision making problems [15].

To the best of our knowledge, this is the first approach in the area that applies a multi-label classification methodology towards a more granular SLA violation prediction for a latency-sensitive VNF in a virtualised network environment, and works with extensive real world data to compare the performance of both machine learning and deep learning methodologies towards such an objective.

III. Defining Service Level Agreements

SLAs are closely tied to product and business definitions, and imply a formally explicit consequence upon breach of contract when the agreed terms are violated. While an SLA is a qualitative measure that binds the service provider and facilitator into a formally agreed contract ensuring QoS for the end user, this is realised on a set of low level metrics delivered through SLOs and Service Level Indicators (SLIs). The SLIs can be defined as quantitative measures that build upon raw system metrics, which further feed into the SLOs as a quantitatively definitive target range or threshold towards the deliverance of an SLA. The breach of an SLA implies an explicit consequence, often financial; while the SLOs and SLIs are typically measurable indicators that define the policy of tolerance [44].

While the SLA is a formal contract between a service provider and a consumer, it is often a high-level definition

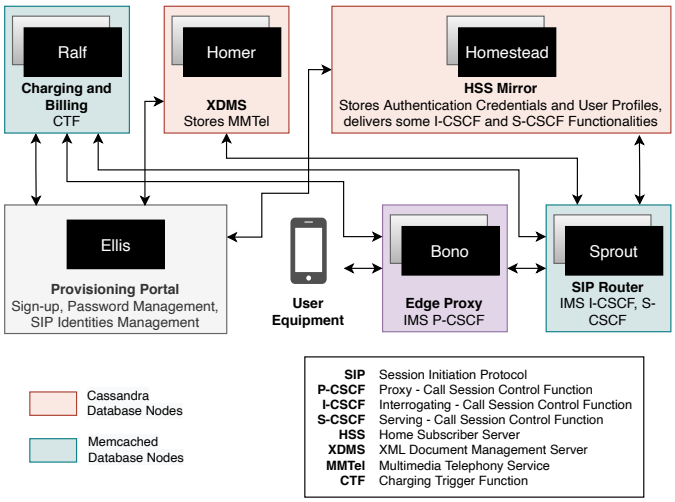


Fig. 2: Clearwater vIMS architecture, depicting the various VNFCs and their high-level functionalities.

of the service provided. From a service provider's perspective, the SLIs and SLOs are the means to that end, and imperatively define the measurable service characteristics that quantifiably deliver that quality. Therefore, the choice of SLOs are critical towards delivering the QoS promised to the end user, and vary depending on the type of application and use-case scenario.

While the end user QoE may be defined by more than just the server level QoS guarantees, the latter forms the core of the service offered, and is an important characteristic when profiling the service offering.

A. Project Clearwater Cloud IMS

The IP Multimedia Subsystem (IMS) is a reference architecture first defined by the 3GPP for delivering fixed-line and mobile communications applications built on the Internet Protocol (IP) [45]. Project Clearwater[1] is an open-source implementation of IMS in the Cloud, following IMS architectural principles and supporting all of the key standardized interfaces expected of an IMS core network. The web services-oriented design inherent to Clearwater makes it ideal for instantiation within NFV environments as a virtualized VNF. The new Service-Based Architecture adopted by the 5G standards is very closely related to the inherent Clearwater model, and it has been widely used in research as a standard testbed setup for NFV related work [6], [7], [16], [13], [33].

In our work, we use Clearwater as the use-case for a Cloud based virtualised NFV application. It consists of 6 main components, namely Bono, Ellis, Homer, Homestead, Ralf, and Sprout. A high level view of these VNFCs and their functionalities replicating a standard IMS architecture is as shown in Figure 2.

B. Defining SLOs

We use raw network telemetry data and system metrics obtained via a standard realization of the Clearwater testbed

[1] https://www.projectclearwater.org

TABLE I: Raw metrics collected through Monasca during Clearwater vIMS application monitoring. This data is available for each of the VNFCs, and is sampled every 30 seconds.

		Metric Name	Semantics
CPU		cpu.idle_perc	Percentage of time the CPU is idle when no IO requests are in progress
		cpu.system_perc	Percentage of time the CPU is used at the system level
		cpu.wait_perc	Percentage of time the CPU is idle AND there is at least one IO request in progress
Disk	**Disk**	disk.inode_used_perc	The percentage of inodes that are used on a device
		disk.space_used_perc	The percentage of disk space that is being used on a device
	IO Read	io.read_kbytes_sec	Kbytes/sec read by an IO device
		io.read_req_sec	Number of read requests/sec to an IO device
		io.read_time_sec	Amount of read time in seconds to an IO device
	IO Write	io.write_kbytes_sec	Kbytes/sec written by an IO device
		io.write_req_sec	Number of write requests/sec to an IO device
		io.write_time_sec	Amount of write time in seconds to an IO device
Load		load.avg_1_min	The normalized (by number of logical cores) average system load over a 1 minute period
		load.avg_15_min	The normalized (by number of logical cores) average system load over a 15 minute period
		load.avg_5_min	The normalized (by number of logical cores) average system load over a 5 minute period
Memory		mem.free_mb	Mbytes of free memory
		mem.usable_mb	Total Mbytes of usable memory
		mem.usable_perc	Percentage of total memory that is usable
Network	**In**	net.in_bytes_sec	Number of network bytes received per second
		net.in_packets_sec	Number of network packets received per second
	Out	net.out_bytes_sec	Number of network bytes sent per second
		net.out_packets_sec	Number of network packets sent per second

setup to define the SLIs and SLOs governing an informal SLA. These metrics were collected on a 30 second sampling frequency through Monasca[2], an open-source Python based monitoring service running on each of the Clearwater VNFCs. The list of these collected metrics is presented in Table I, while further details regarding the data is elaborated upon in Section VI.

These collected metrics are utilised as the foundations for the SLIs, which when matched with a target threshold or range form SLOs. While the SLOs are largely dependent on the kind of application use-case, and the underlying SLA, we recognize them on the basis of the four key areas that are critical towards the deliverance of required performance. Authors in [33] recognize the lack of realistic SLOs in consideration in research, and recommend that an SLO be composed of a combination of atleast two metrics.

To set a fair ground for our analysis, we define the SLOs with this definition in mind, and form these rules for the four key areas that impact the performance of an underlying system. This is to highlight the varying reason behind the loss of QoS at any time withing the use-case application, so that the scaling policies can be customised at a more granular level towards better efficiency.

Formally, the SLOs are defined in terms of SLIs as a target value:

$$SLI \leq target\ threshold \tag{1}$$

or as a range of values for service level:

$$lowerbound \leq SLI \leq upperbound \tag{2}$$

At any time, the state of an SLO can be represented as either violated or compliant. We define four SLOs for the Clearwater VNF, targeting the load, computation, disk, and input/output (IO) characteristics respectively. Let $\mathcal{L}$ denote the set of SLOs thus defined:

$$\mathcal{L} = [SLO_1, SLO_2, SLO_3, SLO_4] \tag{3}$$

This equivalently denotes:

$$\mathcal{L} = [SLO_{load}, SLO_{computation}, SLO_{disk}, SLO_{io}] \tag{4}$$

The metrics as defined in Table I are captured at the granularity of the individual VNFCs as shown in Figure 2, and an SLO violation at any of the individual VNFCs triggers an SLO violation state for the Clearwater application service. Therefore, we ultimately define the SLOs at the application level, i.e. for the entire VNF as an application service. Thus, each data instance is associated with 4 SLOs as defined by $\mathcal{L}$ above, where $SLO_j, j \in [1, 2, 3, 4]$ assumes one of two states:

$$SLO_j = \begin{cases} 1, & \text{if } Violated \text{ (at any VNFC)} \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

The SLOs are formulated after studying the scaling policies, dynamic monitoring offerings, and alarm definitions adopted and used in practice by major cloud service providers such as Amazon Web Services [46], [47], Microsoft Azure [48], Google Cloud [49], and Huawei Cloud [50]. We categorise the SLOs into four broad characteristics, and enhance these for a fine-grained monitoring of a latency-sensitive application that needs high availability and reliability. The formal definitions of the SLOs are described below, with the thresholds largely defined based on the application's usage characteristics, reaction to stress tests, and use-case requirements. The metrics referenced in the rule definitions are as captured and described in Table I.

1) SLO_1: Load

Load is a measure of the computational work ongoing, and captures the running processes— either using the CPU, or in a wait state. The values are normalized by number of CPU

[2]www.monasca.io

cores. This SLO captures the application state based on the average load on an instance over a period of the last 1 minute, 5 minutes, and 15 minutes. A short term surge in load may be due to regular operational usage and thus may not be a direct cause of concern, but higher load averages over longer intervals is a direct sign of overload. The data instances that meet the following criterion are assigned a violation state for SLO_1.

$$\begin{gathered}(load.avg_{1min} \geq \gamma_1 \quad and \quad load.avg_{5min} \geq \gamma_2) \\ \text{or} \\ (load.avg_{15min} \geq \gamma_3)\end{gathered} \tag{6}$$

γ_1, γ_2, and γ_3 are defined as tunable threshold parameters, and were given the respective value of 0.7, 0.5, and 0.8 in the experimentation.

2) SLO_2: Computation

This SLO is defined as a combination of certain CPU and RAM characteristics. While short bursts of IO can spike system kernel usage and is regular, this combined with the lack of adequate idle time for the CPU over time when no IO is in progress is a sign of overload or malfunction. The SLO is also considered violated if the amount of available RAM falls below a threshold, which is a warning sign of inadequate system resource allocation.

$$\begin{gathered}(mem.usable_{perc} \leq \gamma_4) \\ \text{or} \\ (cpu.system_{perc} \geq \gamma_5 \quad and \quad cpu.idle_{perc} \leq \gamma_6)\end{gathered} \tag{7}$$

γ_4, γ_5, and γ_6 are defined as tunable threshold parameters, and were given the respective value of 40, 10, and 60 in the experimentation.

3) SLO_3: Disk

This SLO captures prolonged periods of inefficient IO wait times when the CPU is otherwise idle, which indicates potential bottlenecks in the read/write operations accrued by the hard disk.

$$\begin{gathered}(cpu.wait_{perc} \geq \gamma_7) \\ \text{or} \\ \left(\frac{cpu.wait_{perc}}{cpu.system_{perc}} \geq \gamma_8\right)\end{gathered} \tag{8}$$

γ_7, and γ_8 are defined as tunable threshold parameters, and were given the respective value of 50, and 2 in the experimentation.

4) SLO_4: IO

This SLO captures the latency when interacting with IO devices, when there is a sudden and prolonged surge in incoming network traffic as compared to the moving average. A moving average (or rolling mean) is defined as the unweighted mean of the previous M data instances sampled, where the selection of M (sliding window) depends on the degree of smoothing desired since increasing the value of M improves the smoothing at the expense of accuracy. Mathematically, rolling mean with a window of size M at a time period t is denoted as follows, where $a_t, a_{t-1}, \cdots$ represent the value at instance t, $t-1, \cdots$ respectively, and so on.

$$\overline{\text{Rolling Mean}}_t^M = \frac{a_t + a_{t-1} + \cdots + a_{M-(t-1)}}{M} \tag{9}$$

We choose M to be 2880 (γ_9) for the network traffic characteristics, which, considering that the sampling happens every 30 seconds, corresponds to 24 hours. For the IO read/write characteristics, we use a moving average over the last 3 sampling instances, so $M = 3$ (γ_{10}), which corresponds to the last 1.5 minutes. Thus, this SLO considers both read/write requests per second as compared to the last 90 seconds, as well as the amount of time spent reading/writing with an IO device as compared to the last 90 seconds.

$$\begin{gathered}\Big(net.in_{bytes\_sec} > \gamma_{11}\ \overline{net.in}^{\gamma_9}_{bytes\_sec} \\ and \\ net.out_{bytes\_sec} > \overline{net.out}^{\gamma_9}_{bytes\_sec}\Big) \\ \text{and} \\ \Big(io.read_{req\_sec} > \gamma_{11}\ \overline{io.read}^{\gamma_{10}}_{req\_sec} \\ or \\ io.write_{req\_sec} > \gamma_{11}\ \overline{io.write}^{\gamma_{10}}_{req\_sec}\Big) \\ \text{and} \\ \Big(io.read_{time\_sec} > \overline{io.read}^{\gamma_{10}}_{time\_sec} \\ or \\ io.write_{time\_sec} > \overline{io.write}^{\gamma_{10}}_{time\_sec}\Big)\end{gathered} \tag{10}$$

γ_{11} is defined as a tunable parameter, and was given the value of 1.5 in the experimentation.

IV. Multi-Label Classification

Multi-label classification is defined as a classification task where each data sample instance can be assigned n labels from a set of $|\mathcal{L}|$ possible label classes as defined in 3 and 4, where $n \in [0, \mathcal{L}]$, and $|\mathcal{L}| > 1$. Each of the class labels in $\mathcal{L}$ is binary, i.e. either 0 or 1, where 0 denotes a negative occurrence and 1 denotes the positive occurrence.

This implies that $\mathcal{L}$ is a set of binary classes that are not mutually exclusive, and each sample of input data can be assigned multiple such binary classes as applicable.

In our problem definition, $\mathcal{L}$ is the set of all SLO violation classes, where each class can take a value of 0 or 1, signifying compliance and violation states respectively.

Semantically, a multi-label target can be thought of as a set of labels for each sample. Multi-label classification differs from multi-class classification in that the latter applies mutually exclusive labels to a data sample, which is not the case for multi-label problems. The challenge with multi-label classification is the requirement for such classifiers to treat the multiple classes simultaneously, accounting for the correlated behaviour among them.

A. Mathematical Formulation

Formally, let $\mathcal{D}$ be a multi-label dataset where $\mathcal{X} = \mathbb{R}^d$ is a d-dimensional input instance space of numerical features, and $\mathcal{L} = \{\lambda_1, \lambda_2, \cdots, \lambda_q\}$ a finite output label space of $|\mathcal{L}| = q$ discrete class labels (with values 0 or 1), and $q > 1$.

The task of multi-label learning is to learn a function $f : \mathcal{X} \longrightarrow 2^{\mathcal{L}}$ from the multi-label training set $\mathcal{S}$ with u examples, $\mathcal{S} = \{(\mathbf{x}_i, Y_i) \mid 1 \leq i \leq u\}$. To compare, multi-class classification can be seen as a special case of multi-label classification where $f : \mathcal{X} \longrightarrow \mathcal{L}$, while in binary classification $f : \mathcal{X} \longrightarrow \{0, 1\}$.

For each multi-label example $(\mathbf{x}_i, Y_i)$, $\mathbf{x}_i \in \mathcal{X}$ is a d-dimensional feature vector $(x_{i1}, x_{i2}, \cdots, x_{id})^\top$, and $Y_i \subseteq \mathcal{L}$ is the set of labels associated with $\mathbf{x}_i$. Label associations can also be represented as a q dimensional binary vector $\mathbf{y}_i = (y_{i1}, y_{i2}, \cdots, y_{iq})^\top = \{0, 1\}^q$, where each element is 1 if the label is relevant, and 0 otherwise. By contrast, in single-label (binary or multi-class) learning, $|Y| = 1$.

B. Multi-Label Learning Methods

Approaches to solve a multi-label classification problem typically belong to two main categories— (a) problem transformation, and (b) algorithm adaptation [14].

1) Problem Transformation Methods

Problem transformation methods aim to transform and decompose the multi-label learning problem into one or many single-label classification tasks, followed by a re-transformation of the outputs into a multi-label representation. The key idea of problem transformation methods is to fit the data to the well-represented set of existing algorithms.

This methodology can be further grouped into three use-case specific categories based on the kind of transformation required— binary relevance, label ranking, and multi-class classification.

2) Algorithm Adaptation Methods

Algorithm adaptation methods, on the other hand, aim to directly tackle the multi-label learning task by adapting or extending the existing classification algorithms to work with multi-label data directly. Unlike problem transformation methods, the key idea of algorithm adaptation is thus to fit or extend an algorithm to work with a multi-label data representation.

C. Machine Learning Methodologies

Since multi-label classification can be transformed to a binary classification task with the label transformation method as described above, we use two supported methods to transform the multi-label problem to a single-label problem— Binary Relevance (BR), and Classifier Chains (CC). These enable us to compare the performance of some compatible single-label binary classification algorithms when adapted to our multi-label use-case of predicting SLO violation categories.

1) Binary Relevance (BR)

Given $\mathcal{L} = \{\lambda_j \mid j \in [1, q]\}$ as a finite output label space of q discrete class labels as described above, the Binary Relevance method involves treating the j^{th} class independently, i.e. fitting one binary single-label classifier $\mathfrak{B}$ for each class label λ_j. This is akin to a One-vs-Rest strategy with binary classes, where q binary classifiers each treat one of the q label classes independently. In the rest of the paper, Binary Relevance and One-vs-Rest is used interchangeably, and One-vs-Rest implies a binary One-vs-Rest strategy.

For a binary learning algorithm $\mathfrak{B}$ embedded in a problem transformation methodology, the worst-case bound training complexity is $\mathcal{O}(q.\mathcal{F}_{\mathfrak{B}}(u, d))$, and the testing complexity is $\mathcal{O}(q.\mathcal{F}'_{\mathfrak{B}}(d))$, where $\mathcal{F}_{\mathfrak{B}}$ denotes the training complexity of the binary classification algorithm $\mathfrak{B}$ embedded in a problem transformation method, and $\mathcal{F}'_{\mathfrak{B}}$ denotes its corresponding testing complexity.

We use logistic regression as a base classifier within the BR methodology, and evaluate its performance against other methods.

Logistic Regression is a linear classification model that uses the logistic (sigmoid) function to take in the input log-odds and output the probability of outcomes for the binary dependent variable. This is interpreted as a binary classification model by establishing a cutoff threshold on the output probabilities to classify the outcome as belonging to one of the two classes.

2) Classifier Chain (CC)

Given $\mathcal{L} = \{\lambda_j \mid j \in [1, q]\}$ as a finite output label space of q discrete class labels as described above, the Classifier Chain method involves linking q binary classifiers ordered randomly along a chain, where the j^{th} classifier tackles the binary relevance problem of label λ_j. However, the feature-space of the j^{th} classifier in CC is extended with the binary label associations of all the previous classifiers linked before it in the chain, thus also exploiting label correlations to an extent.

Classifier chains have a worst-case bound training complexity of $\mathcal{O}(q.\mathcal{F}_{\mathfrak{B}}(u, d + q))$, and a testing complexity of $\mathcal{O}(q.\mathcal{F}'_{\mathfrak{B}}(d + q))$. To evaluate the CC methodology, we use the the following machine learning algorithms as base binary classifiers— Logistic Regression, Naive Bayes, AdaBoost, and Support Vector Machine (SVM).

Naive Bayes is a supervised learning probabilistic classifier that leverages Bayes' theorem with an assumption of conditional independence between each pair of features. Owing to the binary feature space, we use the Bernoulli variant for Naive Bayes as a base classifier.

AdaBoost, or Adaptive Boosting, is an ensemble learning classification technique that builds multiple weak learners on the data and adjusts their weights to improve upon misclassifications as they occur, overall resulting in a boosted classifier.

Support Vector Machine (SVM) is a supervised learning methodology that supports both linear and non-linear classification through kernel functions. An SVM classifier is traditionally non-probabilistic, and we deploy one with a Radial Basis Function (RBF) kernel for a non-linear decision function.

3) Multi-Label k-Nearest Neighbours

Multi-label k-nearest neighbors (ML-kNN) extends the k-nearest neighbors (kNN) algorithm, which is an instance based lazy learning algorithm [34]. It works by identifying the k-nearest neighbours for an example instance in the training set, and utilizes the maximum a posteriori (MAP) rule to make a prediction leveraging the labeling information gained through the neighbours. While ML-kNN reasons the relevance of each label separately [37], it inherits the merits of both lazy learning and Bayesian reasoning. ML-kNN has a worst-case

bound training complexity of $\mathcal{O}(u^2d + quk))$, and a testing complexity of $\mathcal{O}(ud + qk))$.

4) Decision Trees

Decision Trees are a non-parametric supervised learning methodology that have been adapted for a multi-label setup by adapting the C4.5 algorithm [36]. The algorithm builds a tree-based model with conditional control statements forming decision rules for classification, and assumes label independence in a multi-label setup [37]. Decision tree models belong to the class of white-box family of algorithms, and the depth of the decision tree is analogous to the complexity of the decision rules. Decision tree based multi-label models have a worst-case bound training complexity of $\mathcal{O}(udq))$, and a testing complexity of $\mathcal{O}(uq))$.

5) Random Forest

A random forest is a decision tree based ensemble learning strategy that works as a meta-estimator and fits a number of decision tree classifiers on various sub-samples of the dataset, leverages this information to control over-fitting. Same as with the adapted decision trees above, random forests belong to the algorithm adaptation method family for handling multi-label classification problems. Likewise, multi-label random forest models have a worst-case bound training complexity of $\mathcal{O}(udq))$, and a testing complexity of $\mathcal{O}(uq))$.

D. Deep Neural Network

Deep learning is a powerful subset of the machine learning domain, that uses over three layers of interconnected neurons (nodes) to create an artificial neural network (ANN) model. Each neuron within a layer represents a mathematical function comprising of inputs, weights, bias, and threshold; and uses an activation function to transform the outputs to a non-linear space to learn and perform more complex tasks. Thus, subject to the right choice of architecture and parameters for the task at hand, ANNs can be trained to address a wide variety of complex tasks, including that of directly addressing multi-label classification.

To this end, we build a deep multi-layer perceptron (MLP) model, i.e. a fully connected deep feed-forward neural network to natively address the multi-label classification problem at hand and compare its performance against other machine learning methodologies. Specific to the task at hand, we appropriately design the model such that the output layer consists of q neurons, each representing a label λ_j in $\mathcal{L}$, where $\mathcal{L} = \{\lambda_j \mid j \in [1, q]\}$. We use sigmoid as the activation function in the output layer, so the j^{th} neuron in that layer outputs the probabilities in the range $[0, 1]$ of the data instance belonging to λ_j. Similar as with logistic regression, this is interpretable as a binary classification by setting a cutoff probability threshold value (set to 0.5) for each class label.

$$Sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{11}$$

The computational complexity for neural network models can be written as $\mathcal{O}(ru \cdot \mathcal{O}(n^3)))$, where r is the number of iterations, and $\mathcal{O}(n^3)$ is the complexity of the underlying matrix multiplications. The worst-case bound complexity of neural networks is thus $\mathcal{O}(n^5))$. However, this is a very wide overestimate, considering that in practice, modern day neural networks are trained efficiently using stochastic gradient descent, a variety of optimizations and efficient activation functions, over-specification, and regularization [51]. To this end, determining the actual complexity of modern neural networks is an active research area. The choice of architecture and learning model is further elaborated upon in section VI.

The evaluation results for the performance of each of these models for the multi-label classification task at hand are as presented in Table II.

E. Metrics

Let $\mathcal{T} = \{(\mathbf{x}_i, Y_i) \mid 1 \leq i \leq p\}$ be the test set with p instances, and $f(\cdot)$ be the learned multi-label classifier. For any unseen instance $\mathbf{x} \in X$, the multi-label classifier $f(\cdot)$ predicts $f(\mathbf{x}) \subseteq \mathcal{L}$ as the set of proper labels for $\mathbf{x}$. Correspondingly, let $Y_i \subseteq \mathcal{L}$ and $Z_i \subseteq \mathcal{L}$ denote the sets of ground-truth and predicted labels for an input instance from the p instances in the test set $\mathcal{T}$.

In traditional single-label classification problems such as the ones belonging to binary classification or multi-class classification, accuracy has been the most common evaluation metric, usually complemented by precision, recall, F-measure, and area under the curve for the receiver operating characteristic (AUC-ROC) [35]. However, multi-label classification requires a wider and contrasting range of metrics for an overall comparison of performance, given the added freedom, flexibility and complexity in such a setup [52], [36], [14]. These can be grouped as example-based, label-based, and rankings-based [34], [14]. The former two belong to the bipartitions-based evaluation category, which is based on the idea of comparing the predicted relevant labels with the corresponding ground truth labels. The ranking based metrics, on the other hand, offer another perspective to measure the generalization performance of multi-label problems, wherein the most relevant label for an example instance is assigned a value of 1, and so on, with the least relevant label assigned a rank of q.

1) Example-based Evaluation

Example-based evaluation metrics are calculated based on the average differences of the predicted and actual sets of labels over all examples of the test set $\mathcal{T}$.

Exact Match Ratio, also known as Subset Accuracy, computes the fraction of examples for which the predicted set of labels is an exact match with the ground-truth labels. This is defined to be the multi-label equivalent of the traditional accuracy metric; and given that it does not distinguish between partially correct and completely incorrect, tends to be an overly strict measure, especially for a larger label space q. It is formally defined as under, where $[\![\cdot]\!]$ denotes an indicator function that returns 1 if $[\![true]\!]$, and 0 if $[\![false]\!]$. The best performance of this metric is 1.

$$Exact\ Match\ Ratio = \frac{1}{p}\sum_{i=1}^{p} [\![Y_i = Z_i]\!] \tag{12}$$

TABLE II: Performance of the various Machine Learning Models as compared to the Base Deep Neural Network Model, evaluated on all presented multi-label classification metrics.

		Binary Relevance	Classifier Chain				ML-kNN	Decision Tree	Random Forest	Deep Feed Forward Neural Network
		Logistic Regression	Logistic Regression	AdaBoost	Naive Bayes	SVM				
Training Loss		NA	NA	NA	NA	NA	NA	NA	NA	0.009
AUC-PRC	Micro Average	0.881	0.902	0.978	0.825	0.857	0.854	0.994	0.980	**1.000**
	Macro Average	0.570	0.632	0.749	0.398	0.477	0.523	0.752	0.745	**0.778**
	Weighted Average	0.880	0.904	0.981	0.832	0.841	0.857	0.996	0.981	**0.998**
AUC-ROC	Micro Average	0.957	0.968	0.994	0.927	0.938	0.940	0.997	0.994	**1.000**
	Macro Average	0.701	0.735	0.857	0.651	0.624	0.657	0.885	0.846	**0.969**
	Weighted Average	0.593	0.634	0.902	0.583	0.526	0.567	0.989	0.901	**1.000**
True Positives (TP)		36211	36886	37976	34220	34582	34961	**38062**	38016	38040
False Positives (FP)		3379	2988	656	4216	3063	3539	130	639	**91**
True Negatives (TN)		100034	100425	102757	99197	100350	99874	103283	102774	**103322**
False Negatives (FN)		2008	1333	243	3999	3637	3258	**157**	203	179
Precision	Micro Average	0.915	0.925	0.983	0.890	0.919	0.908	0.997	0.983	**0.998**
	Macro Average	0.643	0.679	**0.790**	0.582	0.451	0.615	0.779	0.745	0.747
	Weighted Average	0.907	0.922	0.982	0.897	0.831	0.882	**0.996**	0.981	**0.996**
Recall	Micro Average	0.947	0.965	0.994	0.895	0.905	0.915	**0.996**	0.995	**0.996**
	Macro Average	0.631	0.678	0.769	0.529	0.500	0.550	**0.775**	0.748	0.747
	Weighted Average	0.947	0.965	0.994	0.895	0.905	0.915	**0.996**	0.995	**0.996**
F-1 Score	Micro Average	0.931	0.945	0.988	0.893	0.912	0.911	0.996	0.989	**0.997**
	Macro Average	0.633	0.678	0.775	0.436	0.474	0.562	**0.777**	0.747	0.747
	Weighted Average	0.925	0.942	0.988	0.856	0.867	0.894	**0.996**	0.988	**0.996**
Jaccard Similarity Coefficient	Micro Average	0.870	0.895	0.977	0.806	0.838	0.837	0.993	0.978	**0.994**
	Macro Average	0.570	0.631	0.759	0.377	0.451	0.520	**0.763**	0.743	0.744
	Weighted Average	0.875	0.899	0.978	0.815	0.831	0.849	**0.994**	0.978	**0.994**
(Subset) Accuracy (or EMR)		0.856	0.882	0.974	0.793	0.825	0.822	0.991	0.976	**0.993**
Hamming Loss		0.038	0.030	0.006	0.058	0.047	0.048	0.002	0.005	**0.001**
Log Loss		1.837	1.206	0.272	3.731	3.630	2.981	**0.266**	0.279	0.289
Subset Zero-One Loss		0.144	0.117	0.025	0.206	0.174	0.178	0.008	0.023	**0.006**
Coverage		1.236	1.194	1.102	1.364	1.305	1.310	1.092	1.100	**1.080**
Average Precision (Label Ranking)		0.963	0.970	0.993	0.940	0.963	0.953	0.996	0.993	**0.999**
Ranking Loss		0.045	0.034	0.007	0.081	0.060	0.065	0.004	0.006	**0.0002**

Accuracy is defined by micro-averaging the Jaccard Similarity Coefficients across all examples, and is defined as:

$$Accuracy = \frac{1}{p}\sum_{i=1}^{p}\frac{|Y_i \cap Z_i|}{|Y_i \cup Z_i|} \tag{13}$$

where $|\cdot|$ denotes the cardinality, and the Jaccard Similarity Coefficient for the i^{th} example instance is defined as:

$$Jaccard\ Score = \frac{|Y_i \cap Z_i|}{|Y_i \cup Z_i|} \tag{14}$$

Precision is defined as the proportion of correctly predicted labels to the total number of predicted labels, averaged over all examples.

$$Precision = \frac{1}{p}\sum_{i=1}^{p}\frac{|Y_i \cap Z_i|}{|Z_i|} \tag{15}$$

Recall is defined as the proportion of correctly predicted labels to the total number of actual labels, averaged over all examples.

$$Recall = \frac{1}{p}\sum_{i=1}^{p}\frac{|Y_i \cap Z_i|}{|Y_i|} \tag{16}$$

$\mathbf{F}_\beta$ Score is defined as a weighted harmonic mean of precision and recall, whereby β controls the weight of recall in the combined scoring. The case when $\beta = 1$ is referred to as the F_1 score (or balanced F_1 score), which implies that precision and recall are weighted equally in the calculation.

$$F_\beta\ Score = (1+\beta^2)\ \frac{Precision \times Recall}{(\beta^2.\ Precision) + Recall} \tag{17}$$

$$F_1\ Score = 2\ \frac{|Y_i \cap Z_i|}{|Y_i| + |Z_i|} \tag{18}$$

Subset Zero-One Loss is defined as the fraction of imperfectly classified examples, with the best performance at 0.

$$SubsetZeroOneLoss = 1 - ExactMatchRatio \tag{19}$$

Hamming Loss is defined as the fraction of labels predicted incorrectly, and accounts for all misclassifications (prediction errors and omission errors) over total number of label classes over all examples. It is more forgiving in that it penalizes only the individual labels. It is formally defined as under, where Δ stands for the symmetric difference between the two sets, the equivalent of XOR in Boolean logic. The best performance of this metric is 0.

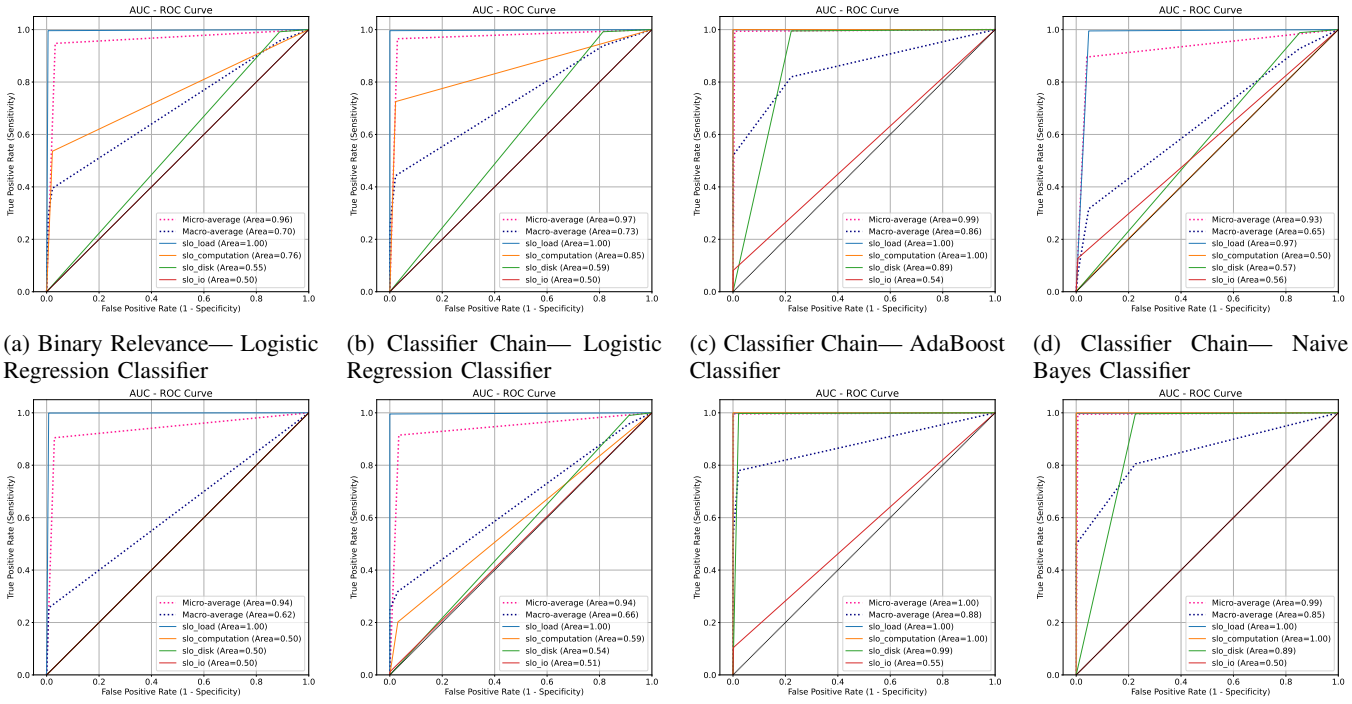
$$Hamming\ Loss = \frac{1}{p}\sum_{i=1}^{p}\ \frac{1}{q}\ |Y_i \Delta Z_i| \tag{20}$$

Log Loss, also called the cross-entropy loss, is used to evaluate the probability outputs of a classifier instead of its discrete predictions. As applicable in the multi-label context, the binary variant is defined as under, where $P(\cdot)$ is defined as the corresponding probability estimate, a threshold value of which leads to Z_i.

$$\begin{aligned} Log\ Loss = -\frac{1}{p}\sum_{i=1}^{p}\ \Big[Y_i \cdot log(P(Y_i)) + \\ (1-Y_i) \cdot log(1-P(Y_i))\Big] \end{aligned} \tag{21}$$

2) Label-based Evaluation

Label-based evaluation metrics are calculated by evaluating the classifier's performance on each class label separately, and then returning the micro or macro averaged value across all class labels.



(a) Binary Relevance— Logistic Regression Classifier

(b) Classifier Chain— Logistic Regression Classifier

(c) Classifier Chain— AdaBoost Classifier

(d) Classifier Chain— Naive Bayes Classifier

(e) Classifier Chain— Support Vector Machine Classifier

(f) Multi-Label k-Nearest Neighbours Classifier

(g) Decision Tree Classifier

(h) Random Forest Classifier

Fig. 3: AUC-ROC plots depicting the learning of the various machine learning classifiers on individual class labels, as well as the micro and macro averaged performance on the entire set.

For the j^{th} class label λ_j, TP_j, FP_j, TN_j, FN_j denote the number of True Positive, False Positive, True Negative, and False Negative test samples from $\mathcal{T}$ with respect to λ_j, where $TP_j + FP_j + TN_j + FN_j = p$.

$$\begin{aligned} TP_j &= \big|\{\mathbf{x}_i \mid \lambda_j \in Y_i \wedge \lambda_j \in Z_i\}\big| \\ FP_j &= \big|\{\mathbf{x}_i \mid \lambda_j \notin Y_i \wedge \lambda_j \in Z_i\}\big| \\ TN_j &= \big|\{\mathbf{x}_i \mid \lambda_j \notin Y_i \wedge \lambda_j \notin Z_i\}\big| \\ FN_j &= \big|\{\mathbf{x}_i \mid \lambda_j \in Y_i \wedge \lambda_j \notin Z_i\}\big| \end{aligned} \tag{22}$$

Any known evaluation measure applicable to a binary classifier can be adapted to a label-based setup. For any binary evaluation metric $\mathcal{B} \in \{Accuracy, Precision, Recall, F_\beta, \cdots\}$ calculated on the basis of $\mathcal{B}(TP_j, FP_j, TN_j, FN_j)$ for a particular label, the overall label based classification metrics can be obtained by one of the following two averaging methodologies:

Macroaveraging, which implies calculating a metric $\mathcal{B}$ for each class in $\mathcal{L}$, and then averaging over all classes. This can be seen as per-class averaging, and since it gives equal weights to all classes, it is a good methodology to highlight the performance of infrequent classes that are nonetheless important.

$$\mathcal{B}_{macro} = \frac{1}{q}\sum_{j=1}^{q} \mathcal{B}\Big(TP_j, FP_j, TN_j, FN_j\Big) \tag{23}$$

Microaveraging, which implies calculating a metric $\mathcal{B}$ globally over all the examples in $\mathcal{T}$ together, aggregating the measure over all classes as a whole. This can be seen as per-example averaging, and tends to be dominated by the performance of the most frequently occurring classes within the example space.

$$\mathcal{B}_{micro} = \mathcal{B}\left(\sum_{j=1}^{q} TP_j, \sum_{j=1}^{q} FP_j, \sum_{j=1}^{q} TN_j, \sum_{j=1}^{q} FN_j\right) \tag{24}$$

An example of the binary evaluation metrics $\mathcal{B}$ that the above averaging methodologies can be applied on include:

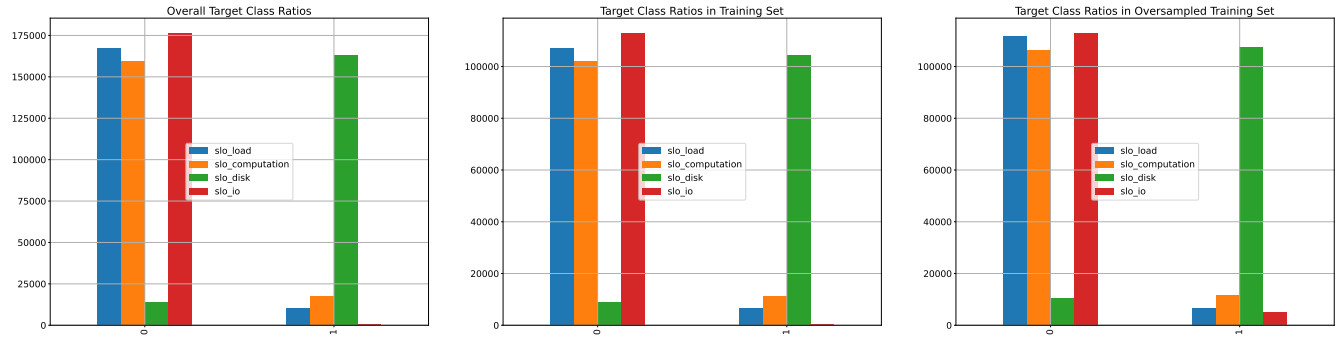
$$\begin{aligned} Accuracy_{(TP_j, FP_j, TN_j, FN_j)} &= \frac{TP_j + TN_j}{TP_j + FP_j + TN_j + FN_j} \\ Precision_{(TP_j, FP_j, TN_j, FN_j)} &= \frac{TP_j}{TP_j + FP_j} \\ Recall_{(TP_j, FP_j, TN_j, FN_j)} &= \frac{TP_j}{TP_j + FN_j} \end{aligned} \tag{25}$$

and so on.

3) Ranking-based Evaluation

Ranking based evaluation metrics compare the predicted ranking of the labels with the ground-truth ranking. The rank predicted by a label ranking method for a label λ is denoted as $\mathcal{R}_i(\lambda)$.

Coverage is defined as an evaluation that calculates an average for how far down the list of ranked labels does the classifier need to go in order to cover all the true labels of an example instance. It is useful in use-cases where it is utmost important to get all true labels predicted, even if that means a few extra false positives [35]. Coverage can be also considered an example-based metric as it is firstly computed



(a) Multi-Label Dataset (b) Training Set (c) Random Oversampling

Fig. 4: Overall distribution of the 4 SLO class labels. The training set contains a positive distribution of 5.64%, 10.05%, 92.16%, 0.23% respectively on the four SLO classes.

for each example, and then averaged across the test set $\mathcal{T}$. The smaller the value of coverage, the better the performance. It is common in implementations to remove the subtraction by 1 in the following equation, so as to be able to extend the metric to handle the degenerate case in which an example instance has no true labels associated with it [53].

$$Coverage = \frac{1}{p}\sum_{i=1}^{p} \max_{\lambda \in Y_i} \mathcal{R}_i(\lambda) - 1 \tag{26}$$

Average Precision is defined as the average fraction of labels ranked higher than a particular label $\lambda \in Y_i$, which actually are in Y_i. This can be also considered an example-based metric as it is firstly computed for each example, and then averaged across the test set $\mathcal{T}$. The best value for this evaluation metric is 1, with larger values indicating better performance.

$$Average\ Precision = \frac{1}{p}\sum_{i=1}^{p} \frac{1}{Y_i} \sum_{\lambda \in Y_i} \frac{\left|\{\lambda' \in Y_i \mid \mathcal{R}_i(\lambda') \leq \mathcal{R}_i(\lambda)\}\right|}{\mathcal{R}_i(\lambda)} \tag{27}$$

Ranking Loss is defined as an evaluation of the average proportion of label pairs that are incorrectly ordered for the example instance, i.e. true labels have a lower score than false labels. $\overline{Y}$ denotes the complementary set of Y in $\mathcal{L}$, where the goal is that the labels in Y be ranked higher than the labels in $\overline{Y}$. This can be also considered an example-based metric as it is firstly computed for each example, and then averaged across the test set $\mathcal{T}$. The best value of this metric is 0.

$$\begin{aligned} Ranking\ Loss &= \frac{1}{p}\sum_{i=1}^{p} \frac{1}{|Y_i||\overline{Y_i}|}\ |E| \\ |E| &= \left\{(\lambda, \lambda') \mid \mathcal{R}_i(\lambda) > \mathcal{R}_i(\lambda'),\ (\lambda, \lambda') \in Y_i \times \overline{Y_i}\right\} \end{aligned} \tag{28}$$

Area Under the Curve (AUC) is defined as either the area under the receiver operating characteristic (AUC-ROC) as illustrated in Figure 3, or the area under the precision-recall curve (AUC-PRC). AUC is an intuitive representation of the probability of a randomly selected positive example getting a higher ranking than a randomly selected negative example. The instance-based definition of AUC as described below follows closely from the Wilcoxon-Man-Whitney Statistic [54].

$$AUC = \frac{1}{p}\sum_{i=1}^{p} \frac{\left|\{(\lambda, \lambda') \in Y_i \times \overline{Y_i} \mid \mathcal{R}_i(\lambda') \geq \mathcal{R}_i(\lambda)\}\right|}{|Y_i||\overline{Y_i}|} \tag{29}$$

This is a label-based ranking metric, and can be calculated as both a macro and micro averaged value based on 23 and 24. Its value ranges from 0 to 1, the higher the better.

$$AUC\text{-}ROC_j = \int_0^1 TPR_j\ d(FPR_j) \tag{30}$$

$$AUC\text{-}PRC_j = \int_0^1 Precision_j\ d(Recall_j) \tag{31}$$

$$\begin{aligned} Sensitivity,\ or\ Recall,\ or\ TPR_j &= \frac{TP_j}{TP_j + FN_j} \\ Specificity,\ or\ TNR_j &= \frac{TN_j}{TN_j + FP_j} \\ FPR_j = 1 - TNR_j &= \frac{FP_j}{TN_j + FP_j} \end{aligned} \tag{32}$$

V. Addressing Class Imbalance in the Multi-Label Context

Typical classification algorithms perform best when the distribution of data in each of the binary classes is equally distributed. However, when dealing with a high volume of data, especially in the multi-label context, class imbalance is a typical side effect, since not all the labels may be evenly distributed across data instances [15].

Figures 4a and 4b show a graphical representation of the distribution of the data in each of the SLO classes in our use-case. Owing to the special properties of a multi-label setup, imbalance in a multi-label dataset is measured differently from regular binary or multi-class classification. For the multi-label dataset $\mathcal{D}$ as described earlier, where $\mathcal{D} = \{(\mathbf{x}_i, Y_i) \mid 1 \leq i \leq |\mathcal{D}|\}$, measures such as cardinality and label density are often used in literature to characterize the distribution of labels [14], [37].

Label Cardinality is a measure that defines the average number of class labels associated with each data instance in a

TABLE III: Metrics capturing the degree of imbalance in the multi-label dataset, and the SLO class labels.

Label Cardinality	Label Density	IR				Mean IR	CVIR
		SLO1	SLO2	SLO3	SLO4		
1.081	0.270	16.259	9.198	1	395.181	105.409	1.833

multi-label dataset $\mathcal{D}$. It is independent of the number of label classes $|\mathcal{L}| = q$ that exist in $\mathcal{D}$.

$$Cardinality(\mathcal{D}) = \sum_{i=1}^{|\mathcal{D}|} \frac{|Y_i|}{|\mathcal{D}|} \tag{33}$$

Label Density is a measure that obtains the ratio of cardinality with the number of label classes that exist in $\mathcal{D}$.

$$Label\ Density(\mathcal{D}) = \frac{Cardinality(\mathcal{D})}{|\mathcal{L}|} \tag{34}$$

However, label cardinality and label density do not accurately convey the notion of imbalance [55]. A more concrete measure of the level of imbalance in a multi-label dataset is through the combined use of three specialised metrics— Imbalance Ratio per Label, Mean Imbalance Ratio, and Coefficient of Variation of the Imbalance Ratio per Label [56]. These are defined as follows:

Imbalance Ratio per Label (IR) is a measure that is defined as the ratio between the majority class label λ and each class label $\lambda_j \in \mathcal{L}$. It therefore takes on the value of 1 for the most frequently occurring class label, and a higher value proportionate to the relative degree of imbalance for the other class labels.

$$IR = \frac{argmax_{\lambda \in \mathcal{L}} \Big(\sum_{i=1}^{|\mathcal{D}|} [\![\lambda \in Y_i]\!] \Big)}{\sum_{i=1,\ j=1}^{|\mathcal{D}|,\ |\mathcal{L}|} [\![\lambda_j \in Y_i]\!]} \tag{35}$$

Mean Imbalance Ratio (Mean IR) is a ratio that presents the mean level of imbalance in $\mathcal{D}$. For example, a Mean IR value of 1.5 represents that there exist, on average, 50% more samples in the majority class label than the minority class label.

$$Mean\ IR = \frac{1}{|\mathcal{L}|} \sum_{j=1}^{|\mathcal{L}|} IR(\lambda_j) \tag{36}$$

Coefficient of Variation of the Imbalance Ratio per Label (CVIR) is an indicator of whether all class labels suffer from the same level of imbalance, or if the degree of imbalance differs between them. For example, a CVIR value of 0.2 represents that there exists 20% variance in the IR values among individual class labels. A higher value of CVIR implies a higher degree of variation of imbalance between individual class labels.

$$CVIR = \frac{1}{Mean\ IR} \sqrt{\sum_{j=1}^{|\mathcal{L}|} \frac{(IR(\lambda_j) - Mean\ IR)^2}{|\mathcal{L}| - 1}} \tag{37}$$

Table III presents the measure for the degree of imbalance in our use-case data, using the above measures. To compare, a multi-label dataset is considered imbalanced if the Mean IR is higher than 1.5, and CVIR is over 0.2 [56].

There exists class imbalance in all the class labels, but the distribution is quite extremely skewed in the last class label, i.e. SLO_{io}. Such a distribution skews the performance as portrayed by the evaluation metrics, as the classifier may not learn the representations of the minority classes in the training set $\mathcal{S}$. For example, in the cases such as our use-case where the binary distribution of classes between the class labels is skewed to such varying range of extremes, a classifier can achieve 94.12% accuracy by just predicting the majority classes for all the class labels in $\mathcal{L}$ in the test set at all times. This is also the reason why training based on maximising accuracy and other conventional metrics do not perform well in an imbalanced multi-label context.

Moreover, since the prediction value for each neuron in the output layer is a continuous value in $[0, 1]$ which is translated to a binary classification by setting a threshold, this creates a trade-off between precision and recall. The AUC-ROC is a great way to ascertain the quality of a predictor without the threshold, and is a very useful metric to evaluate a classifier, especially in the context of an imbalanced class distribution [54]. However, it is much more appropriate to train by aiming to maximize the area under the precision recall curve (AUC-PRC) during training [57], [58].

Post training, while it is helpful to consider the macroaveraged metrics to understand the performance in a multi-label context, they may not independently convey the full picture on the model and its learning capabilities for the individual classes. It may so occur that a class is entirely ignored by the classifier, and its interactions never picked up and modeled in the learning phase, while the performance continues to improve on the macro and microaveraged metrics due to improvements in learning the other classes. As such, for such extreme distributions, it is acutely important to also track each of the class labels individually to understand if the classifier has been modeling them in the learning phase, and to ascertain if it is performing better than a random classifier (i.e. a classifier that assigns classes randomly). We present a visual representation for the performance of each of the learning classifiers on the individual classes by way of confusion matrices and AUC (both AUC-ROC and AUC-PRC). This aspect adds on to the requirement of tracking the classifier's performance on a wider set of metrics as mentioned earlier, as it helps to better understand the shortfalls of a learning strategy, and reveals a bigger picture behind an overly optimistic prediction performance.

As can be understood from Table II, neural networks have a direct advantage in multi-label classification algorithms by concurrently understanding the correlations and working on learning all label classes simultaneously. However, neural network architectures do not implicitly support imbalanced classification [59]. There are two ways to directly address class imbalance in classification problems— either to take steps to make models resilient to class imbalance, or to apply rebalancing and resampling techniques to reduce the imbalance in data [60]. Improving multi-label classification for real world data is currently an active research area [61], [62], and so is addressing imbalance in a multi-label setup [63], [64], [65]. We address imbalance in our use-case by adapting

two conventional methodologies to a multi-label context, and use them to improve the performance of the deep feed-forward neural network model for the minority class labels.

A. Adding Class Weights

The most common strategy to address class imbalance is to introduce appropriate weights for the minority samples, so as to have a weighted learning cost-sensitive strategy[59]. In single output classification models, this is usually done by weighing each class inversely to the ratio of minority to majority labels within it [66].

1) Deep FFNN with Balanced Class Weights

Since multi-label classification consists of multiple class-labels associated with a data instance, and each of the labels is binary, we adapt the above strategy by weighing each class label in inverse proportion to its frequency of positive to negative occurrence. Thus, for each class label λ_j in $\mathcal{L}$, where $\mathcal{L} = \{\lambda_j \mid j \in [1, q]\}$, we weigh λ_j as per the ratio of its $0:1$ label distribution in the training set $\mathcal{S}$:

$$\frac{\mid Total_j \mid}{\mid Positive_j \mid} - 1 \tag{38}$$

which is equivalent to

$$\frac{\mid Negative_j \mid}{\mid Positive_j \mid} \tag{39}$$

This ensures that the class labels that have a lower frequency of positive occurrence in the training set $\mathcal{S}$ are weighted relatively higher than the other class labels, and thus a misclassification for these infrequent classes is compensated by penalizing it higher in that proportion. Thus, the classifier is forced to adequately tune the learning model by also considering the behaviour of the minority classes.

So, for a training set $\mathcal{S}$ with 113,303 data instances with a positive distribution of 5.64%, 10.05%, 92.16%, 0.23% respectively on the four SLO classes, we correspondingly apply the relative positive class label weights of 16.74, 8.94, 0.08, 443.32, with this methodology.

2) Deep FFNN with Clipped Class Weights

Neural networks do not train well with large weights [67]. While imbalance in classes is measured by orders of magnitude, the extreme weight value of 443.32 for the last class with the above method, although formally correct, is anticipated to be detrimental for performance. Furthermore, SLO_3, the most positively distributed class label, assumed a weight of 0.08 by the above methodology. This is lower than the relative weight of 1.0 that is assigned to the distribution of all negative occurrences, which should deteriorate the learning for that class label with a drastic addition in the false predictions for that class label.

Thus, in order to test the above hypothesis, we add a model with clipped class weights. Here, we clip the class weights derived from the above methodology such that the maximum value that any of the class labels assume is set to a tunable upper limit (we set it to 100.0), and the minimum value for any weight is 1.0, i.e. equal to that of the negative distributions. Thus, with this methodology on the four SLO classes, we correspondingly apply the class label weights of 16.74, 8.94, 1.0, 100.0.

Algorithm 1 Decision-making strategies towards addressing extreme class label imbalance for SLA violation prediction in a latency sensitive NFV application

Input: Training Data $\mathcal{S} \in \mathbb{R}^d$

Compute IR for each SLO class label $SLO_j \mid j \in [1, |\mathcal{L}|]$ using Eq. 35, the Mean IR using Eq. 36, and the CVIR using Eq. 37

• **CLASS WEIGHTED STRATEGY**

function CLASS WEIGHTED STRATEGY($\mathcal{S}$)
  if (Mean IR > 0.5 *and* CVIR > 0.2) **then**
    Compute class weights using Eq. 38
    if (Class weight of any $SLO_j > 100$ *or* < 1) **then**
      Compute class weights using Eq. 40
    end if
  end if
end function

• **RESAMPLING STRATEGY**

function RESAMPLING STRATEGY($\mathcal{S}$)
  if (IR of $SLO_j \ggg$ Mean IR) **then**
    Randomly oversample $\mathcal{S}$ for data instances belonging to SLO_j
  else
    Randomly oversample $\mathcal{S}$ until IR for SLO_j gains better proximity to the Mean IR
  end if
end function

3) Deep FFNN with Log Smoothed Class Weights

While the above methodology is expected to be an improvement on the first one, setting the upper threshold for class weights is a heuristic nonetheless, and finding its optimal value would require another grid search each time the model is changed. Thus keeping in mind that the neural network weights should be set to low values for optimal training and bias-variance trade-off, we introduce a log smoothed model that combines the benefits of the above two methods.

$$log\left(\alpha \cdot \left[\frac{\mid Total_j \mid}{\mid Positive_j \mid} - 1\right]\right) \tag{40}$$

α here is a tunable hyperparameter that can be varied to change how many class labels should be weighed above 1. This depends on the distribution of the positive and negative occurrences within a class label, and the unweighted model's learning performance on the skewed classes. We set α to 0.16, which is its maximum value until which the first three classes are weighed at the default value of 1.0, and only the most skewed class, i.e. SLO_4 takes on a class weight. This is the methodology that seeks to set the weighting to a bare minimum, both on the number of classes as well as the relative ratios. Thus, with this methodology on the four SLO classes, we correspondingly apply the class label weights of 1.0, 1.0, 1.0, 4.26.

B. Random Oversampling

An alternative to class weighting is the use of oversampling or undersampling techniques to either increase the occurrence

of minority class labels, or decrease the sampling of the majority class labels respectively during the training phase [68]. It is to be noted that in the multi-label context, this would also cause indirect over or under sampling the other class labels that co-occur in Y_i on these training instances. Further, the degree of oversampling has a proportionately high likelihood of disrupting the learning model in a multi-label context, and is expected to cause overfitting nonetheless, since the model would see certain data instances more than once during training [63]. Since oversampling increases the size of the training set $\mathcal{S}$, it also affects the distributions of labels overall in that set. While traditional resampling strategies aim to create a balanced dataset from an imbalanced one [68], doing so is not as straightforward in a multi-label context due to the high level of concurrence between imbalanced labels, its translational impact on a large number of unique label-sets, and the resultant introduction of a rather high level of noise in training [63].

To this end, a simplistic methodology involves oversampling one or more class labels until their IR matches up to the Mean IR, or to oversample one or more imbalanced class labels until the size of the training set $|\mathcal{S}|$ is a certain percentage larger than the original [56]. However, the exact dynamics vary largely by the individual data and distribution characteristics, as well as the complexity and performance of the classification algorithm in use. Since a deep neural network model is extremely prone to overfitting and was seen to already perform well in capturing the imbalance in the first three class labels, we adopted a random oversampling strategy wherein we oversample the training set $\mathcal{S}$ during training for the data instances that feature the class label with the most extreme IR compared to the other class labels, i.e. SLO_4. Corresponding to the IR values in Table III stating the degree of imbalance in each class label, the actual distribution for each class label (positive occurrence) is 5.64%, 10.05%, 92.16%, and 0.23%. We oversample the instances containing SLO_4 by approximately 4% of $|\mathcal{S}|$, such that the incidence of SLO_4 increases from 0.23% to 4.24%, but still remains below the distribution of the next closest imbalanced class. The new distribution on the bigger training set becomes 5.45%, 9.84%, 91.04%, and 4.24%, with a corresponding IR of 16.70, 9.26, 1, and 21.49. The Mean IR for the new oversampled training set is 12.11, with a CVIR value of 0.74.

With this methodology, the model would see an increased incidence of the most infrequent minority class label in the mini-batches during training, and this is aimed towards increasing the odds that the model will at least be able to capture its behaviour as well as interactions with the other class labels. Nevertheless, the validation and test set remain unchanged, and the training can be readjusted by breaking up the epochs, applying appropriate weight regularization, and providing finer control to early stopping callbacks. Figure 4c shows the resulting distribution after we apply the oversampling strategy on our use-case.

Algorithms 1 and 2 present the consolidated proposition towards effectively addressing class imbalance in a multi-label setup in a deep FFNN model, that extracts the learning from the above four methodologies adapted for our use-case scenario. In the following sections, we present and compare these strategies, highlight model performance and evaluation, and present the challenges through such a setup.

Algorithm 2 A deep feed-forward neural network based multi-label classifier for SLA violation prediction in a latency sensitive NFV application

• PRE-PROCESSING

Input: Data: $\mathcal{D} \in \mathbb{R}^d$

Output : Pre-processed training set $\mathcal{S}$, validation set $\mathcal{V}$ and test set $\mathcal{T}$

function PRE-PROCESSING($\mathcal{D}$)
    Split the data in train, test, and validation sets
    Standardize according to the training set $\mathcal{S}$
end function

• DEEP FFNN MODEL

Input: Pre-processed training set $\mathcal{S}$, validation set $\mathcal{V}$ and test set $\mathcal{T}$

Output: Multi-label classification for SLO violations

function DEEP FFNN MODEL($\mathcal{D}$)
    Construct neural network architecture
    if (Addressing label imbalance with class weight based models) **then**
        CLASS WEIGHTED STRATEGY
    else
        if (Addressing label imbalance with random oversampling) **then**
            RESAMPLING STRATEGY
        end if
    end if
    Apply appropriate values for all key structural and non-structural hyperparameters
    Specify the Early Stopping criterion
    Compile the deep FFNN model
    repeat
        Train and fit the model with batches from $\mathcal{S}$, using $\mathcal{V}$ as validation set
        Output multi-label classification prediction values
        Monitor and evaluate training with validation set
    until Convergence
end function
    Use data from the test set $\mathcal{T}$ to evaluate the trained model

VI. EXPERIMENTAL SETUP

The experiments were all set up using Python (version 3.8.5) and its associated data-science libraries. We use the Scikit-Learn [53] library for all machine learning based methodologies used in this paper, and Tensorflow [69] version 2.4.1 with Keras [70] to program all the deep learning based implementations.

A. Dataset

We use a publicly hosted dataset[3] obtained via a standard Clearwater testbed, a visualization for which is presented in

[3]https://bit.ly/3gPY8c5

Figure 1. The dataset comprises of raw telemetric data files that track 26 metrics for each of the 6 monitored VNFs that compose the Clearwater ecosystem, and includes bursts of abnormal behaviour through its integrated stress testing tools to simulate VNF congestions and QoS degradations. The data is sampled every 30 seconds, and spans an overall period of 2 months. This corresponds to 156 features overall, and 177,098 rows of raw temporal data. A brief description of the captured metrics is summarised in Table I.

B. System Configuration

The experiments were performed in a Docker[4] based containerized environment running atop a bare-metal Linux server with 64 GB RAM, Intel® Xeon® CPU E5-2660 v2 @ 2.20GHz (40 physical processors), 2 NVIDIA® Tesla K20m GPUs, and 500 GB local storage. The Docker image runs an Ubuntu 20.04 LTS operating system, and CUDA version 11.3 for the GPUs.

C. Learning and Adaptation

The data input into any model, be it machine learning or deep learning, is first standardized via mean centering and ensuring a standard deviation of 1, i.e. subtracting each data feature value by its corresponding mean in the training set, and dividing by the standard deviation. This pre-processing, as also presented in Algorithm 2, helps in the learning and convergence of any predictive modeling algorithm [67].

We split the available data into training and test sets in the ratio of $80:20$, and the training set is further split into training and validation sets in the ratio of $80:20$. Hence, overall, the data consisting of 177,098 rows is split in the ratio $64:20:16$, corresponding to train, test, and validation splits respectively.

The choice of architecture has an important role to play in the model learning and performance for a neural network methodology. While there are no fixed guidelines on the number of layers and the number of neurons in each of them, the choice is often driven by following a heuristic based on number of inputs and outputs, and using a random search methodology to arrive at an optimal architecture for the use-case and data at hand. We begin with a shallow universal approximation architecture [13], i.e. with one hidden layer, and the number of neurons in it equal to the average of the neurons in the input and output layers. We use its results as a baseline to adjust the number of layers and neurons, and also adjust the Dropout regularization factor between layers to control overfitting [71] as the model gains complexity. This results in a deep learning architecture, and the key results from this random search based architecture optimization are presented in Table V in the Appendix.

For FFNN model training, we use Binary Crossentropy as the loss function to be minimized. It is a probabilistic loss function that computes the cross-entropy loss between true and predicted labels, and is appropriate for use in a binary classification based setup. Further, we use Adam [72] as the optimizer for its computational efficiency and adaptive learning, with its default learning rate of 10^{-3}. ReLu (Rectified Linear Unit) is used as the activation function for each of the hidden layers due to its computational simplicity and high optimization performance in a deep MLP based setup [67]. As mentioned earlier, we use sigmoid as the activation function for the output layer to concurrently output the individual probability of each label's association with the input data instance, thus supporting multi-label classification outputs.

Since the label classes are imbalanced, we initialize the bias in the output layer to reflect this and enable the model to begin with more reasonable initial guesses, thus contributing to faster convergence. The bias initialization is derived through the log of the ratio of $positive : negative$ in each of the class labels, averaged over all class labels in the training set. This also eliminates the erratic initial behaviour in the learning loss curve in the initial epochs of training, where the model is just learning the bias.

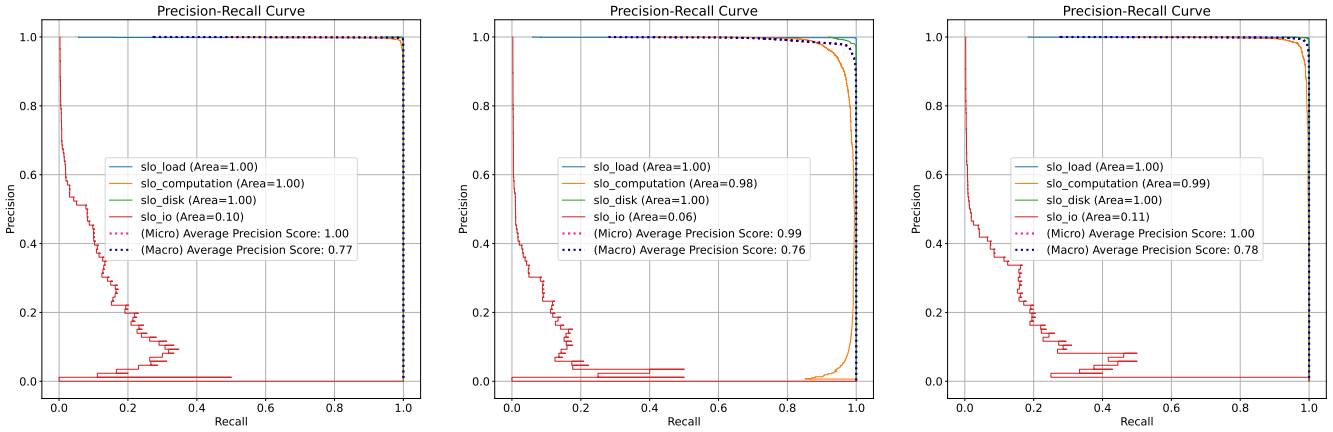
To further control the degree of overfitting during training, we perform a grid search for the optimal choice of weight regularization hyperparameters for the neural network model, and based on the results, apply both L1 and L2 weight regularization on each of the hidden layers in the FFNN, with a regularization factor of 10^{-7} for the unweighted and class weighted models, and 10^{-4} for the model with random oversampling. A visual representation of the outcome of grid search to this end is presented in Figures 12 and 13 in the Appendix.

We use a large batch size of 2048 to increase the probability of class representation from the minority class labels during the training phase. While we set the maximum training epochs to 500, we also deploy an early stopping criteria that tracks the macroaveraged AUC-PRC with a maximization objective, and a patience of 50 epochs to ensure that the training is not stopped at a local optimization minima. At the end of training, model weights are restored from the best epoch, which is considered as the best performance achieved during training, before the model began to overfit on the training set. Finally, the prediction threshold for probability output is set to 0.5, i.e., probability outputs below 0.5 are assigned the class 0, and above 0.5 are assigned the class 1. The AUC plots, however, visualise the values and trade-offs of the model behaviour, should the threshold be varied.

VII. Results and Discussion

Table II presents the results for the performance of the machine learning methods and base deep neural network model on the multi-label task as of SLO violation prediction. The deep feed-forward neural network performs better than all other machine learning methods on most metrics, and is followed closely by the decision tree method. However, when comparing the averaged AUC-ROC for these models with the performance on individual class labels as shown in Figure 3, we notice that almost all the machine learning models have the ROC curves for one or more class labels lying on the diagonal, or close to the diagonal line, which represents a random classifier. This is especially true for the class label SLO_4, i.e. SLO_{io}, which none of the machine learning models learnt well. Hence, the values on precision, recall, and

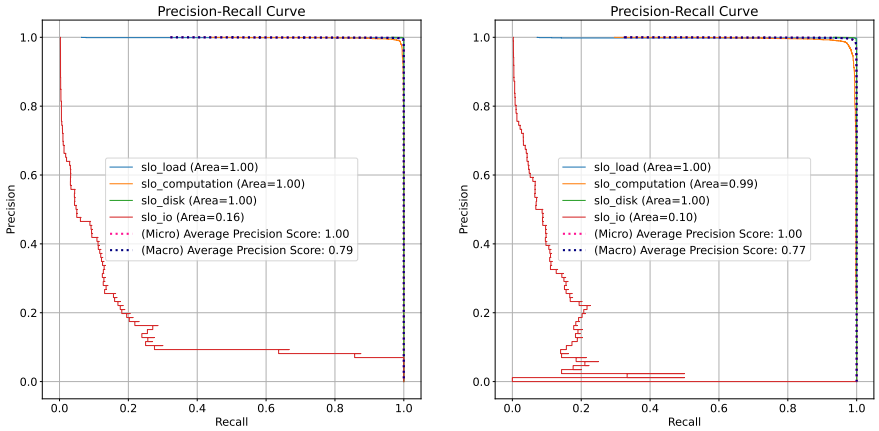
[4]www.docker.com



(a) Base Neural Network Classifier— Deep Feed-Forward Neural Network Model

(b) Deep Feed-Forward Neural Network Model— Balanced Class Weights

(c) Deep Feed-Forward Neural Network Model— Clipped Class Weights



(d) Deep Feed-Forward Neural Network Model— Log Smoothed Class Weights

(e) Deep Feed-Forward Neural Network Model— With Random Oversampling

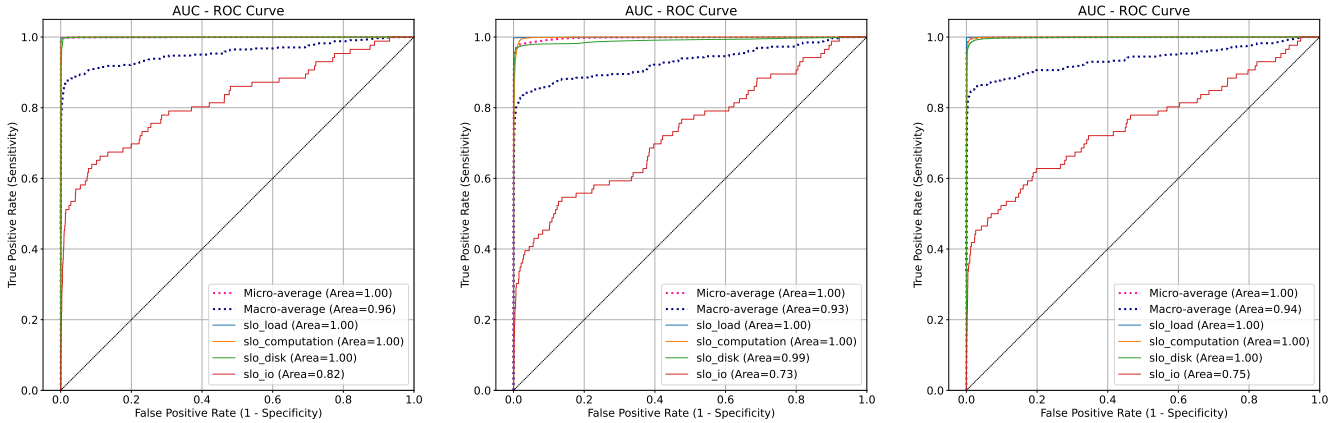
Fig. 5: AUC-PRC plots depicting the learning and precision-recall trade-off of the various deep neural network classifiers on individual class labels, as well as the micro and macro averaged performance on the entire set.

related example and label-based metrics stem from stochastic classification on some class labels, which is undesirable for the use-case here. Further, using such machine learning based methods is suitable only in the cases in which the class labels in the data either have a low level of correlation between them, or are independent altogether. However, it is clear from the comparison with the base deep neural network model that the class labels are correlated, and in a way that is not straightforward to learn given the level of imbalance in the multi-label dataset. Therefore, while binary relevance is the most straightforward in the way it handles the multi-label data, it is limited by its assumptions of complete label independence, which makes it unsuitable for advanced applications. The Classifier Chain method overcomes the limitations of the independent label assumption of the one-vs-rest binary relevance methods, and considers correlations among labels in a random manner. However, the chaining property still has its disadvantages in that it is not a parallel implementation, and the performance is highly dependent on finding the most appropriate order of chaining [73], which random ordering may not always solve. It is also very sensitive to skewed class distributions [74], and thus limited in applications depending on the use-case and the size and characteristics of data.

ML-kNN also assumes label independence, and moreover, takes a lot of time to train. It is therefore unsuited for data of large magnitude. Decision trees are yet another family of methods that assume label independence in solving multi-label problems. While the biggest advantage of using a decision tree methodology is that it is a white-box method and thus the decisions can be traced back to the logic behind them; on our use-case, the decision tree model holds a depth of 35 with 589 leaves. This conveys a high degree of model complexity that is not as easy to analyze and interpret. Random forest is an ensemble of decision trees, and thus prone to the same issues that decision trees face here.

The deep feed-forward neural network has a clear edge on the task, stemming from the fact that it is capable of modeling the patterns behind each of the class labels in parallel, and can concurrently capture the correlations between them. As shown in Figure 6a, the base FFNN model learns the behavior of all classes, including class SLO_4. However, a parallel view of the individual classes for the base FFNN model in Table IV shows that unlike the other class labels, SLO_{io} has no true positive or false positive detections. To address this, we

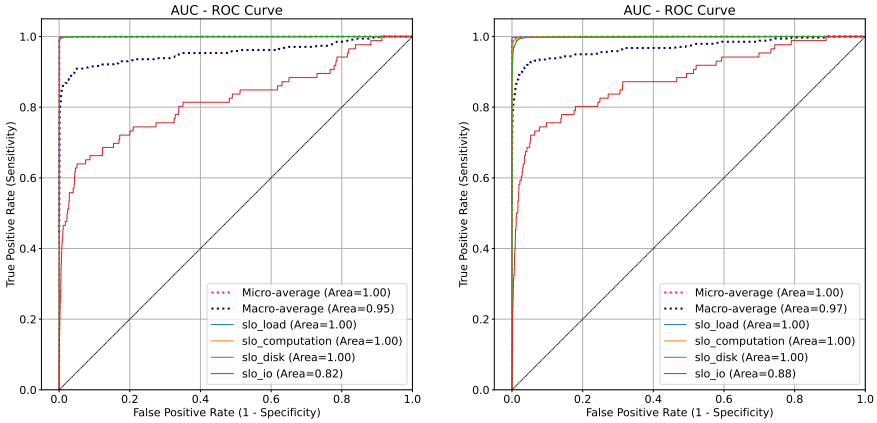
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/OJCOMS.2021.3122844, IEEE Open Journal of the Communications Society



(a) Base Neural Network Classifier— Deep Feed-Forward Neural Network Model

(b) Deep Feed-Forward Neural Network Model— Balanced Class Weights

(c) Deep Feed-Forward Neural Network Model— Clipped Class Weights



(d) Deep Feed-Forward Neural Network Model— Log Smoothed Class Weights

(e) Deep Feed-Forward Neural Network Model— With Random Oversampling

Fig. 6: AUC-ROC plots depicting the learning of the various deep neural network classifiers on individual class labels, as well as the micro and macro averaged performance on the entire set.

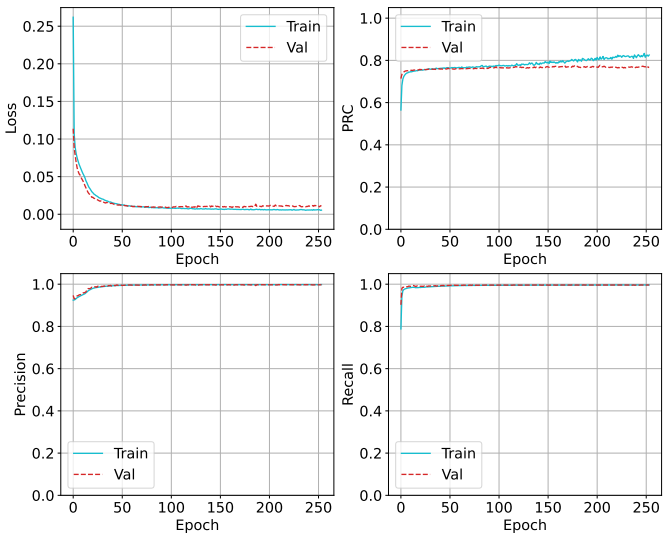


Fig. 7: Learning Curves depicting training and validation performance for Base Neural Network Classifier— Deep Feed-Forward Neural Network Model

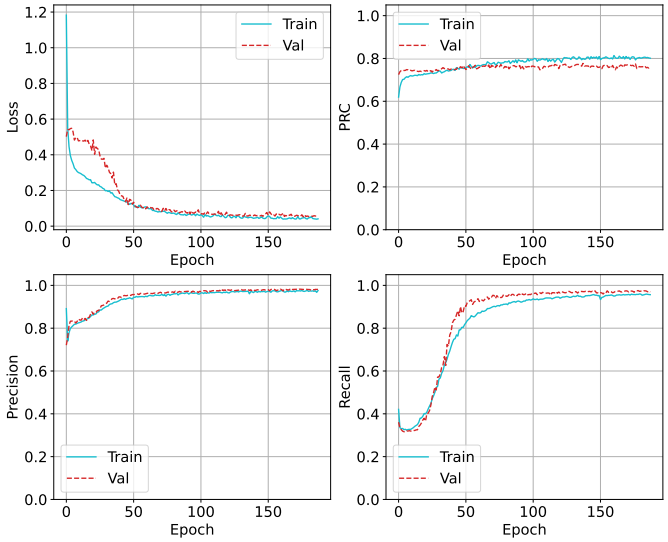


Fig. 8: Learning Curves depicting training and validation performance for Deep Feed-Forward Neural Network Model— Balanced Class Weights

TABLE IV: Performance of the various strategies adopted to address the class label imbalance, evaluated on all presented multi-label classification metrics

			Addressing Class Imbalance			
		Deep FFNN	**Deep FFNN with Balanced Class Weights**	**Deep FFNN with Clipped Class Weights**	**Deep FFNN with Log Smoothed Class Weights**	**Deep FFNN with Random Oversampling**
Training Loss		0.009	0.519	0.020	**0.010**	0.046
AUC-PRC	Micro Average	1.000	0.837	**0.999**	**0.999**	**0.999**
	Macro Average	0.778	0.741	0.771	**0.776**	0.773
	Weighted Average	0.998	0.952	0.997	**0.998**	0.997
AUC-ROC	Micro Average	1.000	0.953	0.999	**1.000**	**1.000**
	Macro Average	0.969	0.885	0.936	**0.959**	0.955
	Weighted Average	1.000	0.757	0.997	**0.999**	**0.999**
True Positives (TP)	SLO 1 (Load)	1973	1974	1976	1973	1972
	SLO 2 (Computation)	3499	3529	3517	3491	3499
	SLO 3 (Disk)	32568	6591	32168	32547	32528
	SLO 4 (IO)	0	37	17	13	31
		38040	12131	37678	38024	**38030**
False Positives (FP)	SLO 1 (Load)	1	9	18	0	1
	SLO 2 (Computation)	37	1818	240	35	93
	SLO 3 (Disk)	53	514	37	81	59
	SLO 4 (IO)	0	505	72	34	202
		91	2846	367	**150**	355
True Negatives (TN)	SLO 1 (Load)	33429	33421	33412	33430	33429
	SLO 2 (Computation)	31822	30041	31619	31824	31766
	SLO 3 (Disk)	2749	2288	2765	2721	2743
	SLO 4 (IO)	35322	34817	35250	35288	35120
		103322	100567	103046	**103263**	103058
False Negatives (FN)	SLO 1 (Load)	5	4	2	5	6
	SLO 2 (Computation)	50	20	32	58	50
	SLO 3 (Disk)	38	26015	438	59	78
	SLO 4 (IO)	86	49	69	73	55
		179	26088	541	195	**189**
Precision	Micro Average	0.998	0.810	0.990	**0.995**	0.991
	Macro Average	0.747	0.663	0.779	**0.813**	0.776
	Weighted Average	0.996	0.904	0.991	**0.994**	**0.994**
Recall	Micro Average	0.996	0.317	0.986	**0.996**	0.995
	Macro Average	0.747	0.656	0.794	0.775	**0.835**
	Weighted Average	0.996	0.317	0.986	**0.996**	0.995
F-1 Score	Micro Average	0.997	0.456	0.988	**0.996**	0.993
	Macro Average	0.747	0.560	0.786	0.787	**0.793**
	Weighted Average	0.996	0.409	0.988	**0.995**	0.994
Jaccard Similarity Coefficient	Micro Average	0.994	0.295	0.976	**0.991**	0.986
	Macro Average	0.744	0.478	0.753	0.764	**0.765**
	Weighted Average	0.994	0.282	0.978	**0.992**	0.991
(Subset) Accuracy (or EMR)		0.993	0.197	0.974	**0.990**	0.984
Hamming Loss		0.001	0.204	0.006	**0.002**	0.003
Log Loss		0.289	1.680	0.312	**0.294**	0.303
Subset Zero-One Loss		0.006	0.802	0.025	**0.009**	0.015
Coverage		1.080	1.186	1.081	**1.080**	**1.080**
Average Precision (Label Ranking)		0.999	0.946	**0.999**	**0.999**	**0.999**
Ranking Loss		0.0002	0.035	0.0006	**0.0003**	0.0004

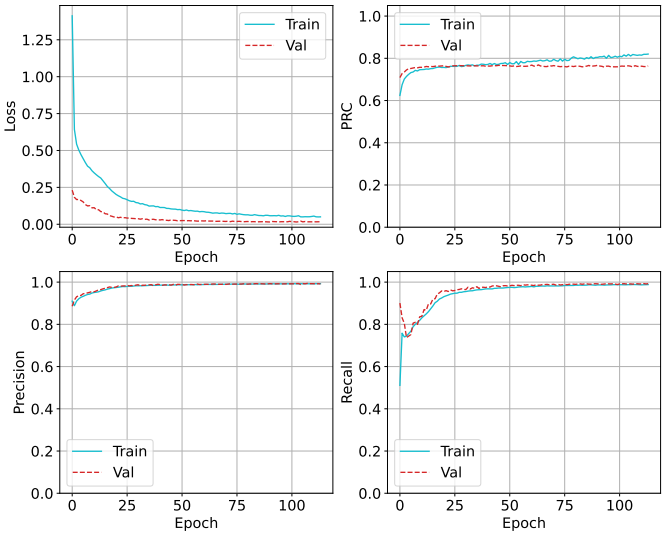


Fig. 9: Learning Curves depicting training and validation performance for Deep Feed-Forward Neural Network Model—Clipped Class Weights

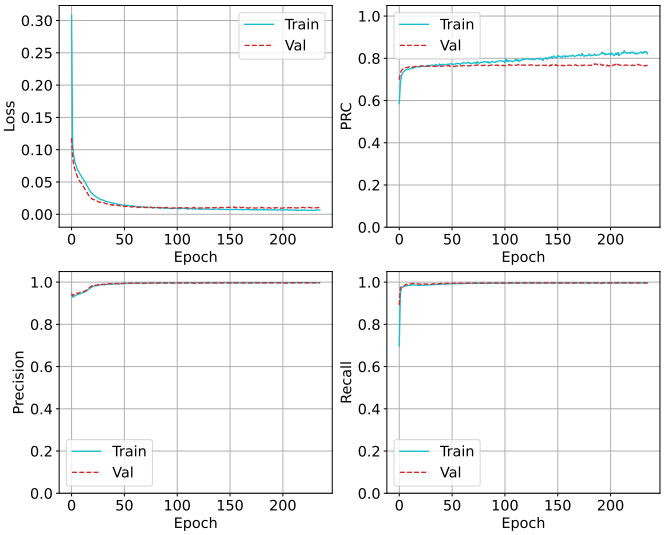


Fig. 10: Learning Curves depicting training and validation performance for Deep Feed-Forward Neural Network Model—Log Smoothed Class Weights

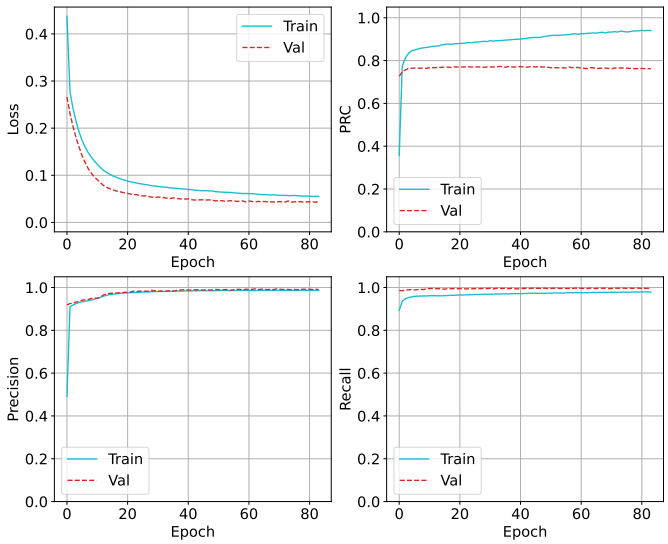


Fig. 11: Learning Curves depicting training and validation performance for Deep Feed-Forward Neural Network Model—With Random Oversampling

improve the model with class weighting and oversampling strategies, as elaborated earlier in Section V. Table IV presents the performance of these strategies as compared and evaluated on the multi-label classification metrics. A direct inference from the comparison is a quantification of how inflated the performance estimates from the base model were. By adding strategies to address class imbalance, we prevent the model from developing a bias towards the majority class labels, and ensure that the prediction is not an overestimation.

As mentioned earlier in Section V, rather than training to maximize accuracy or any other conventional metric, we train the deep neural network models to maximize the area under the precision recall curve (AUC-PRC). Figure 5 presents the PR curves for these deep FFNN models. A high AUC-PRC implies high recall and high precision, i.e. low false negative rate and low false positive rate respectively.

Figure 6 depicts the ROC curves for the individual class labels. Any point on the curve here signifies a trade-off between precision and recall, should that be the threshold—if set low, the recall of the positive occurrence (class value 1) will be high, and the precision will be low; and vice-versa if the threshold is set high. Which one to prioritize depends on the use-case— for example in our case the model can be tolerant of false positives at the cost of minimizing the false negatives, since false negatives signify missed SLO violations, which have a higher cost associated with them. Thus, in our use-case, we prioritize recall over precision when measuring performance.

Figures 7, 8, 9, 10, and 11 show the learning curves depicting loss, AUC-PRC, precision, and recall performance for the corresponding deep FFNN models as they train to converge over the training and validation sets. When comparing the class weighting strategy, we notice the fluctuations when the model assumes large weights as per the balanced class weighting strategy. This is because the model sees an infrequent example associated with a very large weight within some batch of training, which suddenly disrupts the gradient signal at each occurrence. This also transfers to poor performance for all evaluations on the test set. Clipping the class weights to a lower range improves both model training and performance, thus necessitating that formal class weighting strategies that enforce large weights given a high ratio of imbalance are unsuitable in this context. The best performance of the model is achieved when the weights are smoothed by log scaling, and kept to a minimal ratio. The log smoothed model outperforms on almost all categories of metrics with the best convergence and least overfitting, and achieves the highest macroaveraged AUC-PRC (0.776) and AUC-ROC (0.959), with a subset accuracy of 99%, and multi-label accuracy of 99.1%.

As can be seen in Figure 11, the model with random oversampling does overfit given the repetitions in the training set, but the appropriate regularization ensures that the validation curve is smooth, and the model converges well. The model has the highest macroaveraged F_1 score (0.793) and recall (0.835), and fares almost at par with the log weighted model in learning and perform well on all classes. However, while it minimizes the false negatives the most, it has a slightly higher number of false positives. The exact classification distributions for the individual class labels are depicted in Table IV. Ultimately, the choice between opting for sampling based strategies or class weighting strategies depends on the degree of imbalance, the model being used, the use-case and objectives. In the case of a deep FFNN model, when the class weights enforced are small, both class weighting and oversampling work similarly, assigning the equivalent of small weights to infrequent positive examples in individual batches during training. However, when the class weights formally attain large values, the results suggest that oversampling may be better strategy, since it involves a smoother gradient update in each batch seen during training.

VIII. Conclusion and Future Work

In this work, we address the problem of SLA and SLO violation prediction in an NFV environment with the use of multi-label classification methodology. This enables the incorporation of multi-output models as we move towards more complex decision-making in the management of virtualised communication networks, identifying and predicting multiple categories of SLO breaches as applicable to study and mitigate their impact towards SLA and SLO violations. We work with Clearwater, a latency-sensitive NFV based vIMS application to draft realistic SLO definitions for this vertical, and use these as the basis to model the violations as a multi-output target. We propose the use of a deep feed-forward neural network classifier to adequately capture and learn the correlations between the different categories of SLO violations, and predict them as they (co)-occur given the state of the NFV application at a point in time.

The results suggest the suitability of such a deep learning methodology in achieving the target objective, and in also overcoming the issue of class imbalance in training by adapting class weighting and random oversampling strategies to a

multi-label setup. We achieve a subset accuracy of 99%, and multi-label accuracy of 99.1% in the best model approach, working with a dataset where the highest class label imbalance ratio is 395.18, with a mean imbalance ratio of 105.40.

We reason and demonstrate that our proposed methodology can be useful to identify the gaps in SLA policy enforcement, to further fine-tune the scaling policies for an enhanced balance between efficiency and reliability, as well as to identify and address the frequent vulnerabilities and bottlenecks that a latency-sensitive real-time application such as this may face. In future work, we plan to integrate our approach with a traffic and workload forecasting methodology for a higher degree of detail in proactive violations' prediction, and to combine this with dynamic policy enforcement for an end-to-end management control loop.

APPENDIX

Figures 12 and 13 provide a consolidated view of the grid search that was performed to find suitable weight regularization strategy and hyperparameters for the deep FFNN models. Table V provides an indication of the random search for an optimal FFNN architecture for the use-case here. Figures 14 and 15 give an overview of the degree of overfitting in effect in each of the trained neural network models.

ACKNOWLEDGMENT

The authors would like to thank Dr. Imen Grida Ben Yahia and Dr. Jaafar Bendriss for sharing their data from the Clearwater deployment for academic and public usage.

REFERENCES

[1] ITU, "IMT traffic estimates for the years 2020 to 2030, ITU-R M.2370-0," tech. rep., International Telecommunication Union, Geneva, Switzerland, July 2015. Available: https://bit.ly/3g828Ux.

[2] Ericsson, "Mobility Report," tech. rep., Stockholm, Sweden, Nov. 2020. Available: https://bit.ly/3gmcI9E.

[3] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network Function Virtualization: State-of-the-Art and Research Challenges," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 236–262, 2016.

[4] W. Jiang, B. Han, M. A. Habibi, and H. D. Schotten, "The Road Towards 6G: A Comprehensive Survey," *IEEE Open Journal of the Communications Society*, vol. 2, pp. 334–366, 2021.

[5] T. Zhang, H. Qiu, L. Linguaglossa, W. Cerroni, and P. Giaccone, "NFV Platforms: Taxonomy, Design Choices and Future Challenges," *IEEE Transactions on Network and Service Management*, vol. 18, pp. 30–48, Mar. 2021.

[6] M. Di Mauro, G. Galatro, M. Longo, F. Postiglione, and M. Tambasco, "IP Multimedia Subsystem in a containerized environment: availability and sensitivity evaluation," in *2019 IEEE Conference on Network Softwarization (NetSoft)*, (Paris, France), pp. 42–47, IEEE, June 2019.

[7] N. Jalodia, S. Henna, and A. Davy, "Deep Reinforcement Learning for Topology-Aware VNF Resource Prediction in NFV Environments," in *2019 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, (Dallas, TX, USA), pp. 1–5, IEEE, Nov. 2019.

[8] "Dynamic scaling for Amazon EC2 Auto Scaling - Amazon Web Services (AWS)." https://amzn.to/3yPU963, accessed May 2021.

[9] "Predictive scaling for Amazon EC2 Auto Scaling - Amazon Web Services (AWS)." https://amzn.to/3c5pQPc, accessed May 2021.

[10] "Predictive Scaling for EC2 with Machine Learning - Amazon Web Services (AWS)." https://amzn.to/3vBHlyg, accessed May 2021.

[11] "Using Predictive Autoscaling - Google Cloud." https://bit.ly/3vFc5hH, accessed May 2021.

[12] B. Yi, X. Wang, K. Li, S. k. Das, and M. Huang, "A comprehensive survey of Network Function Virtualization," *Computer Networks*, vol. 133, pp. 212–262, Mar. 2018.

[13] R. Mijumbi, S. Hasija, S. Davy, A. Davy, B. Jennings, and R. Boutaba, "Topology-Aware Prediction of Virtual Network Function Resource Requirements," *IEEE Transactions on Network and Service Management*, vol. 14, pp. 106–120, Mar. 2017.

[14] G. Tsoumakas, I. Katakis, and I. Vlahavas, "Mining Multi-label Data," in *Data Mining and Knowledge Discovery Handbook* (O. Maimon and L. Rokach, eds.), pp. 667–685, Boston, MA: Springer US, 2009.

[15] D. Xu, Y. Shi, I. W. Tsang, Y.-S. Ong, C. Gong, and X. Shen, "Survey on Multi-Output Learning," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–21, 2019.

[16] S. Cherrared, S. Imadali, E. Fabre, and G. Goessler, "LUMEN: A global fault management framework for network virtualization environments," in *2018 21st Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*, (Paris), pp. 1–8, IEEE, Feb. 2018.

[17] J. Suomalainen, A. Juhola, S. Shahabuddin, A. Mammela, and I. Ahmad, "Machine Learning Threatens 5G Security," *IEEE Access*, vol. 8, pp. 190822–190842, 2020.

[18] S. Cherrared, S. Imadali, E. Fabre, G. Gossler, and I. G. B. Yahia, "A Survey of Fault Management in Network Virtualization Environments: Challenges and Solutions," *IEEE Transactions on Network and Service Management*, vol. 16, pp. 1537–1551, Dec. 2019.

[19] X. Zheng, N. Huang, S. Yin, G. Wen, and X. Zhang, "A Service Deployment Method Considering Application Reliability of Networks," *IEEE Access*, vol. 9, pp. 28505–28513, 2021.

[20] B. Tola, G. Nencioni, and B. E. Helvik, "Network-Aware Availability Modeling of an End-to-End NFV-Enabled Service," *IEEE Transactions on Network and Service Management*, vol. 16, pp. 1389–1403, Dec. 2019.

[21] M. Di Mauro, M. Longo, F. Postiglione, G. Carullo, and M. Tambasco, "Service function chaining deployed in an NFV environment: An availability modeling," in *2017 IEEE Conference on Standards for Communications and Networking (CSCN)*, (Helsinki, Finland), pp. 42–47, IEEE, Sept. 2017.

[22] A. Binsahaq, T. R. Sheltami, and K. Salah, "A Survey on Autonomic Provisioning and Management of QoS in SDN Networks," *IEEE Access*, vol. 7, pp. 73384–73435, 2019.

[23] L.-V. Le, D. Sinh, B.-S. P. Lin, and L.-P. Tung, "Applying Big Data, Machine Learning, and SDN/NFV to 5G Traffic Clustering, Forecasting, and Management," in *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, (Montreal, QC), pp. 168–176, IEEE, June 2018.

[24] A. A. Gebremariam, M. Usman, and M. Qaraqe, "Applications of Artificial Intelligence and Machine Learning in the Area of SDN and NFV: A Survey," in *2019 16th International Multi-Conference on Systems, Signals & Devices (SSD)*, (Istanbul, Turkey), pp. 545–549, IEEE, Mar. 2019.

[25] J. Vergara-Reyes, M. C. Martinez-Ordonez, A. Ordonez, and O. M. Caicedo Rendon, "IP traffic classification in NFV: A benchmarking of supervised Machine Learning algorithms," in *2017 IEEE Colombian Conference on Communications and Computing (COLCOM)*, (Cartagena), pp. 1–6, IEEE, Aug. 2017.

[26] G. Ilievski and P. Latkoski, "Efficiency of Supervised Machine Learning Algorithms in Regular and Encrypted VoIP Classification within NFV Environment," *Radioengineering*, vol. 29, pp. 243–250, Apr. 2020.

[27] C. Sun, J. Bi, Z. Zheng, and H. Hu, "SLA-NFV: an SLA-aware High Performance Framework for Network Function Virtualization," in *Proceedings of the 2016 ACM SIGCOMM Conference*, (Florianopolis Brazil), pp. 581–582, ACM, Aug. 2016.

[28] E. Kapassa, M. Touloupou, and D. Kyriazis, "SLAs in 5G: A Complete Framework Facilitating VNF- and NS- Tailored SLAs Management," in *2018 32nd International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, (Krakow), pp. 469–474, IEEE, May 2018.

[29] I. G. Ben Yahia, J. Bendriss, A. Samba, and P. Dooze, "CogNitive 5G networks: Comprehensive operator use cases with machine learning for management operations," in *2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN)*, (Paris), pp. 252–259, IEEE, Mar. 2017.

[30] J. Bendriss, I. G. Ben Yahia, P. Chemouil, and D. Zeghlache, "AI for SLA Management in Programmable Networks," in *DRCN 2017 - Design of Reliable Communication Networks; 13th International Conference*, pp. 1–8, 2017.

[31] M. Boucadair, C. Jacquenet, and X. Xu, eds., *Emerging Automation Techniques for the Future Internet:*. Advances in Wireless Technologies and Telecommunication, IGI Global, 2019.

[32] J. Bendriss, *Cognitive management of SLA in software-based networks*. Theses, Institut National des Télécommunications, June 2018. Issue: 2018E0003.

[33] J. Bendriss, I. G. Ben Yahia, and D. Zeghlache, "Forecasting and anticipating SLO breaches in programmable networks," in *2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN)*, (Paris), pp. 127–134, IEEE, Mar. 2017.

[34] E. Gibaja and S. Ventura, "A Tutorial on Multilabel Learning," *ACM Computing Surveys*, vol. 47, pp. 1–38, Apr. 2015.

[35] M. S. Sorrower, "A literature survey on algorithms for multi-label learning," pp. 1–25, 2010.

[36] G. Madjarov, D. Kocev, D. Gjorgjevikj, and S. Džeroski, "An extensive experimental comparison of methods for multi-label learning," *Pattern Recognition*, vol. 45, pp. 3084–3104, Sept. 2012.

[37] M.-L. Zhang and Z.-H. Zhou, "A Review on Multi-Label Learning Algorithms," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, pp. 1819–1837, Aug. 2014.

[38] M. A. Tahir, J. Kittler, and F. Yan, "Inverse random under sampling for class imbalance problem and its application to multi-label classification," *Pattern Recognition*, vol. 45, pp. 3738–3750, Oct. 2012.

[39] J. Shen, S. Li, F. Jia, H. Zuo, and J. Ma, "A Deep Multi-Label Learning Framework for the Intelligent Fault Diagnosis of Machines," *IEEE Access*, vol. 8, pp. 113557–113566, 2020.

[40] W. Hong, W. Xu, J. Qi, and Y. Weng, "Neural Tensor Network for Multi-Label Classification," *IEEE Access*, vol. 7, pp. 96936–96941, 2019.

[41] A. Maxwell, R. Li, B. Yang, H. Weng, A. Ou, H. Hong, Z. Zhou, P. Gong, and C. Zhang, "Deep learning architectures for multi-label classification of intelligent health risk prediction," *BMC Bioinformatics*, vol. 18, p. 523, Dec. 2017.

[42] J. Mandziuk and A. Zychowski, "Dimensionality Reduction in Multilabel Classification with Neural Networks," in *2019 International Joint Conference on Neural Networks (IJCNN)*, (Budapest, Hungary), pp. 1–8, IEEE, July 2019.

[43] M. Ibrahim, M. Torki, and N. El-Makky, "Imbalanced Toxic Comments Classification Using Data Augmentation and Deep Learning," in *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, (Orlando, FL), pp. 875–878, IEEE, Dec. 2018.

[44] B. Beyer, C. Jones, J. Petoff, and N. Murphy, *Site Reliability Engineering: How Google Runs Production Systems*. O'Reilly Media, Incorporated, 2016.

[45] "3GPP - The 3rd Generation Partnership Project, A Global Initiative." https://www.3gpp.org/, accessed May 2021.

[46] "Auto Scaling Overview - Amazon Web Services (AWS)." https://amzn.to/3fzX3UJ, accessed May 2021.

[47] "Auto Scaling Amazon EC2 - Amazon Web Services (AWS)." https://amzn.to/3wKtYM7, accessed May 2021.

[48] "Autoscale in Microsoft Azure - Azure Monitor." https://bit.ly/3wLoMrv, accessed May 2021.

[49] "Autoscaling - Google Cloud." https://bit.ly/3wJxG8J, accessed May 2021.

[50] "Auto Scaling - Huawei Cloud." https://bit.ly/34xww4f, accessed May 2021.

[51] R. Livni, S. Shalev-Shwartz, and O. Shamir, "On the computational efficiency of training neural networks," in *Advances in neural information processing systems* (Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, eds.), vol. 27, Curran Associates, Inc., 2014.

[52] X.-Z. Wu and Z.-H. Zhou, "A Unified View of Multi-Label Performance Measures," in *Proceedings of the 34th International Conference on Machine Learning* (D. Precup and Y. W. Teh, eds.), vol. 70 of *Proceedings of Machine Learning Research*, pp. 3780–3788, PMLR, Aug. 2017.

[53] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[54] T. Calders and S. Jaroszewicz, "Efficient AUC Optimization for Classification," in *Knowledge Discovery in Databases: PKDD 2007* (J. N. Kok, J. Koronacki, R. Lopez de Mantaras, S. Matwin, D. Mladenič, and A. Skowron, eds.), vol. 4702, pp. 42–53, Berlin, Heidelberg: Springer Berlin Heidelberg, 2007. ISSN: 0302-9743, 1611-3349 Series Title: Lecture Notes in Computer Science.

[55] F. Charte, A. Rivera, M. J. del Jesus, and F. Herrera, "A First Approach to Deal with Imbalance in Multi-label Datasets," in *Hybrid Artificial Intelligent Systems* (D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, J.-S. Pan, M. M. Polycarpou, M. Woźniak, A. C. P. L. F. de Carvalho, H. Quintián, and E. Corchado, eds.), vol. 8073, pp. 150–160, Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. Series Title: Lecture Notes in Computer Science.

[56] F. Charte, A. J. Rivera, M. J. del Jesus, and F. Herrera, "Addressing imbalance in multilabel classification: Measures and random resampling algorithms," *Neurocomputing*, vol. 163, pp. 3–16, Sept. 2015.

[57] J. Davis and M. Goadrich, "The relationship between Precision-Recall and ROC curves," in *Proceedings of the 23rd international conference on Machine learning - ICML '06*, (Pittsburgh, Pennsylvania), pp. 233–240, ACM Press, 2006.

[58] T. Saito and M. Rehmsmeier, "The Precision-Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets," *PLOS ONE*, vol. 10, p. e0118432, Mar. 2015.

[59] S. Wang, W. Liu, J. Wu, L. Cao, Q. Meng, and P. J. Kennedy, "Training deep neural networks on imbalanced data sets," in *2016 International Joint Conference on Neural Networks (IJCNN)*, (Vancouver, BC, Canada), pp. 4368–4374, IEEE, July 2016.

[60] B. Liu and G. Tsoumakas, "Synthetic Oversampling of Multi-label Data Based on Local Label Distribution," in *Machine Learning and Knowledge Discovery in Databases* (U. Brefeld, E. Fromont, A. Hotho, A. Knobbe, M. Maathuis, and C. Robardet, eds.), vol. 11907, pp. 180–193, Cham: Springer International Publishing, 2020. Series Title: Lecture Notes in Computer Science.

[61] J. Ma, H. Zhang, and T. W. S. Chow, "Multilabel Classification With Label-Specific Features and Classifiers: A Coarse- and Fine-Tuned Framework," *IEEE Transactions on Cybernetics*, vol. 51, pp. 1028–1042, Feb. 2021.

[62] Y. Zhong, B. Du, and C. Xu, "Learning to reweight examples in multi-label classification," *Neural Networks*, p. S0893608021001106, Apr. 2021.

[63] A. N. Tarekegn, M. Giacobini, and K. Michalak, "A review of methods for imbalanced multi-label classification," *Pattern Recognition*, vol. 118, p. 107965, Oct. 2021.

[64] M.-L. Zhang, Y.-K. Li, H. Yang, and X.-Y. Liu, "Towards Class-Imbalance Aware Multi-Label Learning," *IEEE Transactions on Cybernetics*, pp. 1–13, 2020.

[65] B. Liu and G. Tsoumakas, "Making Classifier Chains Resilient to Class Imbalance," in *Proceedings of The 10th Asian Conference on Machine Learning* (J. Zhu and I. Takeuchi, eds.), vol. 95 of *Proceedings of Machine Learning Research*, pp. 280–295, PMLR, Nov. 2018.

[66] H. He and Y. Ma, eds., *Imbalanced learning: foundations, algorithms, and applications*. Hoboken, New Jersey: John Wiley & Sons, Inc, 2013.

[67] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[68] A. Fernández, S. García, M. Galar, R. C. Prati, B. Krawczyk, and F. Herrera, *Learning from Imbalanced Data Sets*. Cham: Springer International Publishing, 2018.

[69] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Y. Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng, *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015.

[70] F. Chollet and others, *Keras*. 2015.

[71] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014.

[72] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2015.

[73] J. Read, B. Pfahringer, G. Holmes, and E. Frank, "Classifier Chains for Multi-label Classification," in *Machine Learning and Knowledge Discovery in Databases* (W. Buntine, M. Grobelnik, D. Mladenić, and J. Shawe-Taylor, eds.), vol. 5782, pp. 254–269, Berlin, Heidelberg:

Springer Berlin Heidelberg, 2009. Series Title: Lecture Notes in Computer Science.

[74] J. Read, B. Pfahringer, G. Holmes, and E. Frank, "Classifier Chains: A Review and Perspectives," *Journal of Artificial Intelligence Research*, vol. 70, pp. 683–718, Feb. 2021.

Nikita Jalodia is currently a Ph.D. Researcher with the Department of Computing and Mathematics at the Emerging Networks Lab Research Division in Walton Institute of Information and Communication Systems Science, Waterford Institute of Technology, Ireland. She joined in July 2017, and has since been working as a part of the Science Foundation Ireland funded CONNECT Research Centre for Future Networks and Communications. Her current research interests include Data Science, Network Function Virtualization (NFV), Machine and Deep Learning, Knowledge-Defined Networks, Internet of Things (IoT), and Fog and Cloud Computing. She received her Bachelor's Degree in Computer Science and Engineering from The LNM Institute of Information Technology, Jaipur, India in 2017, with a specialization in Big Data and Analytics with IBM. She has also previously worked as a Software Developer at Publicis (Sapient) Global Markets, India.

Mohit Taneja is currently a Lecturer in Business Information Systems, Data Science and Computing with the Department of Accountancy and Economics, School of Business at Waterford Institute of Technology. Previously, he was a Postdoctoral Research Fellow with the Programmable Autonomous Systems Division of Walton Institute; working on EU, National, International, and Industry funded projects. He worked as the Project and Tech Lead on EU-H2020 funded Smart Cities 2030 project. In Walton, he also worked within the Strategic Division at Walton, liaising with all the Research Divisions within the Institute. He has also been associated with the SFI funded VistaMilk Research Centre and CONNECT Centre. He has previously worked as an Experienced Software Research Engineer with the Emerging Networks Lab (ENL) division. He earned his Ph.D. from Waterford Institute of Technology, Ireland in 2020. The title of his dissertation was 'Fog Computing Support for Internet of Things Applications'. He was also a visiting research fellow at IBM Research Labs, Ireland from 2017-2018. His current research interests include Fog and Cloud Computing, IoT, Distributed Systems, and Distributed Data Analytics. He received his Bachelor's Degree in Computer Science and Engineering from The LNM Institute of Information Technology, Jaipur, India in 2015.

Alan Davy is currently the Head of the Department of Computing and Mathematics, School of Science and Computing, Waterford Institute of Technology (WIT), Waterford, Ireland. Previously, he was the Research Division Manager of the Emerging Networks Laboratory in WIT's Walton Institute of Information and Communication Systems Science. He has been the Coordinator and Principal Investigator on a number of National, International, and EU projects such as TERAPOD, SFI-TIDA, 5GinFIRE's C2G-RAN, etc. He was awarded B.Sc. (with Hons.) in Applied Computing and Ph.D. degrees from the Waterford Institute of Technology, Waterford, Ireland, in 2002 and 2008 respectively. He has been a recipient of the Marie Curie International Mobility Fellowship in 2010, and has worked at the Universitat Politècnica de Catalunya, and IIT Madras in India. His current research interests include Virtualised Telecom Networks, Fixed and Wireless Network Management, Software Defined Infrastructure, Internet of Things, Fog and Edge Computing, Bio-inspired Systems, Molecular/Nano-scale Communications, and TeraHertz Communications.

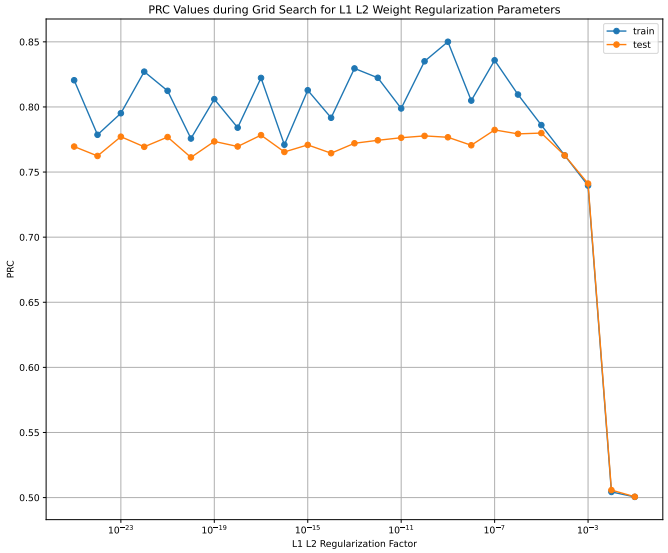


Fig. 12: Grid search for L1 L2 weight regularization on the deep FFNN architecture.

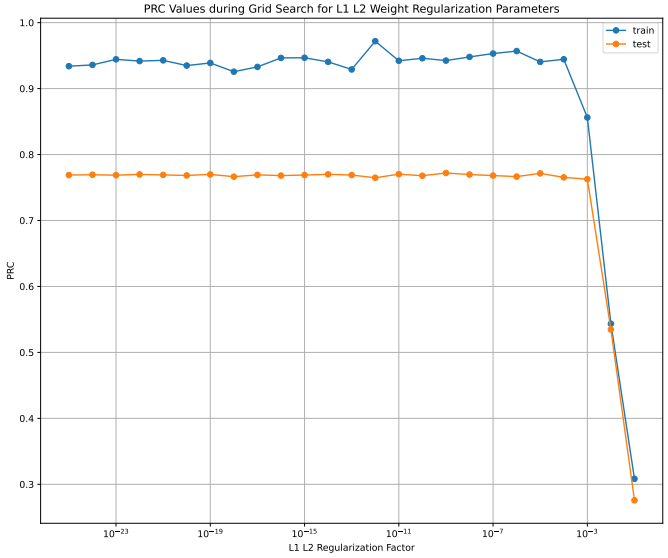
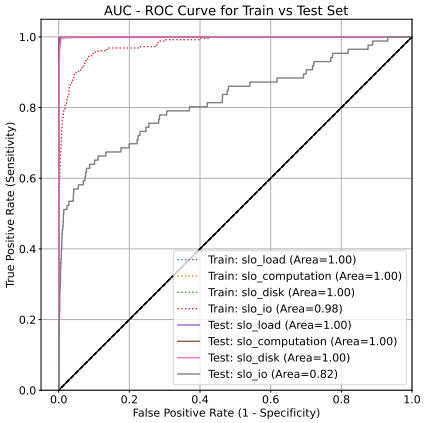


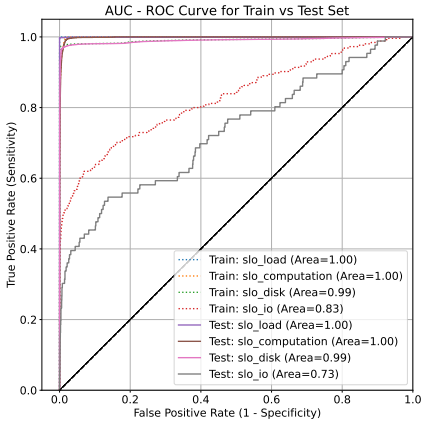
Fig. 13: Grid search for L1 L2 weight regularization on the deep FFNN architecture used for the oversampled training set.

TABLE V: Key results from random search for finding an optimal neural network architecture.

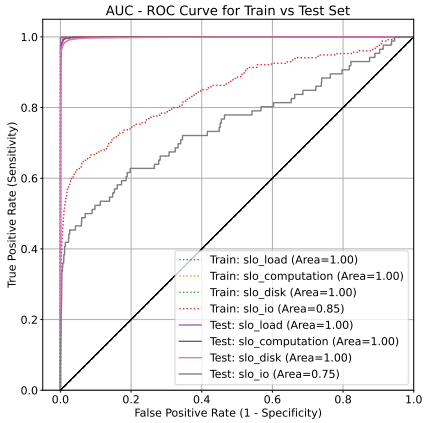
Results of Random Search for Neural Network Architecture											
Neural Network Layers, and Number of Neurons in Each			Loss	TP	FP	TN	FN	Accuracy	Precision	Recall	AUC
Input Layer	Hidden Layers	Output Layer									
126	65, 0.2 (D)	4	0.015	37973	366	103047	246	0.995	0.990	0.993	**0.999**
	65, 0.2 (D), 24		0.010	38043	139	103274	176	**0.997**	**0.996**	**0.995**	**0.999**
	65, 0.2 (D), 65		0.014	37960	248	103165	259	0.996	0.993	0.993	**0.999**
	65, 0.2 (D), 24, 0.1 (D)		0.011	38006	130	103283	213	**0.997**	**0.996**	0.994	**0.999**
	65, 0.2 (D), 65, 0.1 (D), 24, 0.1 (D), 24		0.010	38041	161	103252	178	**0.997**	0.995	**0.995**	**0.999**
	65, 0.3 (D), 65, 0.2 (D), 24, 0.1 (D), 24		**0.008**	38035	**118**	**103295**	184	**0.997**	**0.996**	**0.995**	**0.999**
	65, 0.3 (D), 65, 0.2 (D), 24, 0.2 (D), 24		0.010	**38050**	155	103258	**169**	**0.997**	0.995	**0.995**	**0.999**



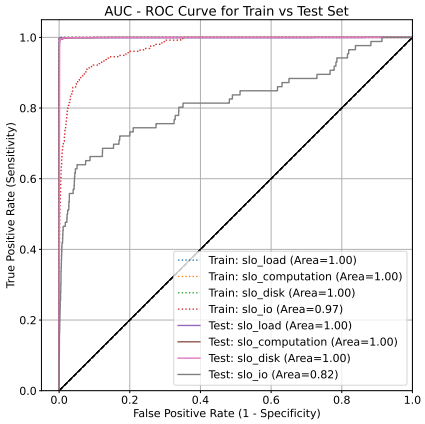
(a) Base Neural Network Classifier— Deep Feed-Forward Neural Network Model



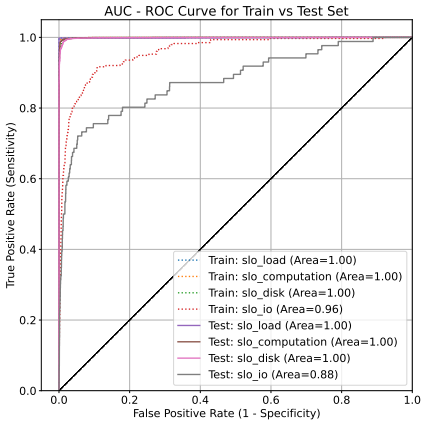
(b) Deep Feed-Forward Neural Network Model— Balanced Class Weights



(c) Deep Feed-Forward Neural Network Model— Clipped Class Weights

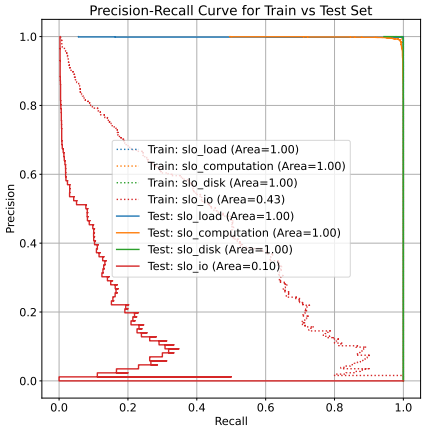


(d) Deep Feed-Forward Neural Network Model— Log Smoothed Class Weights

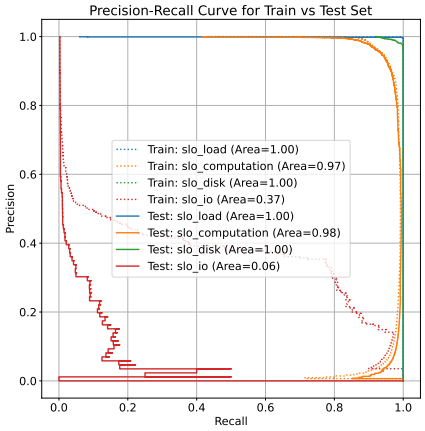


(e) Deep Feed-Forward Neural Network Model— With Random Oversampling

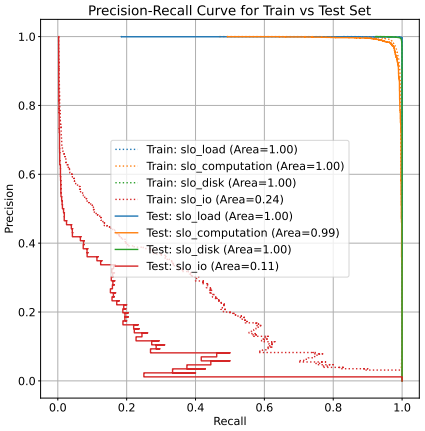
Fig. 14: AUC-ROC plots depicting the performance of the various deep neural network classifiers on individual class labels on the Train vs Test set, signifying the degree of overfitting.



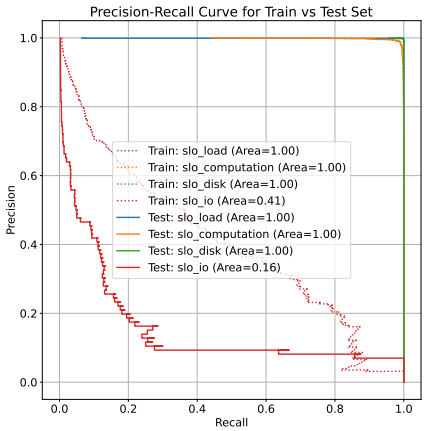
(a) Base Neural Network Classifier— Deep Feed-Forward Neural Network Model



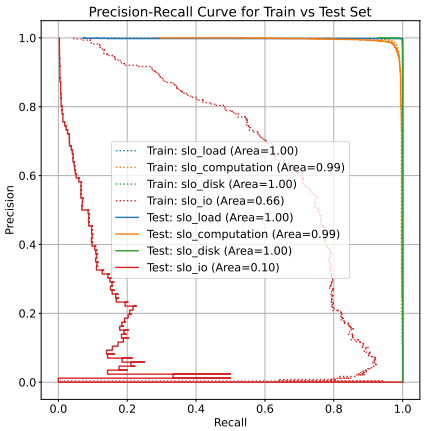
(b) Deep Feed-Forward Neural Network Model— Balanced Class Weights



(c) Deep Feed-Forward Neural Network Model— Clipped Class Weights



(d) Deep Feed-Forward Neural Network Model— Log Smoothed Class Weights



(e) Deep Feed-Forward Neural Network Model— With Random Oversampling

Fig. 15: AUC-PRC plots depicting the performance of the various deep neural network classifiers on individual class labels on the Train vs Test set, signifying the degree of overfitting.