Machine Learning-based Prediction Models for Control Traffic in SDN Systems

Yeonho Yoo, *Graduate Student Member, IEEE,* Gyeongsik Yang, *Member, IEEE,* Changyong Shin, *Graduate Student Member, IEEE,* Junseok Lee, and Chuck Yoo, *Member, IEEE*

Abstract—This paper presents *Elixir*, an automated prediction model formulation framework for control traffic using machine learning. Control traffic is vital in software-defined networking (SDN) systems because it determines the reliability and scalability of the entire system. Various studies have sought to design control traffic prediction models for the proper provisioning and planning of SDN systems. However, previously proposed models are based on descriptive modeling, well-suited for only specific SDN system instances. Furthermore, these models exhibit poor accuracy (errors of up to 85%) because of the heterogeneity of SDN systems. Because descriptive modeling requires a significant amount of human contemplation, it is impossible to formulate adequate prediction models for countless SDN system instances. *Elixir* addresses this problem by applying machine learning. *Elixir* starts the model formulation through self-generated datasets. Then, *Elixir* searches prediction models to fit the accuracy for respective SDN systems. Also, *Elixir* picks robust models that exhibit reasonable accuracy even in a network topology that differs from the topology used for model training. We evaluate the *Elixir* framework on nine heterogeneous SDN systems. As a key outcome, *Elixir* significantly reduces prediction errors, achieving up to 10.6× improvement compared to the previous model for control traffic throughput of OpenDayLight controller.

Index Terms—Machine learning, Software-defined networking, Control traffic, Prediction model formulation, Prediction robustness

1 Introduction

SOFTWARE-defined networking (SDN) is a network system that disaggregates control and data planes of network switches and centralizes the control planes into the SDN controller. The SDN controller governs all the SDN switch activities through control applications that are modules of the SDN controller. Control applications implement the policies for network governing, such as routing, traffic monitoring, and firewall. The decisions generated from the control applications are communicated and realized in SDN switches through control traffic between an SDN controller and switches. The definition (syntax) of control traffic is referred to as south-bound interface (SBI).

SDN allows control plane components (i.e., SDN controller, control applications, and SBI) to be independently selected. The control plane can govern any data plane, such as network topologies of switches and their traffic, because the two planes are disaggregated. These benefits flourish the adoption of SDNs in various sectors. For example, Google manages its datacenter networks [1] and wide-area networks between datacenters [2] through Orion, a custom SDN controller [3]. It uses OpenFlow (OF) as its SBI. Also, the SDN concept is used for various open-source network platforms, such as SD-Fabric for edge computing by ONOS controller [4] and Linux foundation's OPNFV for network function virtualization by OpenDayLight (ODL) controller [5]. Besides, SDN systems are deployed in cable networks of COMCAST and broadband access of AT&T [6].

In SDN systems, control plane performance, especially control traffic, is a critical factor [7], as a bottleneck in control traffic can slow down all network operations, and in the worst case, it may cease to operate. Too much control traffic, in particular, could severely delay the SDN controller's operations (e.g., topology discovery) or even cause its failure [8]. In addition, on the SDN switch side, heavy control traffic, such as flow statistics monitoring and flow rule setup, can significantly slow down packet forwarding [9].

Such bottlenecks have been reported in real-world SDN systems as well. Depending on the amount of control traffic, the utilization of SDN switches can deteriorate significantly, ranging from 20% to 90%, even when managing the identical physical networks [10]. The amount of control traffic determines the energy efficiency, scalability, and reliability of fault handling in real SDN systems [11].

Therefore, it is very important to design and configure the target SDN systems properly, but it is widely known that such design and configuration is extremely difficult due to the complexity of SDN systems. This paper proposes, for the first time, to predict control traffic for the proper design and configuration of the target SDN systems.

Several studies have proposed to predict control traffic as follows [12], [13], [14]. First, previous studies have been confined to a single SDN system's control plane of which they aim to predict the control traffic. The target SDN system instances (i.e., SDN controller, control applications, and SBI) is considered as a stochastic system. Then, causal relationships between the control traffic and the elements of the system (e.g., number of hosts and switches) are identified.

- *Yeonho Yoo and Gyeongsik Yang are the co-first authors who contributed equally to this work. Chuck Yoo is the corresponding author.*
- *All authors are with the Department of Computer Science and Engineering, Korea University, Seoul, Republic of Korea, 02841. E-mail: {yhyoo, ksyang, cyshin, jslee, chuckyoo}@os.korea.ac.kr*
- *This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2021R1A6A1A13044830), by the NRF grant funded by the Korea government (MSIT) (NRF-2023R1A2C3004145), by the Google Cloud Research Credits program, and by a Korea University Grant.*

Manuscript received January 6, 2023; revised xx xx, 2023.

The observed causal relationships are translated into a prediction model. For example, the total throughput or number of control traffic messages is derived by multiplying the message count and length per message type and their sum. This model formulation is known as "descriptive modeling" (DM) [15], [16].

However, the DM approach is challenging to be used in real-world SDN systems because the SDN systems allow a great degree of flexibility in their systems. For example, network operators can freely select any SDN controller among more than 30 alternatives [10], [17]. Also, control applications can be adjusted dynamically—ONOS provides 175 control applications that can be turned on and off at will. Moreover, SBI can be customized, which increases the variety of the SDN systems (§2.1). Simply, there are countless possible SDN system instances.

Each SDN system instance is distinct in terms of its SDN controller, control applications, and SBI, leading to significant differences in control traffic between instances. We replay and evaluate the prediction errors of the previous studies using identical SDN controllers and SBIs as their target systems. At this stage, we have no choice but to run a greater number of control applications than the target systems in the previous studies. This is because, in a real system, certain control applications must be executed to maintain the minimum functions of the SDN systems (called default applications, such as switch liveness verification), in addition to the control applications covered by the prediction models. We observe that the prediction errors reach up to 85%, which is far from the precise prediction (§2.2). This means that the prediction model should be formulated per each SDN system instance.

Yet, applying DM on countless SDN system instances is not feasible at all. The reason is that DM requires the observation and determination of causal relationships, followed by the model formulation on individual SDN systems that involve ruinous human contemplations. Moreover, the DM approach is based only on the observed relationships. Thus, some relationships can be excluded from the model's coverage, which inevitably causes poor accuracy when the real SDN system instances bear such excluded relationships.

We herein present *Elixir*, an automated prediction model formulation framework. *Elixir* produces a prediction model for each SDN system instance. The model predicts control traffic by accommodating the various SDN data plane, such as network topologies and traffic, as its input features. Instead of DM, *Elixir* leverages machine learning (ML) to formulate prediction models through datasets. Therefore, undiscovered relationships of the target system could be reflected by the dataset. Also, *Elixir* formulates the models without causal relationships by updating the models' parameters (e.g., weight and bias) via iterative training. Furthermore, *Elixir* is designed to automate the model formulation without any human contemplations. Thus, *Elixir* is applicable across heterogeneous SDN systems.

Elixir confronts three serious challenges when employing ML: 1) acquiring the dataset, 2) finding a pertinent ML structure, and 3) improving prediction accuracy. First, regarding the dataset, it is widely recognized that the lack of "high quality" datasets is a major difficulty when applying ML to SDN systems [18]. We develop "SDN dataset generator" (SDN-DAG) that self-generates a dataset for a given SDN system instance (§3.2). Consequently, *Elixir* does not require datasets ahead of time.

Next, finding a pertinent ML structure is challenging when applying ML. It is widely recognized that identifying an appropriate ML structure for a specific system is the most laborious and time-consuming step in formulating a prediction model [19], [20]. Specifically, it is difficult to determine a right ML algorithm and its model structure, such as the number of layers in neural networks. Additionally, the optimal ML algorithm and structure for achieving the best accuracy may vary for each SDN system instance. To address this, *Elixir* defines a "model space" and introduces a new search scheme called the "two-phase search" (§3.3). The model space categorizes ML algorithms for control traffic prediction as shallow and deep algorithms based on their structural complexity. The two-phase search then carefully explores the shallow and deep algorithms, involving linear regression, support vector regression, random forest, LightGBM, XGBoost, deep neural network, and convolution neural network and up to 65 candidate models. We design the two-phase search to be time-efficient for training models.

Finally, to improve prediction accuracy, *Elixir* introduces a unique criterion for final model selection from candidate models—"robustness." Robustness refers to the extent to which an ML model is accurate even when the distributions of input features differ from those of the training dataset. Generally, ML models show high accuracy when the input features given for the prediction have a similar distribution to the training dataset. If not, the accuracy falls highly than the one expected from the training [21], [22]. For example, we find that the prediction error can increase 23× on average when the input features' distributions (i.e., network topologies of data plane) differ. To make our model robust, *Elixir* augments the training dataset with different distributions of input features and scores the candidate models using the augmented dataset. Lastly, a prediction model, called supreme model, is selected (§3.4).

We evaluate *Elixir* with nine different SDN systems. The contributions of this study are summarized as follows:

- Design and implement the *Elixir* framework that is applicable on heterogeneous SDN systems.
- Overcome the challenge of applying ML to SDN systems—automated prediction model formulation by self-generating datasets without any prior datasets.
- Devise a new model search scheme that finds a supreme model with best-possible prediction accuracy.
- Improve the prediction accuracy and prediction robustness up to 10.6× and 80.44%, respectively.

The rest of this paper is organized as follows. §2 explains the background, motivation, and goal of this study. §3 shows the design of the *Elixir* framework. §4 presents the implementation and the evaluation results. §5 discusses future research directions. Finally, §6 concludes this paper.

2 Background, Motivation, and Goal

In this section, we present background of this study including SDN systems and control traffic characteristics (§2.1). Then, we discuss previous related studies, their accuracy problems, and the goal of this study (§2.2).

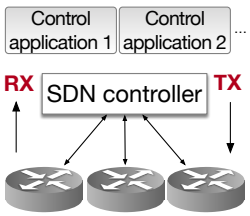


Fig. 1: SDN system example.

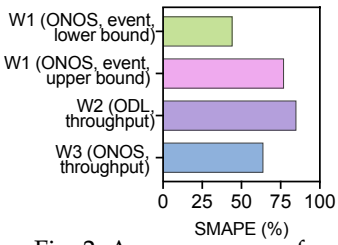


Fig. 2: Average errors of previous studies.

2.1 Background

Heterogeneity of SDN systems. Fig. 1 illustrates an example of an SDN system comprising an SDN controller and switches. We denote the traffic flowing from the SDN controller to a switch as TX, and the other traffic flowing from a switch to an SDN controller as RX. Also, messages consisting of the control traffic are referred to as "events" because they are regarded as events that trigger network operations. For example, upon receiving an event from a switch, such as a new flow rule request, the SDN controller runs the corresponding control applications (e.g., path calculation). Moreover, an event from the SDN controller causes a network switch to perform specific operations, such as flow rule installation at its flow table or generating a statistics reply to the SDN controller.

The heterogeneity of SDN system instances comes from the following reasons. First, there are more than 30 types of SDN controllers [17]. Second, each SDN has its own unique variety of control applications, such as flow rule installation, firewall, network monitoring, and proxy ARP. Additionally, numerous types of SBIs exist as pre-defined protocols, such as OF (from version 1.0 to 1.5), ForCES, POF, NETCONF, and OPFlex [23]. Moreover, network operators can define the SBI arbitrarily (e.g., P4 [24]). In summary, there are numerous options for the SDN controllers, control applications, and SBIs that constitute an SDN system instance.

Necessity of prediction. It is well-known that network planning, such as SDN controller placement, determination of SDN controller parameters (e.g., buffers), and selection of SBI, affects most performance aspects of SDN systems like reliability, scalability, energy efficiency, fault management and quality-of-service [11]. For example, Zhu et al. [10] reported that when handling identical network governing tasks (e.g., flow rule installation), an SDN switch can either be idle (20% CPU utilization) or bottlenecked (90% CPU utilization) depending on the SDN system instance.

Network planning is impossible without accurate control traffic prediction. For example, SDN controller placement takes control traffic throughput as its input for network planning because the control traffic determines the amount of tasks that SDN controllers perform [11], [25]. Similarly, the paper [26] proposed a model to decide the minimum buffer size of an SDN controller for storing control traffic. To determine a buffer size that does not create a bottleneck in an SDN system, the model considers the number of events in control traffic and calculates their arrival rates. In addition to the example mentioned above, various other studies demonstrated the importance of control traffic in designing reliable SDN systems [7], [9], [17], [27], [28], [29].

2.2 Related Work and Goal

Related work. Previous studies [12], [13], [14], [30] have presented various prediction models. Table 1 summarizes the formulation methods, target SDN systems, and output features of the previous studies. As the models are formulated using DM, they constrain the coverage of the target SDN system instances. For example, in [12] and [13], the authors presented models for the ONOS and ODL controllers. Also, the two models covered control applications conducting flow rule installations (i.e., *fwd* and *ifwd* [12] and simple forwarding [13]) with OF 1.0 SBI. Yu et al. [14] built three prediction models for ONOS, POX, and Floodlight (FL), with each model covering a few control applications and OF 1.0 SBI. In another study [30], a prediction model for a special type of control traffic, namely the traffic between ONOS controllers running as a distributed architecture, was formulated. As output features, existing studies focused on average control traffic or its lower and upper bounds. Also, previous studies predicted aggregated values without clearly distinguishing between TX and RX control traffic.

Prediction accuracy of related work. We evaluate the prediction models of previous studies [12], [13], [14] in Table 1. The purpose of the previous studies is to present models for describing a specific SDN system, so most studies lack accuracy evaluations. We test their prediction accuracies on the real values to see the problems in heterogeneous SDN systems. We denote the previous prediction models as "W1" [12], "W2" [13], and "W3" [14]. For W1, we test the prediction accuracy of the lower and upper bounds for the average number of control traffic events of ONOS. Also, for W2 and W3, the average amount of throughput of ODL and ONOS are evaluated, respectively.

Previous studies introduced mathematical equations to predict the features in their SDN system instances using the number of hosts and switches. By using their equations, we obtain the prediction values of the previous studies. Also, we prepare the same target systems as those used in the previous studies to measure real values. We use a server with an Intel Xeon E5-2650 CPU of 20 cores and 64 GB memory, running Ubuntu 16.04. We run an instance of the SDN system as follows. For W1, we run ONOS controller version 2.5 as a container. Then, we run the *fwd* control application with OF 1.0 SBI. We run the same controller for W3, with control applications like *fwd* and monitoring. For W2, we run ODL version 15.1.0 as a container, along with the l2switch control application. As for the physical networks (data plane), we emulate network switches through Open vSwitch and Mininet, which is a software-based emulation. We change the network topology to linear, complete binary tree, 4-ary, 6-ary, 8-ary, 2-tier, and 3-tier as required. We also vary the number of hosts (1–320), traffic connections (1–100), and intervals between traffic connections (1–10 s).

The accuracy is calculated as the symmetric mean absolute percentage error (SMAPE) that gives equal penalty on the positive and negative error between the prediction and real values [31]. SMAPE is defined as Equation 1, where Y_i represents the real value and $\hat{Y}_i$ is the prediction value. Fig. 2 shows the results with the SMAPE ranging from 44.3% (W1, ONOS, event, lower bound) to 85% (W2, ODL, throughput). The errors approaching 85% indicate that the previous

TABLE 1: Related work comparison.

Related work	Formulation method	Target SDN system			Output features (prediction target)
		SDN controller	Control application	SBI	
A. Bianco et al. [12]	DM	ONOS	*fwd* and *ifwd*	OF 1.0	Bounds of events and throughput (aggregated)
A. Bianco et al. [13]		ODL	Simple Forwarding (l2switch)	OF 1.0	Average events and throughput (aggregated)
B. y. Yu et al. [14]		ONOS, POX, FL	A few control applications (e.g., routing, monitoring)	OF 1.0	Average throughput (aggregated)
A. S. Muqaddas et al. [30]		ONOS	None	None	Average throughput (aggregated)
Elixir	**ML**	**Any SDN controllers**	**Any control applications**	**Any**	**Average and maximum events and throughput per TX and RX**

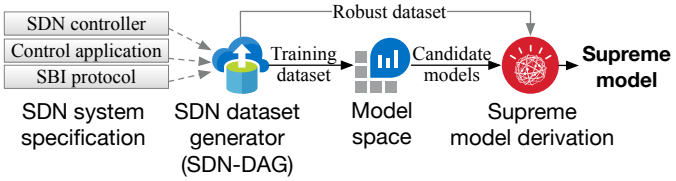


Fig. 3: *Elixir* workflow.

studies are not applicable to the in-practice systems. Due to the poor control traffic prediction, network planning is likewise inaccurate. Therefore, ensuring the reliability of SDN systems is not feasible.

$$SMAPE = \frac{100}{n} \times \sum_{i=1}^{n} \frac{|Y_i| - |\hat{Y}_i|}{(|Y_i| + |\hat{Y}_i|)/2} \quad (1)$$

We find that the poor predictions of previous studies result from the coverage in control applications. Previous studies made their own assumption in which control applications run in the target SDN system. However, real SDN system instances run additional control applications that are not part of the assumption. For example, when using ONOS controller, a minimum of seven control applications is required to run on the SDN system instance (e.g., SBI parser for OF and topology discovery). However, previous studies only assume one or two of control applications because the studies are based on DM of mathematical equations, which find it challenging to capture the various control applications. As a result, the predicted values significantly differ from the actual values. Note that the poor accuracy and coverage of DM have also been reported in other areas (e.g., middleware) [32].

Approach of this study. Thus, the prediction model should be formulated uniquely for each SDN system instance. However, DM requires heavy contemplations that cannot be automated (§1). *Elixir* uses machine learning that generates prediction model based on dataset. As the prediction model is trained using this dataset, the control applications captured in the dataset can be predicted by the model. So *Elixir* can automatically formulate prediction models for any control application. Thus, this study overcomes the limitations of previous studies that only covered two to three control applications. In our experiments, we use *Elixir* to automatically generate ML models for nine representative different SDN system instances (§4.2), each encompassing up to 13 control applications.

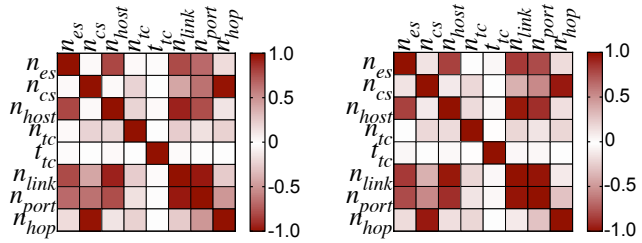
3 *Elixir* Design

This section introduces the *Elixir* framework. The *Elixir* workflow is illustrated in Fig. 3. *Elixir* starts with the specifications of an SDN system instance for which it formulates a prediction model. Note that *Elixir* is designed to be a framework that predicts any SDN system instances by employing multiple ML models. It is pretty well-known that obtaining the dataset is critical for ML training, and to the best of our knowledge, there exists no training dataset for SDN systems. So, the first step of *Elixir* is for the SDN-DAG to produce datasets for the SDN system instance. Based on the dataset from SDN-DAG, the model space searches the various ML algorithms and produces candidate models. Afterward, the supreme model derivation selects the optimal model from the candidate models that meets both the prediction accuracy and robustness.

To explain the SDN-DAG, the input and output features of the prediction models are described in §3.1. Then, the SDN-DAG (§3.2), model space (§3.3), and supreme model derivation (§3.4) are presented.

TABLE 2: Input feature candidates—definition and notation.

Topology features		Traffic features	
Number of edge switches	n_{es}	Number of switch ports	n_{port}
Number of core switches	n_{cs}	Number of traffic connections	n_{tc}
Number of hosts	n_{host}	Network monitoring interval	t_{mon}
Average number of switch hops of traffic connections	n_{hop}	Packet size of events	l_{mes}
Average traffic connection interval	t_{tc}	Packet size for topology discovery	l_{topo}
Number of network links	n_{link}		



(a) ONOS, *fwd*, OF 1.3 (b) ODL, *l2switch*, OF 1.3

Fig. 4: Correlation between input features (Pearson R).

3.1 SDN System Specification (Features of Models)

SDN system specifications include the SDN controller, SBI, and control applications. Then, we define the features that constitute the dataset. Specifically, to create accurate and robust prediction models for each instance, input features are defined to capture the control traffic characteristics of the SDN system. After input features, output features are defined for the control traffic metrics. Finally, based on the specification, *Elixir* generates a training dataset for the prediction model formulation by SDN-DAG (§3.2).

Input features. We review all the features that have been used in DM in previous studies [12], [13], [14]. Table 2 classifies the 11 features from the previous studies into two categories: 1) network topology-related (topology features) and 2) network traffic-related features (traffic features). We

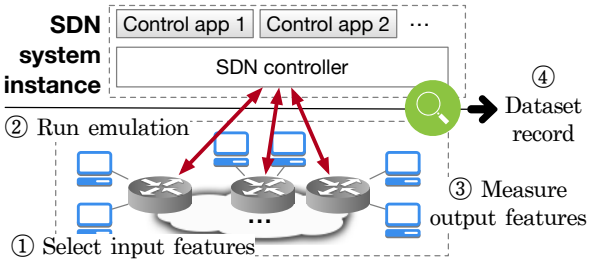


Fig. 5: SDN-DAG structure.

observe that several features listed in Table 2 remain unchanged within an SDN system instance, implying that they are not useful as input features. For instance, once the SBI is fixed, the control traffic length per network control (l_{mes}) remains constant. Also, the network monitoring interval (t_{mon}) and packet size for topology discovery (l_{topo}) are fixed for an SDN controller and control application, making them constant. Accordingly, we exclude these three features, and consider the remaining eight for the input features.

Next, we perform a correlation analysis on the features to identify those that are distinct from the others. The results are depicted in Fig. 4. The number of switch ports (n_{port}) and network links (n_{link}) are significantly close (0.97 Pearson R score). This tendency is also observed in other SDN system instances that we have tested. As input features, we select n_{link} over n_{port} because n_{link} contains the connection information between the switches and hosts, whereas n_{port} only contains the number of existing ports in the switches, regardless of whether they are connected to the switches and hosts. Consequently, the input features are: n_{es}, n_{cs}, n_{host}, n_{tc}, t_{tc}, n_{link}, and n_{hop}. For a single SDN system instance, the input features reflect the characteristics of network topology and traffic.

Output features. Because the control traffic is composed of TX and RX traffic (Fig. 1), the number of events and throughputs for TX and RX are considered as output features. In addition, the average amount of control traffic (per second) from all switches is vital because the SDN controller should be able to process the control traffic on top of the data plane traffic. So, the average control traffic (avg) per second is considered as an output feature.

Additionally, the maximum amount of control traffic (max) is considered as an output feature so that the network planning can prepare for the burst control traffic throughput and events. In summary, the eight output features are avg TX event, avg TX throughput, max TX event, max TX throughput, avg RX event, avg RX throughput, max RX event, and max RX throughput.

3.2 Dataset Generator (SDN-DAG)

Fig. 5 depicts the dataset generator of *Elixir*, SDN-DAG, that generates datasets required to train models for a given SDN system instance. SDN-DAG generates two types of datasets (i.e., training dataset and robust dataset). To generate a record of the training dataset, SDN-DAG first selects the input features to establish a data plane (physical network) that will be controlled by an SDN system instance (①, Fig. 5). SDN-DAG builds the data plane with the selected topology features and runs network emulation with the traffic features (②). Then, the output features are measured by SDN-DAG (③). A dataset record is obtained by combining the selected input features and the measured output (④). The process of Fig. 5 is repeated until the desired number of records are created.

TABLE 3: Input feature range of the SDN-DAG.

Value	Range	Value	Range	Value	Range	Value	Range
n_{es}	1–32	t_{tc}	0–10	n_{cs}	1–64	n_{link}	0–382
n_{host}	1–320	n_{hop}	1–64	n_{tc}	1–100		

Generation of input features. The input features are either topology or traffic features. SDN-DAG selects the topology features (defined in §3.1) first because they are topology-dependent. Some topology features are fixed according to the topology. In 2-tier and 3-tier tree topologies, for example, the numbers of switches and network links are fixed [33]. Furthermore, the network topology influences the interdependence of the input features. In a linear topology, the number of network links is decided based on the number of hosts and switches because the links connect the hosts and switches in a linear fashion. Thus, the SDN-DAG decides on a network topology to select topology features.

As for topologies, we use linear and tree topologies.[1] The linear topology, the simplest form of network topology, clearly depicts the control traffic consumption with the input features. The tree topology has more complicated network configurations than the linear topology has but is more widely utilized for datacenter topologies, such as the fat-tree topology. For the training dataset that is used to train candidate models using the model space (§3.3), SDN-DAG uses linear and complete binary tree topologies. This is because these topologies operate well on most SDN system instances that we have tested and empirically show fewer errors in the output feature results, while the seven input features vary considerably. For the robust dataset that is used to select one supreme model (§3.4) from among candidate models, we augment data records generated from other tree topologies (i.e., 2-tier and 3-tier) that reflect real-world scenarios.

We have also considered the mesh topology because it is one of the most complex topologies that connects all the network switches. However, we ruled out the mesh topology for the following reason. The mesh topology comprises circular paths whose starting and ending points become identical; however, many existing SDN controllers are incapable of dealing with network routing loops [34]. Addressing this issue requires a new implementation of control applications on each SDN controller; so, we do not use the mesh topology in this study.

Once the network topology is decided, the topology features are fixed depending on the topology or other interdependent input features. After determining the topology features, the traffic features, such as the number of traffic connections (n_{tc}) or traffic connection intervals (t_{tc}), are determined. Table 3 presents the value ranges of the input features. Note that when the input features are not balanced, the predictions of the trained model are biased [35]. Therefore, we design the SDN-DAG to select the input features with an even distribution of the features' ranges.

1. SDN-DAG can create records from other topologies, such as fat-tree and ISP topologies to evaluate prediction robustness (§3.4).

Generation of output features. All switches are emulated by Open vSwitch, which is a widely used software switch in real-world network systems that supports a variety of SBIs. We use Open vSwitch for both core and edge switches, as it provides the required functionalities for both. SDN-DAG then starts new TCP connections on the switches. By receiving packets from the new connections, the switches generate control traffic to the SDN controller, and in turn, the SDN controller also sends control traffic to switches, such as for routing. SDN-DAG measures the control traffic and calculates eight output features (e.g., avg TX event). In this study, we run SDN-DAG on a server of an Intel Xeon E5-2650 CPU (20 cores) and 64 GB of memory.

Data records are generated with the input features and the corresponding output features. They are used to train prediction models in the model space. We report the time that SDN-DAG takes to generate datasets in §5.

Consideration of control application characteristics. We define an SDN system instance as a combination of the SDN controller, SBI, and control applications. Our dataset is created per SDN system instance so that the dataset includes control traffic of a specific set of control applications. So, the characteristics of the control applications are captured in the dataset. For example, suppose that we generate a dataset for FL controller running "*forwarding*" control application with OF 1.0 SBI. SDN-DAG creates control traffic and measures output features by running them. Each data record consists of 15 numerical values: seven input features (n_{es}, n_{cs}, n_{host}, n_{tc}, t_{tc}, n_{link}, and n_{hop}) and eight output features (avg TX event, avg TX throughput, max TX event, max TX throughput, avg RX event, avg RX throughput, max RX event, and max RX throughput).

Control applications can behave differently depending on their triggers or thresholds. For example, network monitoring that triggers statistics requests every 1 s generates a larger amount of control traffic than those triggered at 5 s intervals. Prediction models from previous studies can model the control traffic, but they are limited in terms of fixed values of thresholds. The reasons are: they 1) view the SDN system as a stochastic system, 2) manually observe and identify the causal relationship between the control traffic and the elements of the SDN system instances, such as network topology, control application, and thresholds, and 3) translate the relationship into the model. Especially, these previous studies manually analyze and observe the causal relationships from the control application implementations (codes) and not measure the realistic workloads as datasets.

Given that control applications typically include a number of thresholds (e.g., 13 per control application in ONOS), it is challenging for a single DM to reflect multiple thresholds all together. So, because previous studies rely on fixed thresholds, they are limited in the prediction power for the control traffic with changing thresholds. On the other hand, *Elixir* utilizes machine learning to formulate prediction models that only needs datasets and requires significantly less human effort than DM of previous studies. So, when the threshold is changed, its effect is reflected in the dataset. So *Elixir* can build prediction models without analyzing the changed causal relationship manually. We release the datasets generated by SDN-DAG at GitHub repository [36].

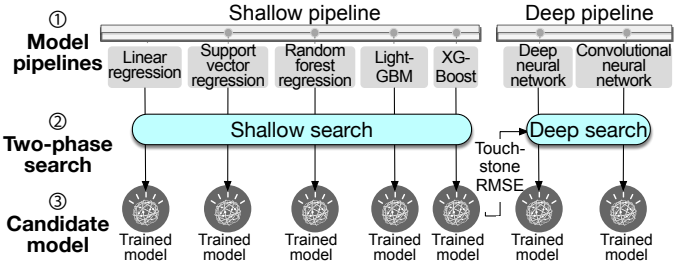


Fig. 6: Model space.

3.3 Model Space

There are many ML algorithms, and the answer to the question of which algorithm works best depends on the target system (use-cases). Thus, identifying an appropriate algorithm for a target system is typically done on a time-consuming process and trial-and-error basis because grid search is commonly used. This challenge has also been reported in previous studies. For example, one previous study [19] reported that three expert data scientists spent several weeks working full-time to find the proper ML algorithm for a public transport maintenance system. Another study [20] reported that 84% of ML practitioners expended significant effort and energy in finding and selecting the appropriate algorithm for their data.

As an alternative to the time-consuming trial-and-error efforts, *Elixir* introduces the notion of "model space." The model space includes collections of ML algorithms (①, Fig. 6). The model space generates prediction models for each ML algorithm in pipelines (to be explained in the next paragraph). We also devise "two-phase search," a new search strategy that improves time and resource efficiency in training ML models in the model space (②, described subsequently). The models identified through the two-phase search are referred to as "candidate models" (③).

The model space has two model pipelines: shallow pipeline and deep pipeline. The shallow pipeline includes classic ML algorithms such as linear regression. Also, the shallow pipeline has ensemble algorithms that connect multiple models of classic ML algorithms as a single model (e.g., XGBoost). We design the shallow pipeline with linear regression (LR), random forest regression (RF), support vector regression (SVR), LightGBM (LG), and XGBoost (XG). These algorithms are also known to be effective for system performance prediction (e.g., network QoS, link outage, GPU utilization, and CPU utilization [37], [38], [39], [40]). Model space sequentially produces candidate models by the ML algorithms in the shallow pipeline. In particular, ML algorithms of the shallow pipeline generate one candidate model for each because the algorithms do not have a sophisticated model structure and have only a few hyperparameters [37], [38], [39], [40]. Therefore, the shallow pipeline results in five candidate models.

The second model pipeline, i.e., deep pipeline, comprises deep learning algorithms. Such algorithms provide a variety of model structure options. For instance, deep neural networks (DNNs) include diverse hyperparameters, such as the number of fully connected (fc) layers, model optimizer, and initializer. So, deep learning algorithms can generate multitudinous model structures. For the deep pipeline, we include DNNs and convolutional neural networks (CNNs)

TABLE 4: Notations and descriptions.

Notation	Description	Notation	Description
d_t	Training dataset	d_r	Robust dataset
d_t^i	i-th data record of d_t	d_r^i	i-th data record of d_r
$d_t^i.input$, $d_t^i.output$	Input of the d_t^i, output of the d_t^i	M	Set of models from both shallow and deep pipelines
M^i	i-th model in M	$p(M^i, d_r^j)$	Predicted value from M^i for j-th element of d_r

because they are nonlinear yet flexible ML algorithms that can handle noisy data and robust predictions [41].

The structure of CNN consists of 1) input nodes, 2) convolution layers, 3) hidden layers, and 4) output nodes. Input nodes take input features. Convolution layers focus on subsets of the input features (through filters) to detect important aspects. The results of the convolution layers are passed to hidden layers that perform calculations using weights and biases. Output nodes predict final eight output features on control traffic.

Elixir takes seven input features. CNN requires three-dimensional data as its input (like images), so we organize our input features into 7x1x1 matrix. This matrix is entered into the CNN structure through the input nodes. We have also tested other matrix forms, such as 1x7x1, but the prediction accuracies show no differences.

The ML models from deep learning algorithms are multiple. Among them, the model space can choose one model showing the best accuracy, such as RMSE, as a candidate model. However, we consider all the multiple trained models from the deep pipeline as the candidate models to consider the prediction robustness, which requires a scoring (prediction) with another dataset (to be explained in §3.4).

In general, training deep pipeline algorithms consumes a large amount of computing resources and time efforts, although they tend to achieve high prediction accuracy. The two-phase search aims to reduce such efforts by splitting the training of the model space. The two-phase search first trains algorithms of the shallow pipeline (shallow search) and then 2) trains algorithms of the deep pipeline (deep search) with the results from the shallow search.

The two phases are explained in depth as follows. For simplicity, we utilize the notations listed in Table 4. In the shallow search, the prediction models from the shallow pipeline are generated. For the training, we use a training dataset (d_t) from SDN-DAG, which contains 2K records having the input features of linear and complete-binary tree topologies (§3.2). d_t is divided into 80% and 20% for model training and validation, respectively. Specifically, the former 80% is used for model training iterations (forward and backward propagations); the latter part 20% is used to determine whether the model's parameters have converged by calculating the prediction accuracy—average root mean squared error (RMSE)—for the eight output features. When the calculated RMSE does not change significantly, the shallow search is complete. Then, we have five candidate models from the shallow ML pipeline (one each from LR, RF, SVR, LG, and XG).

The deep search then trains deep learning models in the deep pipeline. The deep search requires navigations of possible models of deep learning algorithms to find the proper model structure. For example, Table 5 lists the hyperparameters of DNN and CNN (e.g., the loss function [42], activation function [43], initializer [44], and optimizer [45]).

TABLE 5: Hyperparameters (deep search).

Hyperparameter	Category	Hyperparameter	Category
Loss function	MSE, MAE, MSLE, MAPE	Activation function	Elu, relu, linear
Initializer	he_normal, he_uniform	Optimizer	adam, rmsprop
Number of hidden layers	1–10	Number of nodes per hidden layer	2–700
Stride size*	1–6	Number of convolution layer*	1–4
Number of filters*	1–100	Filter size*	2–7
Activation function of convolution layer*	elu, relu		

* Hyperparameter for CNN. Others for both DNN and CNN.

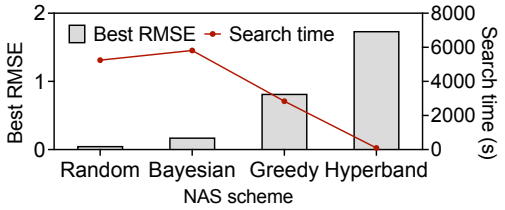


Fig. 7: Comparison of NAS schemes.

Different hyperparameter combinations result in different model structures; thus, the fundamental challenge in using the deep search is to discover proper model structures [46].

Generally speaking, the strategy of determining a suitable model structure in deep learning is known as network architecture search (NAS). We conduct the following experiments to design NAS in deep search. We test four representative schemes: 1) hyperband, 2) greedy search, 3) Bayesian optimization, and 4) random search. Hyperband initially trains all possible model structures for only a few epochs, selects the best one, and then trains it further until it converges. Greedy search finds the best model structure layer-by-layer at each trial, it identifies the optimal hyperparameter for each model layer without considering the impact on other layers. Bayesian optimization uses past trials data within a probabilistic model and prioritizes promising hyperparameters from past trials. Random search selects hyperparameters randomly for each trial.

We compare search time and prediction error (RMSE) of the four schemes. The experiments run 60 trials for search schemes to find a suitable model structure for the DNN algorithm. We use a training dataset created for an SDN system instance of ODL running l2switch with default control applications and OF 1.0 SBI. We measure the best RMSE for "avg TX event" and search time for the 60 trials.

Fig. 7 presents the best RMSE (bars, left y-axis) and search time (line with dots, right y-axis) for NAS schemes (x-axis). Among the four schemes, random search and Bayesian optimization show the best RMSE under 0.2. Greedy search and hyperband show much higher RMSE values (up to 32× higher than random search). So, random search and Bayesian optimization are candidates for control traffic prediction.

Next, we further consider the search time between random search and Bayesian optimization. The search time of random search is 10% shorter than Bayesian optimization. The experimental results reveal that random search exhibits better efficiency in terms of time and accuracy. Thus, we design the deep search based on the random search.

In addition, we set an early termination condition of

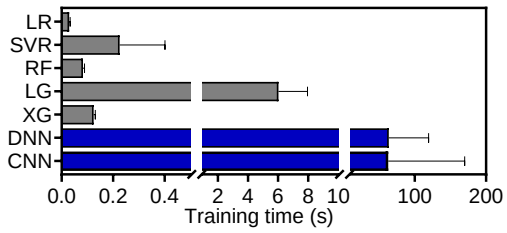
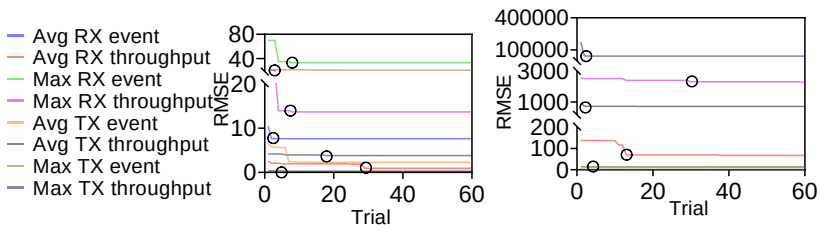


Fig. 8: Training time of each model.



(a) DNN / ONOS, OF 1.0 (b) CNN / ONOS, P4

Fig. 9: RMSE over trials.

NAS to improve time and resource efficiency. The early termination of the NAS is judged based on "touchstone RMSE" that is the lowest average RMSE (best value) among the five candidate models from the shallow search. This is the "knowledge" learned from the shallow search because the shallow pipeline takes relatively less time for training, and it becomes the baseline for the deep pipeline.

Specifically, the deep search identifies candidate models in the deep pipeline as follows. First, a set of hyperparameters (Table 5) is randomly selected. Then, the deep search performs a trial—a deep pipeline model is generated and trained using 80% of d_t[2]. The RMSE for the trained model is calculated at the end of each trial using 20% of d_t. The deep search repeats the above search process until one of the following criteria is met: 1) the RMSE of the trained model is better than the touchstone RMSE, or 2) the number of trials exceeds the threshold (n). The models trained before the search is stopped become the candidate models, which will be scored for the prediction robustness (§3.4).

The rationale for the criteria as termination conditions is as follows. The first criterion, stopping based on the touchstone RMSE, is derived from the following observation. We measure the training time of the candidate models in the shallow pipeline and deep pipeline. The results are shown in Fig. 8. Training all five shallow pipeline models takes 6.38 s on average, whereas training only a single deep pipeline model (one trial) takes 63.07 s on average, which is 9.7× longer. Because the deep search necessitates repeated trials of the DNN and CNN models, the training time increases significantly. So, we establish the first criterion to bound the training time of deep pipeline models.

Second, to identify n, the maximum number of random search trials, we conduct a series of experiments to measure the RMSE after training the deep pipeline model. The purpose of the experiments is to evaluate whether the number of trials is related to the accuracy of the trained models. Fig. 9a shows the DNN model training for ONOS with OF 1.0 SBI, and Fig. 9b shows the CNN model training for the same controller but a different SBI, P4. The x-axis represents the number of trials, and the y-axis represents the best (minimum) RMSE of the DNN models trained during each trial. The circled part of each line corresponds to the trial where the RMSE finally converges to the lowest value. For example, at the number of trials 30, Avg RX throughput in Fig. 9a and Max RX throughput in Fig. 9b show the RMSE converges at its lowest values. In other words, the prediction models ultimately converge to the best accuracy by 30 trials. We also observe that the other SDN system instances exhibit similar tendencies. However, note that the massive differences in RMSE exist because the values to predict (control traffic generated in SDN system instances) in Fig. 9a and Fig. 9b are quite different. For instance, the SDN system instance in Fig. 9b (ONOS, P4) consumes 10,938× more control traffic than the one (ONOS, OF 1.0) in Fig. 9a on average. Since the values to predict vary greatly, the range of RMSE also differs (10,573× on average).

In other words, the model space after the two-phase search can contain from seven to 65 candidate models—five from the shallow search and up to 60 from the deep search (from one to 30 models per DNN and CNN).

3.4 Supreme Model Derivation

The final stage of *Elixir* is "supreme model derivation," which selects supreme models from the candidate models (M=$\{M^i | i = 1, 2, ..., n\}$ where n candidate models exist and M^i is the i-th candidate model). It is known that the ML models show high accuracy on input features that have similar distributions to the ones of training dataset (e.g., dependency between topology features, §3.1). However, the prediction accuracy drops when the input features have different distributions from those of the training dataset [21], [22]. For example, we pick a prediction model having the best RMSE value over the d_t among the M, and conduct predictions with the model on the new dataset. The new dataset includes records for 2-tier (Fig. 10c) and 3-tier (Fig. 10d) topologies. Their input features form different distributions from linear (Fig. 10a) or complete binary tree (Fig. 10b) topologies. Compared to the prediction accuracy on the d_t, the prediction accuracy on the new dataset drops by up to 150.7×. This problem has also been reported in other areas such as image classification [21].

It is impossible to reflect all the possible network topologies in accordance with input feature distributions in d_t. Instead, we incorporate "robustness," a prediction power over input features of different distributions (network topologies), into the selection of supreme models. Supreme model derivation begins with M that has completed the training process. For each M^i, we conduct predictions against a newly created dataset, called "robust dataset" (d_r, to be explained later). Then, we calculate the scores on the robustness based on the prediction results. We devise two kinds of robustness scores (criteria) called RMSE robustness (robust/RMSE) and frequent robustness (robust/freq), to be explained later. We pick two candidate models, called "contenders," which show the best scores (one by RMSE robustness and one by frequent robustness). We explain the robustness scores (i.e., RMSE robustness and frequent

2. We use the identical dataset for training both shallow and deep pipeline algorithms because they have the same input features. The prediction robustness on a different dataset is considered in the supreme model derivation (§3.4).

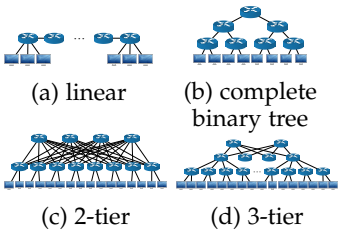


Fig. 10: Network topologies for d_t and d_r.

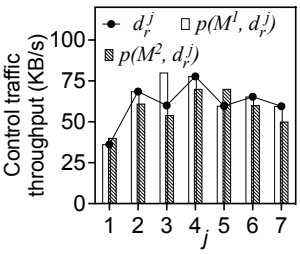


Fig. 11: Need for the robust/freq scheme.

robustness) and the selection of the final supreme model in the subsequent paragraphs, one by one.

RMSE robustness (robust/RMSE). To calculate robustness scores, *Elixir* augments d_t into d_r. Specifically, half of d_r comes from d_t. The remaining half of d_r is augmented with the new data records—created with the 2-tier (Fig. 10c) and 3-tier (Fig. 10d) topologies that have more link connections between switches than linear (Fig. 10a) and complete binary tree (Fig. 10b) topologies. d_r consists of records with different input distributions with d_t, so the prediction against d_r shows the prediction robustness of M^i.

Based on the augmented d_r, *Elixir* performs predictions for the output features for each M^i. The score, robust/RMSE of M^i (denoted as s^i_{rm}), is then calculated as the average RMSE between the predicted and measured (real) values. We select a candidate model with the minimum s^i_{rm} as a contender based on its robust/RMSE.

Frequent robustness (robust/freq). The second score is robust/freq. The necessity of the robust/freq score comes from the fact that the robust/RMSE alone may be insufficient for evaluating robustness. We present an example case in Fig. 11. The x-axis presents data records of d_r (with index j), and the y-axis presents the avg RX throughput of each data record. Dots in the solid line are the output features (measured values), and the two bars are the predicted values M^1 and M^2. M^1 makes good predictions for the data records except for j=3. In other words, M^1 has a single and significant error at j=3; the calculated s^1_{rm} and s^2_{rm} values are 7.6 and 7.5, respectively, so the robust/RMSE criterion selects M^2 as a contender. However, M^1 shows fewer errors (RMSEs) than M^2 for 86% of the data records (i.e., j=1, 2, 4, 5, 6, 7). Thus, the robust/RMSE criterion does not catch this case when used alone as a criterion.

For this purpose, we develop robust/freq score. The robust/freq score of M^i (s^i_{fr}) is calculated as Equation 2. Equation 2 counts the number of times that M^i showed the best RMSE for each record in d_r. So, s^i_{fr} means that the number of d_r records on which M^i shows the highest prediction accuracy. After calculating the s^i_{fr} values, *Elixir* selects a candidate model with maximum s^i_{fr} as a contender.

$$F(a,b) = \begin{cases} 1, & \text{if } a = b \\ 0, & \text{otherwise} \end{cases},$$
$$s^i_{fr} = \sum_j F(\arg\min_i \sqrt{(p(M^i, d_r^j) - d_r^j.output)^2}, i) \quad (2)$$

Final supreme model. *Elixir* finally selects a supreme model from robust/RMSE and robust/freq. The final selection takes into consideration which of the criteria is more lucrative for evaluating robustness. The contender from robust/freq is advantageous only when it has an exclusively higher s^i_{fr} than the other models, implying that the robust prediction frequency is concentrated in the contender. We quantify the "exclusiveness in robust prediction frequency" by dividing s^i_{fr} of the contender from robust/freq by the number of records in d_r. We select the contender from robust/freq as the supreme model only when the calculated exclusiveness of frequency exceeds a threshold (k). If k is under the threshold, the contender from robust/RMSE is selected. We empirically set k to 44%, which yields the best prediction accuracy in our system (for nine SDN system instances). The determination of k on real-world scenarios is discussed in §5.

TABLE 6: SDN system instances for evaluation.

Name	Controller	SBI	Representative control applications
FL-1.0-forwarding	FL	OF 1.0	Forwarding
FL-1.3-forwarding		OF 1.3	
ODL-1.0-l2switch	ODL	OF 1.0	Layer 2 switch (l2switch), LLDP, monitoring
ODL-1.3-l2switch		OF 1.3	
ONOS-1.0-fwd	ONOS	OF 1.0	Routing (reactive forwarding)
ONOS-1.3-fwd		OF 1.3	
ONOS-1.3-mon		OF 1.3	Routing, monitoring (1 s)
ONOS-1.3-arp		OF 1.3	Routing, proxy ARP
ONOS-P4-routing		P4	Routing, Stratum, NG-SDN tutorial, monitoring (30 s)

4 Implementation and Evaluation

In this section, we first explain the implementation details of the *Elixir* framework in §4.1. The following sections then present evaluations, including evaluation settings (§4.2), automated model formulation (§4.3), and prediction performance (§4.4). The major findings are as follows:

- *Elixir* outperforms previous studies up to 10.6× by its ML-based model formulation (§4.3).
- Through model space and supreme model derivation, *Elixir* improves prediction robustness up to 80.4% (§4.4).
- Out of 72 cases, *Elixir* employs both shallow and deep pipeline algorithms as supreme models (24% and 76%, respectively), showing the reasonability of considering the algorithms (§4.4).

4.1 Implementation

Owing to space limitations, we explain only the major pieces of the *Elixir* implementation. The control plane of an SDN system instance is containerized and used to produce output parameters in the SDN-DAG. For the control plane, only 30, 24, and 18 LoCs are required for the containerization of the ONOS, ODL, and FL controllers [47], [48], [49]. For the data plane, network switches are emulated using Open vSwitch, and hosts are created as separate containers. The model space and supreme model derivation of *Elixir* are implemented through TensorFlow 2.2, Scikit-Learn, and Keras libraries. Through the *Elixir* implementation, the training of prediction models of an SDN system instance takes an average of 138.8 minutes (ranging from 101.4 to 183.7 minutes) using an NVIDIA RTX 2080Ti GPU. We release the implementation of *Elixir* framework on GitHub [36].

4.2 Evaluation Settings

Elixir evaluation is conducted using nine different SDN system instances (Table 6). The names in Table 6 refer to

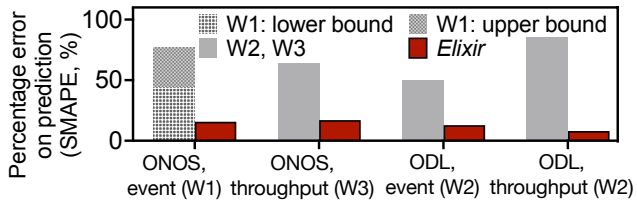


Fig. 12: Prediction errors comparison.

"controller-SBI-control applications." FL, ONOS, and ODL controllers are selected because they are proven for practical use (including more than 100 deployments such as AT&T and T-Mobile [50]). For SBI, the numbers "1.0" and "1.3" refer to OF 1.0 and OF 1.3, respectively. As FL, ODL, and ONOS support OF, we test OF as SBI. For the control applications, we put the name of the major control application running with the default applications of the SDN system instance name. For example, "forwarding" of FL, "l2switch" of FL, and "fwd" of ONOS are the packet forwarding control applications. Also, "arp" and "mon" of ONOS indicate proxy ARP processing from ONOS and network monitoring, respectively.

We include the *l2switch* (ODL) and *fwd* (ONOS) with OF 1.0 because they have been modeled by DM. Then, we vary the SDN system instances by adding 1) another SBI (e.g., OF 1.3), 2) additional SDN controller (e.g., FL), and 3) other control applications such as monitoring and proxy ARP. The monitoring interval of the ONOS is set as 1 s because 1 s is the minimum amount that we can set without controller implementation changes. Moreover, we include the P4 SBI for the evaluation case. The P4 SBI has no specified protocol on syntaxing the control traffic. The available operations of the switch and control traffic can be customized, which increases the modeling difficulty.

Next, to run ML algorithms, we use one NVIDIA RTX 2080Ti GPU of 11 GB GPU memory. We use the following software libraries for ML on this machine: CUDA (version 10.2), TensorFlow (version 2.2), Scikit-Learn (version 0.19.1), and Keras (version 2.11.0). The host side runs Ubuntu 18.04 on Intel Xeon Silver 4210 CPUs and 128 GB memory.

The evaluation consists of the following subsections:

- **Automated Model formulation:** Whether the *Elixir* can accurately formulate prediction models than previous studies (§4.3)
- **Prediction robustness:** Whether the supreme model of the *Elixir* achieves better prediction accuracy in terms of robustness (§4.4)

All predictions are conducted using the evaluation dataset (d_e). To measure the prediction robustness, the augmented records of d_r (i.e., 2-tier and 3-tier topologies) make up the d_e. Also, we add records of new topologies, such as 4-ary, 6-ary, 8-ary, 8-host ISP, and 16-host ISP, to d_e. Shortly, d_e includes records of topologies not used for model training (model space) or supreme model derivation.

4.3 Automated Model Formulation

Here, we evaluate the automated model formulation ability by comparing the prediction accuracy of models built by *Elixir* with the DM models proposed in previous studies [12], [13], [14]. The prediction accuracy is calculated by SMAPE similar to §2.2. We use the notations W1, W2 and W3 for the previous studies (defined in §2.2). For the corresponding SDN system instances to W1, W2, and W3, *Elixir* formulates the prediction models (supreme model). Fig. 12 presents the prediction errors. The average SMAPEs for the DM models and *Elixir* are 64.18% and 13.41%, respectively; thus, SMAPE is improved 4.78× on average, ranging from 2.8× (ONOS, event) to 10.6× (ODL, throughput).

Fig. 13 depicts a microscopic view of Fig. 12, as its x-axis is a set of random selections of two data records of each network topology in d_e. The y-axis of Fig. 13 is the measured output and predicted output. Fig. 13a shows the number of events in ONOS that W1 and *Elixir* predict. The prediction from W1 has upper and lower bounds that are depicted by black lines. Also, Fig. 13b is the amount of throughput in ONOS predicted by W3 and *Elixir*. Figs. 13c and 13d are the result of W2 and *Elixir* for the throughput and event for ODL, respectively.

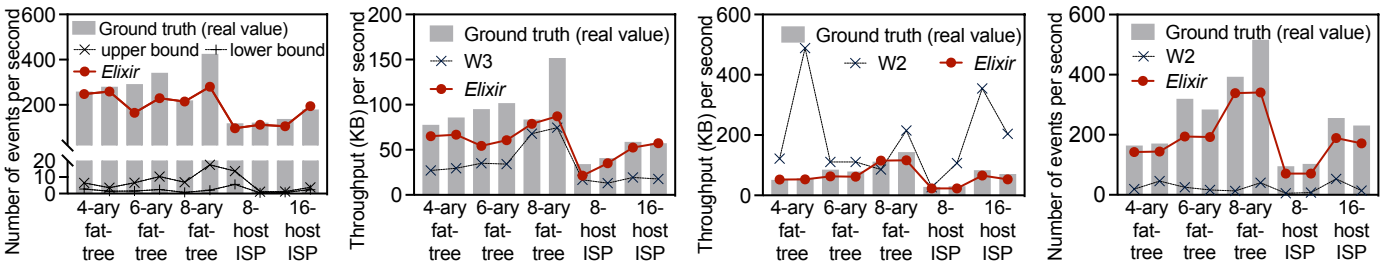
In the results, previous studies (line with × or + marks in Fig. 13) mostly exceed or fall short of the actual measured values (gray-colored bars), exhibiting large errors. In contrast, *Elixir* (red line with circle marks) shows results that are relatively close to the actual values. Specifically, in Fig. 13a, the lower and upper bounds of W1 show 9.2× and 8.8× higher errors on average than *Elixir*, each. Also, in Fig. 13b, W3 shows 3.04× higher error than *Elixir*. In Figs. 13c and 13d, W2 show 3.71× and 5.46× higher errors in predicting events and throughput of ODL than *Elixir*, respectively.

Compared to DM models (previous studies), *Elixir* improves prediction accuracy for the following reasons. The DM models are built on the limited number of control applications of an SDN system instance. For example, W1 and W2 cover only one control application running on ONOS and ODL (i.e., *fwd* OF 1.0 and *l2switch* with OF 1.0). W3 covers more control applications on ONOS (two deep le types) but less than the default applications running on ONOS (seven types). In contrast, *Elixir* generates prediction models based on the datasets that reflect all control applications running on the SDN system instance. Thus, the above results show that the prediction models of *Elixir* are generalizable to control applications.

Moreover, to our knowledge, maximum control traffic (i.e., max TX event, max TX throughput, max RX event, and max RX throughput) has not been predicted in the literature. This is because DM requires a causal relationship for maximum traffic; however, such a relationship is much more challenging to grasp due to its complexity—for example, peak traffic consumption is difficult to model from stochastic mechanisms (e.g., implementation of control applications and the message size of SBI) that humans can perceive. In contrast, using the dataset, *Elixir* reflects relationships that are difficult to catch. Considering that the prediction of maximum control traffic is paramount for bottleneck prevention and resource provisioning, *Elixir* has an obvious benefit.

4.4 Prediction Performance

Here, we investigate the prediction robustness achieved by supreme model selection. The prediction robustness of the supreme models from *Elixir* is compared to that of models that do not undergo the supreme model derivation. The



(a) ONOS, event, W1. (b) ONOS, throughput, W3. (c) ODL, throughput, W2. (d) ODL, event, W2.

Fig. 13: Prediction results obtained for representative data records.

latter models are referred to as "naïve" models. The naïve model is one of the candidate models in the model space, and it has the lowest prediction error (RMSE) on 20% of d_t, not on d_r of supreme model selection. We formulate the supreme models and naïve models for nine SDN system instances (Table 6). Also, for each SDN system instance, eight prediction models per individual output feature are chosen. Thus, 72 supreme and 72 naïve models are obtained.

Prediction accuracy. We compare the prediction accuracy of the supreme and naïve models. For both models, prediction is made against d_e, which contains new records (network topologies) that are not used in either model training or selection. Fig. 14 shows the heatmaps that illustrate prediction accuracy improvement. The x and y axes represent the SDN system instance and output features, each. We compute s^i_{rm} (Fig. 14a) and s^i_{fr} (Fig. 14b), which are used to account for prediction robustness in supreme model selection. Then, for each output feature, we collect the maximum improvement values from s^i_{rm} or s^i_{fr} and present the values in Fig. 14c. Fig. 14c is presented to determine whether at least one of the s^i_{rm} or s^i_{fr} is improved.

In Fig. 14a, we show the improved s^i_{rm}, which is calculated by dividing s^i_{rm} of the supreme models by that of naïve model. A higher percentage signifies greater improvement, whereas 0% indicates no improvement. The darker the cell of Fig. 14a, the larger the improvement. Among the 72 models compared, 72% (52 models) exhibit improved prediction accuracy. The improved models indicate an average of 12.1%, including a maximum improvement of 58.1% (ODL-1.0-l2switch, avg RX throughput). The average prediction accuracy for all 72 models increases by 7.6%.

Fig. 14b shows the increased s^i_{fr} of supreme models. We calculate the increased s^i_{fr} by subtracting s^i_{fr} of the naïve models from that of the supreme model. A larger value indicates a greater improvement of accurate prediction frequency. From the results, 43 supreme models (59.7%) exhibit better s^i_{fr}, with an average improvement of 12.6% and a maximum improvement of 51.9% (ONOS-1.3-arp, Avg RX throughput). On average, all 72 models improve by 6.2%.

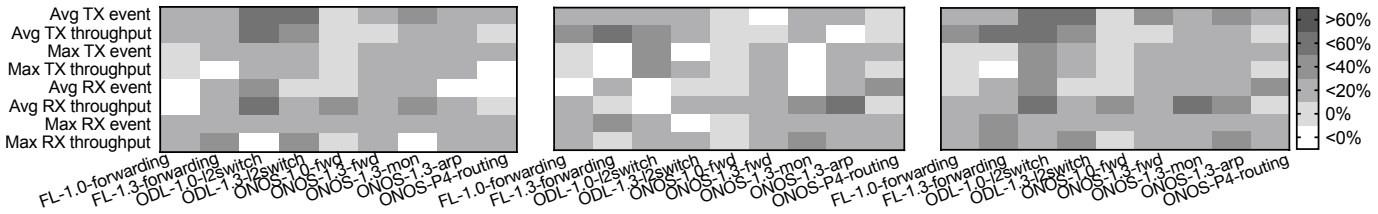
In Fig. 14c, we check whether at least one of the two metrics has improved. From a total of 72 models, 57 (79.2%) are improved, 13 have similar errors (18.1%), and two (only 2.7%) have greater errors. For the models with improved performance, either one of the two metrics improves by an average of 16.5%. For the entire models, the metrics improve 13% on average. Considering that the majority of models improve either one of the two metrics, we believe that the supreme model selection scheme is reasonable.

Prediction robustness. We validate the prediction robustness of *Elixir*. The prediction robustness is to reduce the amount of increased error when the dataset changes. We measure the prediction robustness by computing the reduction in prediction errors on changing the dataset. Specifically, we perform the following. We begin by conducting predictions on d_t (used for model training) and d_e (used for robustness evaluation) through the naïve and supreme models in Fig. 14. For each naïve and supreme model, the RMSE of d_e inference is divided by d_t inference, which we denote as "robustness error." Then, we get robustness errors of naïve and supreme models.

Fig. 15a shows the robustness errors of naïve and supreme models. For each SDN system instance, eight models for eight output features exist. We average the robustness error of the eight models (SDN system instance. In Fig. 15a, the errors of supreme models (lines with circles) are normalized to the value of naïve models (lines with $\times$ marks). All nine SDN system instances show lower robustness errors on supreme models. On average, supreme models show 38.1% lower robustness errors than naïve models.

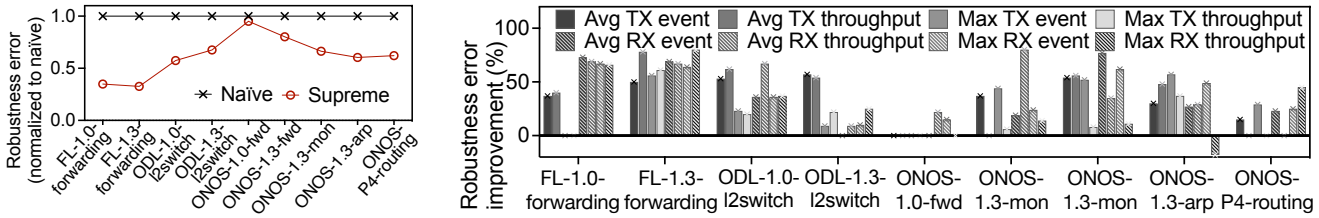
Next, we show the robustness error improvement for 72 individual models. We calculate the improvement by dividing the robustness error of the naïve model by that of the supreme model. For instance, 20% improvement means that the supreme model shows 20% lower robustness error than the naïve model. The y-axis of Fig. 15b displays the calculated prediction robustness according to each SDN system instance (x-axis). Among the 72 models, 58 models (80.6%) reduce the robustness error by an average of 41.9%. The maximum improvement is 80.44% of avg RX throughput of ONOS-1.3-fwd. The other models exhibit similar or poor robustness errors, but the amount of increase is only 1.3%. The evaluation results reveal that the design for prediction robustness in *Elixir* is considerably effective in formulating prediction models for dynamic SDN system instances.

ML algorithms. Here, we explain the distributions of the ML algorithms selected as supreme models. 24% of the models trained in Fig. 14 are from shallow algorithms, and the remaining 76% are from deep algorithms. Specifically, XG is the most frequently used among shallow algorithms (for 7%). For deep algorithms, DNN and CNN are used 53% and 23%, respectively. In general, shallow algorithms, such as RF, have fewer parameters in their models, so the algorithms are prone to be overfitted to d_t. The results of supreme models show that the model selected based on robustness includes numerous deep algorithms to enhance the prediction robustness. Meanwhile, shallow algorithms are successful in SDN systems with little complexity in predicting control traffic. This is because the relatively fewer



(a) s^i_{rm} comparison. (b) s^i_{fr} comparison. (c) aggregated comparison.

Fig. 14: Robust prediction error improvement.



(a) robustness errors. (b) individual model improvement.

Fig. 15: Improvement of robust prediction.

parameters and simple structures required allow the model to be concentrically trained on the core causal relationships of the low-complexity system. It is worth noting that supreme models use both shallow (24%) and deep algorithms (76%), demonstrating that *Elixir* properly considers the characteristics of ML algorithms and the complexity of the SDN system instance.

5 Discussion

SDN-DAG in practice. We discuss the data generation time and the difference between the emulated data from the SDN-DAG and the real-world data. First, for the data generation, creating 2K data records with eight parallel SDN-DAG instances takes 10 to 12 hours, which could decrease with a greater number of instances. Second, for the emulated data, our implementation covers the major functions of in-practice SDN as SDN primarily aims to softwarize each network component. Thus, we believe that ML models based on this are similar to real-world situations.

Final supreme model threshold (k). As our work focuses on building domain-specific solutions, we set the value k empirically, resulting in the best prediction accuracy in SDN systems. Specifically, we prepare realistic SDN systems as follows. For the control plane, we test various real-world SDN controllers (e.g., ONOS, FL, and ODL) and run their control applications. For the data plane, we execute various physical network topologies used in real-world scenarios, such as linear, two-tier tree, three-tier tree, and fat-tree topology. The topology consists of OpenFlow switches created by Open vSwitch and Mininet, the most widely-used framework for evaluating SDN systems as production networks for research and validation [51]. Note that these configurations are the ones used in previous works to model real-world control traffic as well [12], [13], [14]. We find the value 44% in the above setting, so we believe that the value is applicable in real-world scenarios because real-world SDN controllers and control applications are used.

However, we recognize that 44% may not be optimal in every case. We investigate how prediction accuracy (error) changes as the k value varies (sensitivity). When the k value is set to 80% (1.8× higher than the optimal value) for nine different SDN system instances used in our experiments, the average prediction errors differ by 7.2% from that with 44%. When the k value is set at 10% (0.2× the optimal value), the average prediction errors differ by 16.9%. Compared to the changes in k values (0.2× to 1.8×), the variances in prediction errors are relatively small (7.2% to 16.9%).

Time complexity of model space. We compare the time complexity of our approach to that of grid search. Grid search is 1) to train various models on every possible combination of model structures and hyperparameters and 2) to select the model with the highest accuracy as the appropriate model. Suppose n_i is the number of possible values of the i-th hyperparameter or model structure. The time complexity of grid search is $O(\prod_i n_i)$. On the other hand, our model space is designed based on random search that restricts the number of maximum training trials. We meticulously design our own criteria for search termination, ensuring that the number of search trials is bounded at a certain integer (i.e., 30, details in §3.3). The time complexity of *Elixir* is $O(1)$, which is highly efficient. However, *Elixir* searches models with accuracy similar to grid search.

Model search trial and prediction accuracy. In our experiments, there are cases where the number of trained models in the model space is less than 65. For example, for the SDN system instance ONOS-1.3-arp, the total number of trained candidate models is 19 when the touchstone RMSE is met, comprising five from the shallow pipeline models and 14 from the deep pipeline models (four DNNs and 10 CNNs). In this case, even though the number of trained models is below the upper bound of 65, its robust prediction improvement is 1.96× better than the other SDN system instances that search for models up to the upper bound (e.g., ONOS-1.3-forwarding, FL-1.0-forwarding) on average. We experiment how the best RMSE changes over the number of trials. The results are in Fig. 9, and it exhibits that for all cases, the best RMSE occurs before 30 trials. So we believe that experiments with 65 trained models are sufficient to ensure the accuracy of *Elixir*.

Other NAS methods. One might wonder why *Elixir* does not utilize the existing NAS methods. Due to the following reasons, *Elixir* has to devise its own method. First,

existing NAS methods (e.g., Google Model Search [52]) find models predicting discrete values (e.g., dog or cat). So, they mostly focus on supporting classification problems. However, since the control channel performance is continuous, they have to be tailored for predicting "continuous" values (e.g., regression). Second, the NAS methods are primarily based on the ensemble that assembles prediction models from pre-defined ML algorithms [53]. However, the control channel needs prediction on continuous values, and we find that more algorithms (than NAS provides) need to be explored to obtain reasonable accuracy (§4.4).

6 Conclusion

We present *Elixir*, a framework to build control traffic prediction models by employing ML. Through SDN-DAG, the dataset for training is generated. *Elixir* introduces the model space that includes prediction models made from several ML algorithms, and the models are searched and trained. Finally, the supreme model is derived among the prediction models. Advantages of *Elixir* include the following: 1) it does not require any collection of datasets, 2) *Elixir* automates the prediction model formulation on heterogeneous SDN systems, and 3) the supreme models achieve prediction robustness over different network topologies. Through comprehensive evaluations, the prediction errors are improved up to 10.6× compared with existing studies.

As a framework, *Elixir* has two characteristics: first, it can build prediction models for any SDN system instance even when the controller or SBI is changed. This means that as SDN controllers are newly released every quarter and SBI protocols continuously evolve, *Elixir* can be viable to provide the adaptation to such changes in SDN system instances. Second, *Elixir* is fully automated to generate the dataset and perform training without manual intervention. Thus, *Elixir* can help design (particularly large-scale) SDN systems with the prediction of their control traffic.

References

[1] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving, G. Desai, B. Felderman, P. Germano, A. Kanagala, J. Provost, J. Simmons, E. Tanda, J. Wanderer, U. Hölzle, S. Stuart, and A. Vahdat, "Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*. ACM, 2015, p. 183–197.

[2] C.-Y. Hong, S. Mandal, M. Al-Fares, M. Zhu, R. Alimi, K. N. B., C. Bhagat, S. Jain, J. Kaimal, S. Liang, K. Mendelev, S. Padgett, F. Rabe, S. Ray, M. Tewari, M. Tierney, M. Zahn, J. Zolla, J. Ong, and A. Vahdat, "B4 and after: Managing hierarchy, partitioning, and asymmetry for availability and scale in google's software-defined wan," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. ACM, 2018, p. 74–87.

[3] A. D. Ferguson, S. Gribble, C.-Y. Hong, C. Killian, W. Mohsin, H. Muehe, J. Ong, L. Poutievski, A. Singh, L. Vicisano, R. Alimi, S. S. Chen, M. Conley, S. Mandal, K. Nagaraj, K. N. Bollineni, A. Sabaa, S. Zhang, M. Zhu, and A. Vahdat, "Orion: Google's Software-Defined networking control plane," in *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. USENIX, Apr. 2021, pp. 83–98.

[4] "SD-Fabric," (Accessed on 11/15/2022). [Online]. Available: https://opennetworking.org/sd-fabric/

[5] M.-P. Odini and A. Manzalini, "SDN in NFV architectural framework," *IEEE Software Defined Networks Newsletter*, 2016.

[6] "Open network operating system (ONOS) SDN controller for SDN/NFV solutions," (Accessed on 12/29/2021). [Online]. Available: https://opennetworking.org/onos/

[7] M. Karakus and A. Durresi, "A survey: Control plane scalability issues and approaches in software-defined networking (SDN)," *Computer Networks*, vol. 112, pp. 279 – 293, 2017.

[8] D. Kreutz, F. M. Ramos, and P. Verissimo, "Towards secure and dependable software-defined networks," in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*. ACM, 2013, p. 55–60.

[9] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: Scaling flow management for high-performance networks," in *Proceedings of the ACM SIGCOMM 2011 Conference*. ACM, 2011, p. 254–265.

[10] L. Zhu, M. M. Karim, K. Sharif, C. Xu, F. Li, X. Du, and M. Guizani, "SDN controllers: A comprehensive analysis and performance evaluation study," *ACM Comput. Surv.*, vol. 53, no. 6, dec 2020.

[11] T. Das, V. Sridharan, and M. Gurusamy, "A survey on controller placement in sdn," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 1, pp. 472–503, 2020.

[12] A. Bianco, P. Giaccone, R. Mashayekhi, M. Ullio, and V. Vercellone, "Scalability of ONOS reactive forwarding applications in ISP networks," *Computer Communications*, vol. 102, p. 130–138, 2017.

[13] A. Bianco, P. Giaccone, A. Mahmood, M. Ullio, and V. Vercellone, "Evaluating the SDN control traffic in large ISP networks," in *2015 IEEE International Conference on Communications*, 2015, pp. 5248–5253.

[14] B. Yu, G. Yang, and C. Yoo, "Comprehensive prediction models of control traffic for SDN controllers," in *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, 2018, pp. 262–266.

[15] P. C. Austin, F. E. Harrell Jr, and E. W. Steyerberg, "Predictive performance of machine and statistical learning methods: Impact of data-generating processes on external validity in the "large n, small p" setting," *Statistical methods in medical research*, vol. 30, no. 6, pp. 1465–1483, 2021.

[16] A.-L. Boulesteix and M. Schmid, "Machine learning versus statistical modeling," *Biometrical Journal*, vol. 56, no. 4, pp. 588–593, 2014.

[17] S. Ahmad and A. H. Mir, "Scalability, consistency, reliability and security in SDN controllers: A survey of diverse SDN controllers," *Journal of Network and Systems Management*, vol. 29, no. 1, pp. 1–59, 2021.

[18] J. Xie, F. R. Yu, T. Huang, R. Xie, J. Liu, C. Wang, and Y. Liu, "A survey of machine learning techniques applied to software defined networking (SDN): Research issues and challenges," *IEEE Communications Surveys Tutorials*, vol. 21, no. 1, pp. 393–430, 2019.

[19] L. Tuggener, M. Amirian, K. Rombach, S. Lörwald, A. Varlet, C. Westermann, and T. Stadelmann, "Automated machine learning in practice: state of the art and recent results," in *2019 6th Swiss Conference on Data Science (SDS)*. IEEE, 2019, pp. 31–36.

[20] M. A. Munson, "A study on the importance of and time spent on different modeling steps," *ACM SIGKDD Explorations Newsletter*, vol. 13, no. 2, pp. 65–71, 2012.

[21] M. Cauchois, S. Gupta, A. Ali, and J. C. Duchi, "Robust validation: Confident predictions even when distributions shift," *arXiv preprint arXiv:2008.04267*, 2020.

[22] M. Cauchois, S. Gupta, and J. Duchi, "Knowing what you know: valid and validated confidence sets in multiclass and multilabel prediction," *arXiv preprint arXiv:2004.10181*, 2020.

[23] Z. Latif, K. Sharif, F. Li, M. M. Karim, S. Biswas, and Y. Wang, "A comprehensive survey of interface protocols for software defined networks," *Journal of Network and Computer Applications*, vol. 156, p. 102563, 2020.

[24] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, p. 87–95, jul 2014.

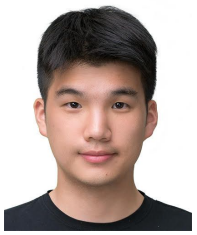
[25] Q. Qin, K. Poularakis, G. Iosifidis, and L. Tassiulas, "SDN controller placement at the edge: Optimizing delay and overheads," in *IEEE Conference on Computer Communications*, 2018, pp. 684–692.

[26] S. Azodolmolky, P. Wieder, and R. Yahyapour, "Performance evaluation of a scalable software-defined networking deployment," in *2013 Second European Workshop on Software Defined Networks*, 2013, pp. 68–74.

[27] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On controller performance in software-defined networks," in *2nd USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*. USENIX, Apr. 2012.

[28] M. Karakus and A. Durresi, "A scalability metric for control planes in software defined networks (SDNs)," in *2016 IEEE 30th*

International Conference on Advanced Information Networking and Applications (AINA), 2016, pp. 282–289.
[29] G. Yang, Y. Yoo, M. Kang, H. Jin, and C. Yoo, "Accurate and efficient monitoring for virtualized sdn in clouds," *IEEE Transactions on Cloud Computing*, vol. 11, no. 1, pp. 229–246, 2023.
[30] A. S. Muqaddas, A. Bianco, P. Giaccone, and G. Maier, "Inter-controller traffic in ONOS clusters for SDN networks," in *2016 IEEE International Conference on Communications*, 2016, pp. 1–6.
[31] R. J. Hyndman and A. B. Koehler, "Another look at measures of forecast accuracy," *International Journal of Forecasting*, vol. 22, no. 4, pp. 679–688, 2006.
[32] D. Didona, F. Quaglia, P. Romano, and E. Torre, "Enhancing performance prediction robustness by combining analytical modeling and machine learning," in *6th ACM/SPEC international conference on performance engineering*, 2015, pp. 145–156.
[33] F. Yao, J. Wu, G. Venkataramani, and S. Subramaniam, "A comparative analysis of data center network architectures," in *International Conference on Communications*. IEEE, 2014, pp. 3106–3111.
[34] A. Basta, A. Blenk, S. Dudycz, A. Ludwig, and S. Schmid, "Efficient loop-free rerouting of multiple SDN flows," *IEEE/ACM Transactions on Networking*, vol. 26, no. 2, pp. 948–961, 2018.
[35] F. Thabtah, S. Hammoud, F. Kamalov, and A. Gonsalves, "Data imbalance in classification: Experimental evaluation," *Information Sciences*, vol. 513, pp. 429 – 441, 2020.
[36] Y. Yoo, G. Yang, C. Shin, J. Lee, and C. Yoo, "Elixir: machine learning-based prediction model formulation framework for control traffic in SDN systems." [Online]. Available: https://github.com/yeonhooy/Elixir
[37] M. Mongelli, T. De Cola, M. Cello, M. Marchese, and F. Davoli, "Feeder-link outage prediction algorithms for SDN-based high-throughput satellite systems," in *2016 IEEE International Conference on Communications*, 2016, pp. 1–6.
[38] R. Pasquini and R. Stadler, "Learning end-to-end application QoS from OpenFlow switch statistics," in *2017 IEEE Conference on Network Softwarization (NetSoft)*, 2017, pp. 1–9.
[39] C. Sieber, A. Obermair, and W. Kellerer, "Online learning and adaptation of network hypervisor performance models," in *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, 2017, pp. 1204–1212.
[40] G. Yeung, D. Borowiec, A. Friday, R. Harper, and P. Garraghan, "Towards GPU utilization prediction for cloud deep learning," in *12th USENIX Workshop on Hot Topics in Cloud Computing*, Jul. 2020.
[41] B. Eftekhar, K. Mohammad, H. E. Ardebili, M. Ghodsi, and E. Ketabchi, "Comparison of artificial neural network and logistic regression models for prediction of mortality in head trauma based on initial clinical data," *BMC medical informatics and decision making*, vol. 5, no. 1, pp. 1–8, 2005.
[42] P. Christoffersen and K. Jacobs, "The importance of the loss function in option valuation," *Journal of Financial Economics*, vol. 72, no. 2, pp. 291 – 318, 2004.
[43] S. Qian, H. Liu, C. Liu, S. Wu, and H. S. Wong, "Adaptive activation functions in convolutional neural networks," *Neurocomputing*, vol. 272, pp. 204 – 212, 2018.
[44] B. Hanin and D. Rolnick, "How to start training: The effect of initialization and architecture," in *Advances in Neural Information Processing Systems*, vol. 31, 2018, pp. 571–581.
[45] I. Bello, B. Zoph, V. Vasudevan, and Q. V. Le, "Neural optimizer search with reinforcement learning," 2017.
[46] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, p. 281–305, Feb. 2012.
[47] "onosproject/onos Dockerfile - Docker Hub," (Accessed on 01/02/2023). [Online]. Available: https://hub.docker.com/r/onosproject/onos/dockerfile
[48] "glefevre/floodlight Dockerfile - Docker Hub," (Accessed on 12/22/2022). [Online]. Available: https://hub.docker.com/r/glefevre/floodlight/dockerfile
[49] "docker-opendaylight/Dockerfile," (Accessed on 01/02/2023). [Online]. Available: https://github.com/sfuhrm/docker-opendaylight/blob/master/Dockerfile
[50] J. H. Cox, J. Chung, S. Donovan, J. Ivey, R. J. Clark, G. Riley, and H. L. Owen, "Advancing software-defined networks: A survey," *IEEE Access*, vol. 5, pp. 25 487–25 526, 2017.
[51] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *9th ACM SIGCOMM Workshop on Hot Topics in Networks*. ACM, 2010.
[52] "google/model_search," (Accessed on 12/15/2022). [Online]. Available: https://github.com/google/model_search
[53] "Vertex AI | google cloud," (Accessed on 12/15/2022). [Online]. Available: https://cloud.google.com/vertex-ai/docs/tabular-data/overview#classification_and_regression_with



Yeonho Yoo [GSM] received his B.S. degree in computer science from Kookmin University, Seoul, Republic of Korea, in 2017, and his M.S. degree in computer science from Korea University, Seoul, Republic of Korea in 2021. He is currently pursuing his Ph.D. degree with Korea University. He worked as a research intern at Microsoft Research Asia in 2023. His current research interests include network virtualization, SDN, datacenter systems, and AI systems.

Gyeongsik Yang [M] received his B.S., M.S., and Ph.D. degrees in computer science from Korea University, Seoul, Republic of Korea, in 2015, 2017, and 2019, respectively. He worked as a research intern at Microsoft Research Asia and as a research professor at Korea University. He is currently an assistant professor in the Department of Computer Science and Engineering at Korea University. His research interests include operating systems, AI systems, datacenter systems, network virtualization, and SDN.

Changyong Shin [GSM] received his B.S. degree in computer science from Korea University, Seoul, Republic of Korea, in 2021. He is currently pursuing his Ph.D. degree with Korea University, Seoul, Republic of Korea. His current research interests include distributed deep learning systems, cloud orchestration, and SDN.

Junseok Lee received his B.S degree in electrical engineering from Kyeonghee University, Yongin, Republic of Korea, in 2022. He is currently pursuing his M.S degree with Korea University, Seoul, Republic of Korea. His current research interests include SDN, datacenter systems, and blockchain.

Chuck Yoo [M] received his B.S. and M.S. degrees in electronic engineering from Seoul National University, and M.S. and Ph.D. degrees in computer science from the University of Michigan, Ann Arbor. He worked as a researcher at Sun Microsystems. Since 1995, he has been at the College of Informatics at Korea University, where he is currently a professor. His research interests include server/network virtualization and operating systems.