A Hybrid Deep Learning Approach for Bottleneck Detection in IoT

ABSTRACT

Cloud computing is perhaps the most enticing innovation in the present figuring situation. It gives an expense-effective arrangement by diminishing the enormous forthright expense of purchasing equipment foundations and processing power. Fog computing is an additional help to cloud infrastructure by utilizing a portion of the less-registered undertaking at the edge devices, reducing the end client's reaction time, such as IoT. However, most of the IoT devices are resource-constrained, and there are many devices that cyber attacks could target. Cyber-attacks such as bottleneck, Dos, DDoS, and botnets are still significant threats in the IoT environment. Botnets are currently the most significant threat on the internet. A set of infected systems connected online and directed by an adversary to carry out malicious actions without authorization or authentication is known as a botnet. A botnet can compromise the system and steal the data. It can also perform attacks, like Phishing, spamming, and more. To overcome the critical issue, we exhibit a novel botnet attack detection approach that could be utilized in fog computing situations to dispense with the attack using the programmable nature of the software-defined network (SDN) environment.We carefully tested the most recent dataset for our proposed technique, standard and extended performance evaluation measures, and current DL models. To further illustrate overall performance, our findings are cross-validated. The proposed method performs better than previous ones in correctly identifying 99.98% of multi-variant sophisticated bot attacks. Additionally, the time of our suggested method is 0.022(ms), indicating good speed efficiency results.

EXISTING SYSTEM

Several researchers are focusing on detecting botnet attacks these days [28]_[30]. The main requirement in botnet detection is identifying the infected devices before they can exploit the network by initiating malicious activity. Authors propose numerous methods that claim to secure the network against botnet attacks. These approaches focus on anomaly detection schemes using artificial intelligence, primarily ML and DL algorithms. In various research approaches, authors [21]_[23] used ML and hybrid ML techniques for botnet detection such as BayesNet (BN), Support Vector Machine (SVM), J48, Decision Tree (DT), and Naive Bayes (NB). Furthermore, Machine Learning methods are categorized as the supervised, the unsupervised, or the semi-supervised learning.

Parakash *et al.* performed experiments using three well-known machine learning algorithms to detect DDoS packets: K-Nearest Neighbors algorithm (KNN), SVM, and NB. The findings show that the KNN performs better in detecting DDoS attacks having 97% accuracy, while SVM and NB algorithms achieve 82% and 83% accuracy, respectively [33]. In [34], the authors proposed a detection scheme that uses the SVM algorithm with their own proposed idle timeout adjustment algorithm (IA). They demonstrated the way their proposed methodology outperforms and achieves better results. In another work, [35] uses, NB, SVM and neural network. Results show that the neural network and NB models performed outclass and achieved 100% accuracy, while the SVM model was at 95% accuracy. Ye *et al.* [36] also used the SVM algorithm and achieved an average accuracy of 95.24%. In [37], authors performed experiments using various algorithms such as Naive Bayesian and decision tree classifier algorithms. They achieved a 99.6% detection accuracy rate.

DL algorithms are the subset of ML. That can deal with large datasets and unstructured data. ML algorithms do not provide better results for extensive data produced by IoT devices and unstructured data [38]. Hence DL algorithms are

preferable for IoT compared to traditional ML algorithms such as KNN, SVM, NB, and others. Different DL and hybrid DL approaches are applied for detecting various kinds of malware in IoT devices [39]_[41]. In [42], the authors described a technique for defending the IoT environment against malware and cyber attacks, such as DDoS, brute force, bot, and infiltration. This strategy makes use of DL in SDN.

Disadvantages

- An existing system is not hybrid deep learning detection policy to improve the efficiency and effectiveness of the SDN-based fog computing architecture. Results show that the proposed scheme works better and provides a better detection rate.
- can't customize the policies and applications dues to its programmable nature.

Proposed System

- The system suggests an efficient deep learning framework for detecting Botnet attacks in an SDN-based fog computing environment.
- The practical experiment is performed on N_BaIoT Dataset, which comprises both Botnet attack and benign samples.
- The proposed technique is evaluated against well-known performance evaluation metrics of the machine and deep learning algorithms known as precision, F1-score, recall, accuracy, and so forth.
- For unbiased results, we also applied the technique of 10-fold-cross-validation.

Advantages

System can manage the secure connection for thousands of devices connected over the fog for data transmission.

System can provide real-time monitoring and awareness with low latency.

System can dynamically balance the load with its flexible architecture.

SYSTEM REQUIREMENTS

- **H/W System Configuration:-**

- Processor - Pentium –IV
- RAM - 4 GB (min)
- Hard Disk - 20 GB
- Key Board - Standard Windows Keyboard
- Mouse - Two or Three Button Mouse
- Monitor - SVGA

SOFTWARE REQUIREMENTS:

- **Operating system** : Windows 7 Ultimate.
- **Coding Language** : Python.
- **Front-End** : Python.
- **Back-End** : Django-ORM
- **Designing** : Html, css, javascript.
- **Data Base** : MySQL (WAMP Server).

IEEE Transaction on Deep Learning,Volume:10,Issue Date:20.July.2022

A Hybrid Deep Learning Approach for Bottleneck Detection in IoT

FRAIDOON SATTARI[1], ASHFAQ HUSSAIN FAROOQI[2], ZAKRIA QADIR[3], BASIT RAZA[1], HADI NAZARI[1], AND MUHANNAD ALMUTIRY[4], (Member, IEEE)

[1]Department of Computer Science, COMSATS University Islamabad, Islamabad 45550, Pakistan
[2]Department of Computer Science, Air University, Islamabad 44000, Pakistan
[3]School of Computing Engineering and Mathematics, Western Sydney University, Penrith, NSW 2751, Australia
[4]Department of Electrical Engineering, Northern Border University, Arar 91431, Saudi Arabia

Corresponding author: Basit Raza (basit.raza@comsats.edu.pk)

The authors extend their appreciation to the Deputyship for Research & Innovation, Ministry of Education in Saudi Arabia for funding this research work through the project number IF_2020_NBU_431.

ABSTRACT Cloud computing is perhaps the most enticing innovation in the present figuring situation. It gives an expense-effective arrangement by diminishing the enormous forthright expense of purchasing equipment foundations and processing power. Fog computing is an additional help to cloud infrastructure by utilizing a portion of the less-registered undertaking at the edge devices, reducing the end client's reaction time, such as IoT. However, most of the IoT devices are resource-constrained, and there are many devices that cyber attacks could target. Cyber-attacks such as bottleneck, Dos, DDoS, and botnets are still significant threats in the IoT environment. Botnets are currently the most significant threat on the internet. A set of infected systems connected online and directed by an adversary to carry out malicious actions without authorization or authentication is known as a botnet. A botnet can compromise the system and steal the data. It can also perform attacks, like Phishing, spamming, and more. To overcome the critical issue, we exhibit a novel botnet attack detection approach that could be utilized in fog computing situations to dispense with the attack using the programmable nature of the software-defined network (SDN) environment. We carefully tested the most recent dataset for our proposed technique, standard and extended performance evaluation measures, and current DL models. To further illustrate overall performance, our findings are cross-validated. The proposed method performs better than previous ones in correctly identifying 99.98% of multi-variant sophisticated bot attacks. Additionally, the time of our suggested method is 0.022(ms), indicating good speed efficiency results.

INDEX TERMS Fog security, software defined networks, deep learning, Internet of Things, botnet, intrusion detection.

I. INTRODUCTION

One of the most significant issues for the network system to be efficient and reliable while doing transactions over the IoT is security [1]. The tremendous growth of IoT in different fields, i.e., surveillance, healthcare, transportation, manufacturing industry, education, and others, encourages securing IoT infrastructure to improve its performance. Earlier IoT devices generate data through various types of sensors, and it becomes tidy for the cloud servers to handle or process these transactions efficiently. Fog computing is among the newly proposed schemes that could be utilized to add preferred features to the IoT infrastructure [2]. Fog computing is competent in doing some regional analysis of information [3] before communicating the aggregated data to the cloud server. It helps in keeping the latency constraints in some time compelled real-time issues, making them appropriate for IoT-based applications such as vehicular ad-hoc networks (VANETs) [4]–[11]. These advancements towards using fog servers in IoT infrastructure motivate the adversaries to target

The associate editor coordinating the review of this manuscript and approving it for publication was Ahmed M. Elmisery.

TABLE 1. List of acronyms in manuscript.

Notations	Explanation	Notations	Explanation
ML	Machine Learning	DL	Deep Learning
ROC	Receiver Operating Characteristic	TNR	True Negative Rate
DoS	Denial of Service	FNR	False Negative Rate
IoT	Internet of Things	RNN	Recurrent Neural Network
CNN	Convolutional Neural Network	FOR	False Omission Rate
DDoS	Distributed Denial of Service	IIoT	Industrial Internet of Things
DNN	Deep Neural Network	FPR	False Positive Rate
SDN	Software Defined Network	FDR	False Discovery Rate
NPV	Negative Predictive Value	MCC	Matthews Correlation Coefficient
LSTM	Long short-term Memory	TS	Threat Score

the fog server with malicious intent to lower its performance. Hence, security and protection of the system are among the major issues that can affect the performance of fog computing [12]. In this regard, availability is among the core security requirements for offering services to the actual customer applications according to their interest. However, this is constantly tested by the adversaries by launching different types of attacks, such as DoS or DDoS attacks [13]. An individual or a group can perform these attacks. If a group performs it, it is named "botnet," while if an individual launches it, it is known as "bot-master." [14]. The bot-master is the attacker node that can launch several types of attacks on the server, such as Phishing, spam, Click fraud, and others. A command-and-control channel remotely controls a botnet. The command-and-control channel is a system the adversary uses to control by sending messages and commands to a compromised system. The adversary can steal the data through these commands and manipulate the infected network [8]. In a botnet attack, some 'n' number of compromised nodes are controlled by a bot-master, and they launch an attack on the server from different compromised systems.

In the fog computing paradigm security is still challenging task, and various security schemes are proposed to make it resilient against vulnerabilities. However, most of the schemes focus on flexibility and continuous monitoring of the fog server. Software-defined networking (SDN) is used at fog servers to address flexibility, and continuous monitoring issues [15]. SDN is an emerging networking paradigm that assists in making the network more flexible that can help in managing the network, analyzing the traffic, and assisting in the routing control architectures [16], [17] as there is a separate control plan that provides a flexible device management policy. Hence, an SDN-based fog computing environment provides centralized control to the fog computing system. The characteristics of the SDN based fog computing system are discussed below:

- SDN can manage the secure connection for thousands of devices connected over the fog for data transmission.
- SDN can provide real-time monitoring and awareness with low latency.
- SDN can dynamically balance the load with its flexible architecture.
- SDN can customize the policies and applications dues to its programmable nature. [18].

The software-defined network plays a vital role as its network control architecture can be directly programmable through the command requests. SDN-based fog computing architecture can assist in analyzing and managing IoT devices. The motivation behind SDN is to give consistency to network management through partitioning the network into the data plane and the control plane. SDN can add programmability, adaptability, and versatility to the fog computing system. In high-speed networks, discovering the botnet attack is a significant concern [19]. The proposed work shows the methodology through which the botnet attack is identified with a high detection rate which can be used in SDN to enhance the security of fog computing. Deep learning (DL) based detection approach in the SDN-based fog computing application can be a better counterattack to improve the overall performance of the system [20]. DL strategy is adaptable to conditions to recognize the abnormal behavior of the network. We proposed a hybrid deep learning detection policy to improve the efficiency and effectiveness of the SDN-based fog computing architecture. Results show that the proposed scheme works better and provides a better detection rate.

A. RESEARCH CONTRIBUTIONS

The research contribution includes the comprehensive evaluation of botnet attacks for different IoT devices and evolving cyber threats in IoT using the dataset N_BaIoT 2018. Our proposed hybrid technique comprises two DL algorithms: DNN and LSTM. We rigorously evaluate the proposed mechanism with standard performance metrics (i.e., Recall, Accuracy, F1-Score, Precision, AU-ROC, etc.). The presented hybrid scheme results show better detection accuracy with low computational complexity. The contributions of this research work are as given below:

- We suggest an efficient deep learning framework for detecting Botnet attacks in an SDN-based fog computing environment.
- The practical experiment is performed on N_BaIoT Dataset, which comprises both Botnet attack and benign samples.

TABLE 2. Summary of existing work.

Ref	Year	Threats	Evaluation Metric	Dataset	Deep Learning	Machine Learning	Achievement
[21]	2017	Botnet Attack	TPR, FPR, TNR	CTU-13	—	Random Forest	93.6% Accuracy
[22]	2021	DNS-Based Botnet Detection	FPR, Precision, F1-Score	CTU-13	—	Hybrid Rule-based Model	99.96% Accuracy and 1.6% false positive rate
[23]	2018	IoT Botnets	Precision, Recall, and F1-Measure	—	—	Logistic Regression Model	97.30% Accuracy
[24]	2022	IoT Botnets	Recall, Precision, Accuracy	N_BaIoT 2018	—	ANN, K-NN, Ensemble tree, Fuzzy classifier	Tree-based algorithm achieved 99% accuracy
[25]	2022	IoT Botnets	Accuracy, F1-Score, Sensitivity	N_BaIoT 2018	—	RF and eXtreme Gradient Boosting (XGB-RF)	99.9426% Accuracy with error score of 0.06%
[34]	2021	IoT Botnets	Accuracy, Recall, F1-Score, Sensitivity	N_BaIoT 2018	CNN-LSTM	—	90.88% Accuracy
[35]	2020	DoS Attacks	Precision, Accuracy	BoT-IoT	Multilayer Perceptron, CNN	RF, SVM	CNN model achieved 91.27% Accuracy
[36]	2021	IoT Botnets	Accuracy, FPR, F-Measure, Recall	IoT-23	CNN-LSTM	—	96% Accuracy
[37]	2021	Intrusion Detection	Recall, F1-Score, and Precision	CICIDS2018	Cu-DNNGRU + Cu-BLSTM	—	Hybrid model achieved FPR of 0.0554% and accuracy of 99.87%
[38]	2018	Botnet DDoS Attacks	—	Self Generated	BLSTM-RNN	—	Hybrid model achieved 99% Accuracy
[39]	2018	DoS,R2L,U2R, probing Attacks	Recall, Specificity, Precision, FPR, F1-Score	KDD99	Restricted Boltzmann Machines (RBM)	—	Achieved precision higher than 94%
[40]	2019	Botnet DoS Attack	Precision, Recall, F1-Score	CTU-13 and ISOT	LSTM and CNN	—	99.3% Accuracy and 99.1% F1-Score

- The proposed technique is evaluated against well-known performance evaluation metrics of the machine and deep learning algorithms known as precision, F1-score, recall, accuracy, and so forth.
- For unbiased results, we also applied the technique of 10-fold-cross-validation.

The paper's organization is as follows; section II introduces related literature. Section III shows the security issues in Fog computing. Section IV provides information about Deep Learning and its algorithms. Section V details our proposed system and the methodology used for detection and experimentation, such as Dataset, detection phase, evaluation phase, and experiment. While Section VI comprises the experimental results and our assessment results. Finally, section VII provides the conclusion and defines the future map.

II. LITERATURE REVIEW

Security remained one of the top research areas in networking paradigms whether it is based on cloud computing [12], fog computing [3], IoT [1] or SCADA (Supervisory Control and Data Acquisition) systems [26], [27] or others. Several researchers are focusing on detecting botnet attacks these days [28]–[30]. The main requirement in botnet detection is identifying the infected devices before they can exploit the network by initiating malicious activity. Authors propose numerous methods that claim to secure the network against botnet attacks. These approaches focus on anomaly detection schemes using artificial intelligence, primarily ML and DL algorithms. In various research approaches, authors [21]–[23] used ML and hybrid ML techniques for botnet detection such as BayesNet (BN), Support Vector Machine (SVM), J48, Decision Tree (DT), and Naive Bayes (NB). Furthermore, Machine Learning methods are categorized as the supervised, the unsupervised, or the semi-supervised learning [24], [25], [31], [32]. Parakash *et al.* performed experiments using three well-known machine learning algorithms to detect DDoS packets: K-Nearest Neighbors algorithm (KNN), SVM, and NB. The findings show that the KNN performs better in detecting DDoS attacks having 97% accuracy, while SVM and NB algorithms achieve 82% and 83% accuracy, respectively [33]. In [34], the authors proposed a detection scheme that uses the SVM algorithm with their own proposed idle timeout adjustment algorithm (IA). They demonstrated the way their proposed methodology outperforms and achieves better results. In another work, [35] uses, NB, SVM and neural network. Results show that the neural network and NB models performed outclass and achieved 100% accuracy,

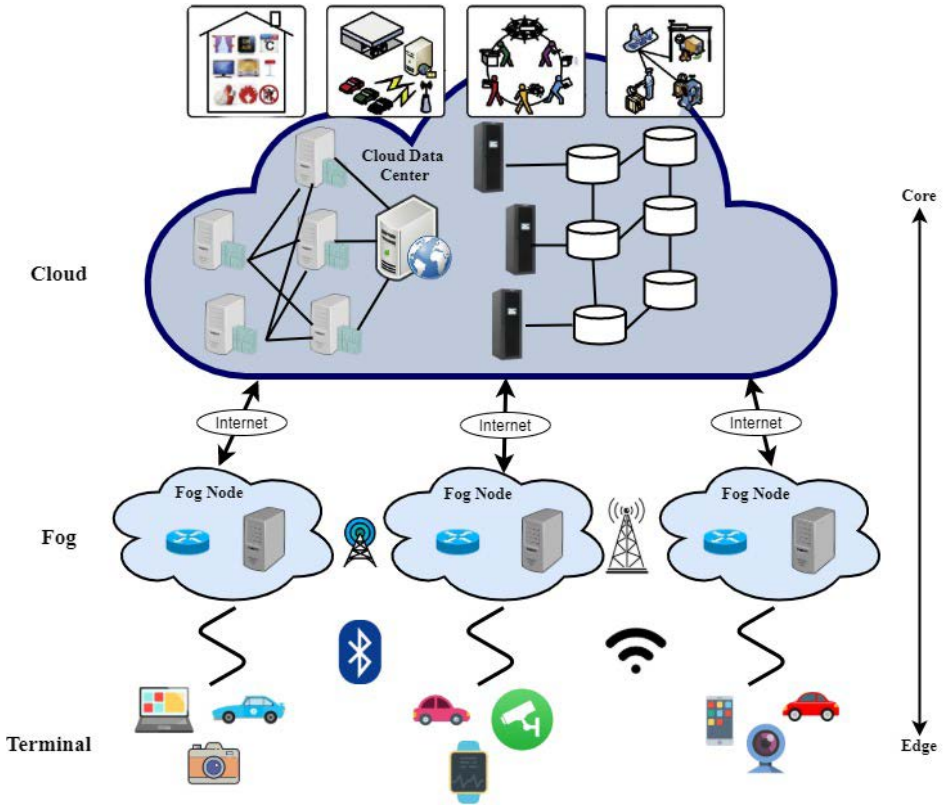


FIGURE 1. Fog computing and communications.

while the SVM model was at 95% accuracy. Ye *et al.* [36] also used the SVM algorithm and achieved an average accuracy of 95.24%. In [37], authors performed experiments using various algorithms such as Naive Bayesian and decision tree classifier algorithms. They achieved a 99.6% detection accuracy rate.

ML algorithms face challenges like scalability, learning from massive data, and low-value density data. To convert big data into usable intelligence in the face of an ever-growing big data universe, ML must develop and improve. Massive data is developing exponentially, so ML must develop and evolve to turn big data into valuable insight.

DL algorithms are the subset of ML. That can deal with large datasets and unstructured data. ML algorithms do not provide better results for extensive data produced by IoT devices and unstructured data [38]. Hence DL algorithms are preferable for IoT compared to traditional ML algorithms such as KNN, SVM, NB, and others. Different DL and hybrid DL approaches are applied for detecting various kinds of malware in IoT devices [39]–[41]. In [42], the authors described a technique for defending the IoT environment against malware and cyber attacks, such as DDoS, brute force, bot, and infiltration. This strategy makes use of DL in SDN. They used the CICIDS2018 dataset for the evaluation of the presented scheme. The proposed model achieved 99.87% accuracy, 0.0554% FPR, with a testing time of only 18.9ms. Likewise, in [43], using two-way LSTM to implement DL for evaluation demonstrates a new method for packet-level inspection on the IoT and networks. The authors utilized Mirai and normal IoT traffic generated in this paper for experiments.

Consequently, the authors of [44] used SDN to deploy an detection mechanism system to safeguard the IoT and showed a testing success rate with 95% accuracy. They considered the KDD99 dataset for attack detection using the Restricted Boltzmann Machine for DoS, login, and Probe (RBM). Moreover, in [45], authors considered a hybrid model consisting of CNN and RNN. The proposed solution is based on network flow attributes. They applied the proposed model to two datasets, i.e., CTU13 and ISOT. These combined datasets form two classes, i.e., botnets and benign. The author of [16] offered an IoT based work that acknowledges the effectiveness of a DL-based algorithm (LSTM) for botnet attack detection. The study used data from various IoT devices from the N IoT 2018 dataset, which had a 99.90% detection rate.

DL approaches are helpful for intrusion detection in SDN-based architectures [20], [46] [47]. DL methods are applied to identify botnet attacks in non-SDN infrastructures [48] while requiring more research to analyze the feasibility and efficacy of using DL (CNN, RNN, and LSTM) algorithms to detect and mitigate botnet attacks on SDN controllers.The studies show that to effectively defend the system against newly developing threats, a centralised mechanism

TABLE 3. Security issues in fog computing.

Attacks	Description
Spam	An unwanted message was developed and distributed by intruders. Spam is one of the most severe security threats since it can allow malware to propagate and waste resources.
DoS	To make the Fog nodes unavailable to real users, flood them with many bogus requests.
Man-in-the-middle	An attack in which an attacker is placed between two connecting parties to intercept and manipulate data passing between them.
Tampering	The attacker alters, delays or drops the data packets to reduce or disrupt the performance and efficiency of Fog computing.
Eavesdropping	Sniffing and spoofing attacks are other names for these types of attacks. Hackers take data from computers, smartphones, and other devices and send it across the internet without the users' permission.
Jamming	A DOS attack where one node blocks other nodes from interacting on the channel by occupying the channel they are communicating.
Forgery	The attackers use bogus information to fool victims by imitating their identities. Because of the bogus data packets, this attack reduces network performance using energy, storage, and bandwidth.
Sybil	An attack where one node steals the identity of another node to take control of and compromise Fog nodes to generate fraudulent sensing data while exposing the users' personal information.

and intelligence are still needed. The accuracy of botnet malware detection varies for different algorithms applied to different datasets.

In Table 2, several ML or DL based detection schemes are presented. It shows that the selected research area is among the emerging research trends in the field of IoT security. There is ongoing research in this area using different datasets [49]. As per our findings, a thorough study of the DL hybrid combinations is required to explore the possibility of increasing the accuracy and precision in the detection of botnet attacks such that it further achieves lower FPR in consuming less time. Hence, we tested various combinations of DL algorithms in our research work and concluded that the hybrid deep learning algorithm uses DNN [50] and LSTM [51] is effective. It also produces better outcomes compared to other strategies that have been suggested. Additionally, it completely pinpoints sophisticated and devastating multi-attacks in the IoT environment.

III. SECURITY ISSUES IN FOG COMPUTING

Edge computing, IoT, and Industry 4.0 have advanced and developed quickly in recent years. Recently, there have been many cloud-based service providers (Cisco, VMware, IBM, Juniper, Big Switch Networks, Versa Networks, Colt Technology, and Lumina SDN) who have shifted from the traditional network paradigm towards Software-Defined Fog (SD-FoG) [52]. The fog server is the fundamental part of the system as it holds critical information related to IoT devices, accumulating and storing the client's data. The basic architecture of the fog paradigm is depicted in Figure 1, representing edge devices connected with fog servers for communication. Numerous distributed fog and edge servers are deployed to offer services over the network to millions of consumers, whereas; the fog servers are connected to cloud servers. The openness and accessibility of the network assets through these devices make fog computing vulnerable. Fog computing has created a new security conundrum due to its significant distribution properties, heterogeneity, mobility, and restricted resources. Because of its limited computing capability, the Fog would struggle to implement a comprehensive security solution to detect and prevent attacks.

Furthermore, because of its position (i.e., close to IoT devices, meaning protection and surveillance are insufficient), the fog will be easier to access and more accessible to hack than the cloud, increasing the likelihood of attacks. Due to its capacity to acquire private information from IoT devices and the cloud, as well as the amount of throughput data, fog will be a target for multiple cyberattacks. Various attacks could be performed to compromise the systems and get important information. These attacks may include SQL injection, Zero-day attack, Man-in-the-middle attack, or others. Botnet attack is a champion among the most lethal attacks. A botnet may be a compromised note in the frameworks linked over the web and are controlled distantly by a criminal to perform malignant action without consent and approval [53]. The essential purpose of criminals to perform Botnet is a cash-related advantage by performing Denial of service assault, Phishing, spamming, and other attacks. SDN is an open, programmable emerging paradigm that permits simple enhancement between the control plane and network devices. SDN energizes network innovation and deploys network capabilities. Research is in progress toward providing security to SDN-based fog computing architectures. There is a need for a proper framework integrated with SDN controllers to monitor fog computing and identify compromised devices. Billions of objects are connected; their administration and control of the enormous number of objects is a challenging task in a circulated system [9]. Hence, monitoring and providing security in fog computing on the internet from cyber-attack, specially Botnet using SDN, is our primary focus. In Table 3, various attacks are illustrated that can be launched against Fog nodes to degrade the performance of the system.

IV. DEEP LEARNING

In this section, we elaborate on the DL algorithms focused on in our proposed scheme. These are as follows:

A. DEEP NEURAL NETWORK (DNN)

A DNN has an input layer, an output layer, and in any case, there is a hidden layer [50]. Each level meets clear organizational types and requirements in the excellence chain

interaction. Managing unlabeled or unstructured data is one of these advanced neural networks' most important applications. The term ''deep learning'' is also used to describe this deep neural network, in which innovation uses part of artificial consciousness. Try to group and query data in ways that go beyond basic information/performance conventions. Utilizing a DL technique has the benefit of being able to automatically identify the crucial features from data without the need for a feature selection procedure. Moreover, DNN methods have proved to be dependable and generalized, if designed well, capable of detecting zero-day threats. Furthermore, DNN computational complexity is defined as follows.

$$m_i = f(\sum_{i=1}^{s} y_i + x_i + v) \tag{1}$$

where the weights for input are y_i, and x_i while the weight for output is m_i. likewise 's' show number of samples, 'v', 'f' is for bias vector and the training function respectively.

B. LONG SHORT TERM MEMORY (LSTM)

LSTM is a neural network used to identify sequence data [51]. The idea of LSTM is to take care of the contribution to the next layer with the output layer of the past layer for better learning of deliberation of information and comprehension. LSTM plays out a comparable assignment for every segment of a course of action, with the output dependent upon past computations. Another way to deal with considering LSTM is that they have a ''memory'' that gets information about what has been resolved up until this point. Also, neurons in LSTM are intended to speak with layers for and significant execution upgrade.

There are three essential gates in the LSTM: a gate for input, a forget gate and a gate for output. The input gate's responsibility is to store the training data in long-term memory. The previous time step is used to initialise the short-term memory, whilst the most recent input data is used to initialise the long-term memory. The training data is separated from the useless information by filters in the input gate, and the valuable data is passed to the sigma function. There are two indicator values for the sigma function: 0 and 1. Fundamental values are represented by a 1, whereas unimportant values are represented by a 0. Long-term memory is used to store the output from the input layer. The forget gate is one of the most significant gates in the LSTM model. Which information should be saved or ignored is determined by multiplying the forget vector values by the current input gate. The subsequent cell receives a new copy from long-term memory when the output from the forget gate has been sent.

$$\begin{aligned} a_t &= \sigma(w_a[n_{t-1}, x_t] + b_a), \\ i_t &= \sigma(w_i[n_{t-1}, x_t] + b_i), \\ \tilde{C}_t &= \tanh(w_c[n_{t-1}, x_t] + b_c), \\ C_t &= a_t * C_{t-1} + i_t * \tilde{C}_t, \\ O_t &= \sigma(w_o[n_{t-1}, x_t] + b_o), \end{aligned}$$

$$n_t = o_t * \tanh(C_t) \tag{2}$$

For the input layer, i_t is the output values, W represents weight values, and b is used for the base. The crucial information is transferred to the following cell using the σ activation function. The forget gate's output is a_t, the output gate is O_t, the cellular cell is c_t, the input information is x_t, and the output information is n_t. As opposed to conventional feed-forward neural networks, LSTM has feedback connections. It can also evaluate the entire data flow along with single data points.

C. CONVOLUTIONAL NEURAL NETWORK (CNN)

The CNN has an input layer, an output layer and many hidden layers [54]. Few among them are convolutional layers. The results are transformed into progressive layers using mathematical models to reproduce a section of the human brain in the progressive layer. When a complex model stimulates the development of the potential of artificial intelligence and proposes a structure that accurately reproduces the types of human brain activities, this is an actual example of in-depth insight.

V. PROPOSED HYBRID DEEP LEARNING APPROACH

We suggest a hybrid DL approach integrating DNN and LSTM to identify IoT botnet attacks. Hybrid models are pretty effective at getting high detection accuracy in a short amount of time [55]. Therefore, to benefit from many DL classifiers simultaneously, we have taken into account DNN and LSTM to enhance the final results. IoT devices produce massive amounts of surge data quickly; therefore, in the suggested approach, LSTM is considered owing to its capacity to perform effective learning for longer data sequences. In contrast, DNN is used to increase the algorithm's predictive capability by enhancing speed efficiency. The proposed architecture is depicted in Figure 2, and Table 5 elaborates on the specific configuration of the hybrid architecture we have presented.

A. DATASET

To evaluate and compare our proposed approach, we have used N_BaIoT [56] dataset in this study. This dataset contains traces of normal and malicious IoT traffic. There are 117 attributes total, 116 of which are network features and a tag. Features name is mentioned in [25]. The Botnet traffic consists of six IoT devices mentioned in Table 4. N_BaIoT dataset has benign traffic along with two malware families of Botnet, namely *1) GAFGYT 2) MIRAI*. *Benign* represents the normal traffic in the network. GAFGYT or BASHLITE is a well-known IoT botnet family having various variants. Its variation can enable real-time DDoS attacks, malicious command execution, and malware download and execution. The attacks that the GAFGYT node can launch are Combo (i.e., establishing a connection to a particular IP address and port to send spam), Junk (i.e., sending spam data), TCP flooding, and UDP flooding, and other attacks. While the MIRAI

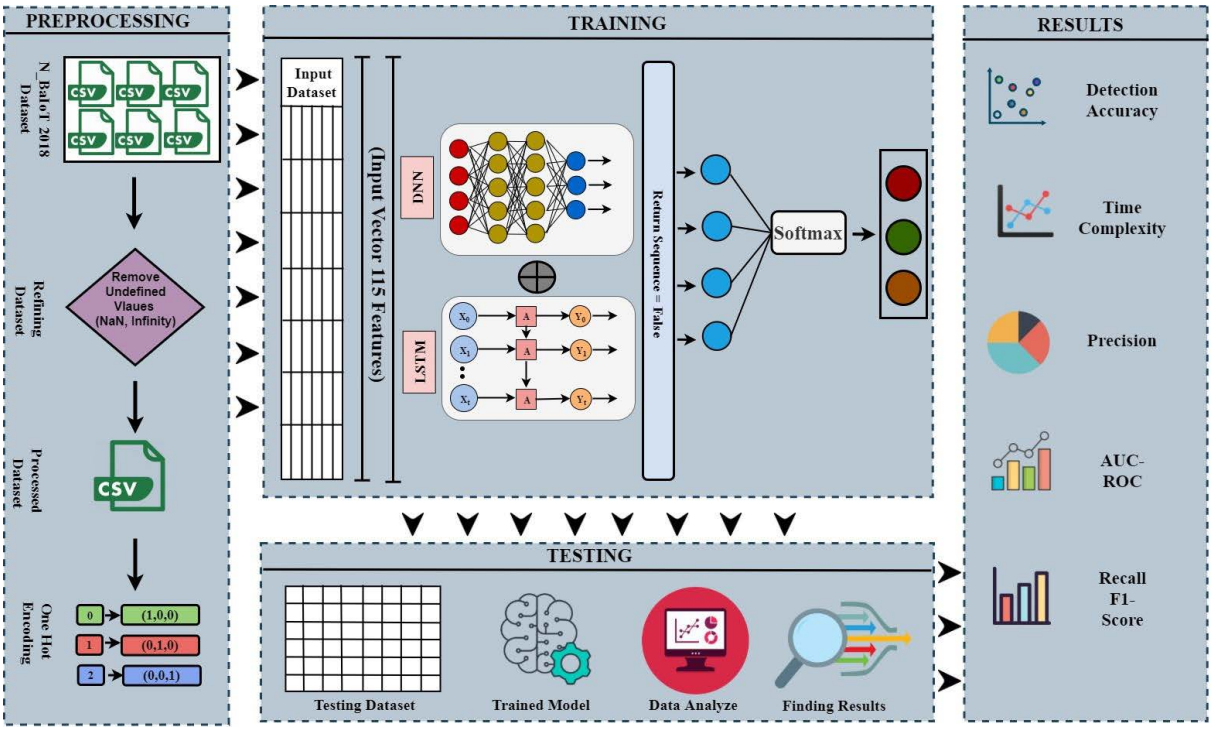


FIGURE 2. Proposed architecture.

TABLE 4. Details of N_BaIoT 2018 dataset.

Devices	Benign	Mirai	Gafgyt
Thermostat	10,000	10,000	10,000
838E Security Camera	10,000	10,000	10,000
737E Security Camera	10,000	10,000	10,000
1011 Web Cam	10,000	–	10,000
1002 Security Camera	10,000	10,000	10,000
1003 Security Camera	10,000	10,000	10,000
Total Records	**60,000**	**50,000**	**60,000**

malware uses ARC processors for targeting smart devices. It transforms them into a network of controllable "zombies" or bots. It can launch UDP flooding, Domain Name Service, TCP ACK, GRE Ethernet, TCP STOMP, HTTP, TCP SYN, UDP Plain, and GRE IP attacks. N_BaIoT dataset is composed of one lac seventy thousand (170,000) records. Of which 60,000 records are for benign traffic, 60,000 are for GAFGYT, and the remaining 50,000 belong to MIRAI traffic. This distribution is also shown in Table 4 with a detailed description of included devices and classes.

B. DETECTION METHODOLOGY

The selection of models used in work is established on the most cutting-adge performers in individual families. The model uses DL methods to produce a adaptable, reliable, and highly accurate botnet antimalware strategy. The proposed detection approach detects benign and malicious behavior by designing a DL-based system. We will use DNN and LSTM to predict results. The findings were extracted and compared to find the best potential solution. Our proposed framework is evaluated using the N_BaIoT dataset.We split the dataset into train and test sets for system analysis and preparation. To train our algorithm, we have given it 90% of the entire dataset, and it will prepare it before making predictions on the test set, which makes up 10% of the whole dataset. The technical details of the proposed and contemporary algorithms are written in Table 5. The table lists each hybrid algorithm's layers, neurons, epochs, optimizer, batch size, and activation functions. The optimizer, activation, and loss functions considered for the implementation are Relu, Categorical cross-entropy, and adam, respectively. The selection of parameters is dependent upon our rigorous experimentation and interactively determining the optimal numbers of layers and neurons along with other parameters. For the DNN-LSTM the pseudo-code is presented in algorithm 2 and the pre-processing phase is presented in algorithm 1. The working of the various phases of the suggested approach is depict in Figure 2 and are explained below:

1) PRE-PROCESSING

The N_BaIoT dataset is pre-processed in order to improve the effectiveness and performance of our proposed hybrid deep learning methodology. In order to guarantee the accuracy of the data, we first inspected the dataset and removed any missing nan and infinite values. To hasten the learning process, we used MinMaxScaler to standardise data between 0 and 1. In addition, we trained a deep learning system on target labels using OHE. The steps for pre-processing are as follows:

TABLE 5. Technical description of proposed algorithm.

Model	Layers	Neurons/Kernel	AF/ LF	Optimizer	Epochs	Batch-size
DNN-LSTM	*DNN Layer* (3)	(400, 100, 50)	*RelU/CC-E*	*Adam*	5	32
	LSTM Layer(3)	(400, 100, 50)	-			
	Merge Layer		-			
	Dense Layer	40	-			
	Dense Layer	15	-			
	Output Layer	3	*softmax*			
CNN2D-LSTM	*CNN2D Layer* (3)	(400, 100, 50)	*RelU/CC-E*	*Adam*	5	32
	LSTM Layer(3)	(400, 100, 50)	-			
	Merge Layer		-			
	Dense Layer	40	-			
	Dense Layer	15	-			
	Output Layer	3	*softmax*			
CNN2D-CNN3D	*CNN2D Layer* (3)	(400, 100, 50)	*RelU/CC-E*	*Adam*	5	32
	CNN3D Layer(3)	(400, 100, 50)	-			
	Merge Layer		-			
	Dense Layer	40	-			
	Dense Layer	15	-			
	Output Layer	3	*softmax*			

AF = Activation Function. LF = Loss Function. CC-E = categorical cross-enotropy.

Step 1 (Input): It is the input stage, where the N_BaIoT dataset is loaded that contains 170,000 IoT traffic which consists of six IoT devices to train our algorithm. Here, six CSV files containing data records for each node are loaded.

Step 2 (Refining Dataset): In this step, the dataset's NAN and infinite values are eliminated because they are the primary causes of the many errors that can occur when the gradient disappears, slowing down the network and rendering it unsafe. The dataset is refined using MinMaxScaller.

Step 3 (Processed Dataset): Here, we get the processed dataset which contains a single CSV file for data collected from all nodes which are free from nan and infinity values.

Step 4 (One Hot encoding): OHE is performed to facilitate the DL algorithm to provide better results. In our case, it normalizes the data according to the three categories based on its label (0, 1, 2).

2) TRAINING

Step 1 (Input pre-processed dataset): 90% of the processed dataset was given as input to our algorithm for training, which comprises 115 features.

Step 2 (Hybrid deep learning phase): In this phase, the hybrid DL technique is applied using two DL algorithms, namely DNN and LSTM. They are executed in parallel on the inputted dataset. Further, the result is merged to get better output.

Step 3 (Add Layer): This layer adds a list of inputs. It accepts a list of similar-shaped tensors as input and outputs a single tensor (also of the same shape).

Step 4 (Return Sequence: Boolean. Whether the final output in the output sequence should be returned or the entire output sequence should be returned. False is the default value.

Step 5 (Softmax function): The softmax function is used as we have multiple classes (Benign, GAFGYT, and MIRAI) that need to be classified properly. As discussed in the results section, it minimizes the prediction errors and improves the detection rate.

Algorithm 1 Pre-Processing

Require: Dataset files Dâ, Dataset Rows R, Dataset Columns C, Combined Dataset D, One Hot Encoding OHE

Ensure: Pre-processing of D

```
function Pre-processing(Dâ)
    D = Merge (Dâ)
    D = Convert Datetime to integer
    D = Labels = OHE(Labels)
    D = Convert Source ip to integer
    D = Convert Destination ip to integer
    for R ← 1 to Rows do
        for C ← 1 to Columns do
            if D[R][C] in ['nan','infinity','null',' ', 'NaN'] then
                Drop Sample (Row)
            end if
        end for
    end for
    Save D as CSV
end function
```

3) TESTING

In this phase, 10% of the dataset has been set aside for testing our algorithm. After performing training of the hybrid algorithm, this dataset is given to analyze the working of the trained algorithm. In the results section, the output of the trained algorithm was gathered and explained.

C. EXPERIMENT DETAILS

The suggested approach's efficiency is demonstrated using various state-of-the-art evaluation measures. These include ROC curve, f1-Score, precision, confusion matrix, recall, and

Algorithm 2 DNN-LSTM

Data training input: There are X network features for every Y unit of network traffic.

Output: is N for normal or label attacks

For each Y_i for N
|
C_i = CNN (Y_i) process
|
End
|
For each C_i process
L_i = LSTM (C_i) process
|
Merge
|
End
For Each L_i Process
N = softmax (L_i) end

TABLE 6. Hardware specification for evaluation of proposed system.

Components	Specification
CPU	Corei7-8750H@2.21GHz
RAM	12GB
OS	Windows 10
Language	Python
Libraries	Numpy, TensorFlow, Scikitlearn, Pandas
Software	Anaconda.Navigator, Spyder 4.1.5, Origin 2019b

accuracy. In table 6 details of the hardware and software resources used in the experiment are shown.

1) CONFUSION MATRIX

Subsequently, other parameters are calculated using the confusion matrix's FP, FN, TP, and TN values. The confusion matrix of proposed and contemporary algorithms are shown in Figure 3-5. Where in Figure 3 the confusion matrix of the proposed model of DNN-LSTM shows the exceptional performance of hybrid DL models to predict and identify attacks efficiently. In Figure 4 hybrid CNN2D-LSTM results are also promising here, while Figure 5 shows the performance of the hybrid DNN2D-DNN3D that performs poor results.

2) ROC CURVE

The ROC graph represents the relationship between the true and FP rates. The AUC (Area Under the Curve) reveals that the larger the AUC, the better the system performance. The ROC curve for the suggested approach is shown in 6.

3) ACCURACY

The accuracy rate shows the number of correctly categorized connections based on the proposed model, including normal and intrusive connections. The formula for accuracy is as

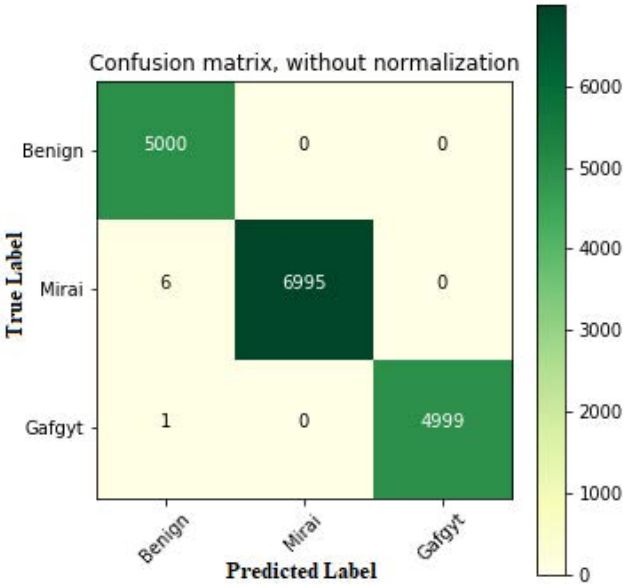


FIGURE 3. Confusion matrix of DNN-LSTM.

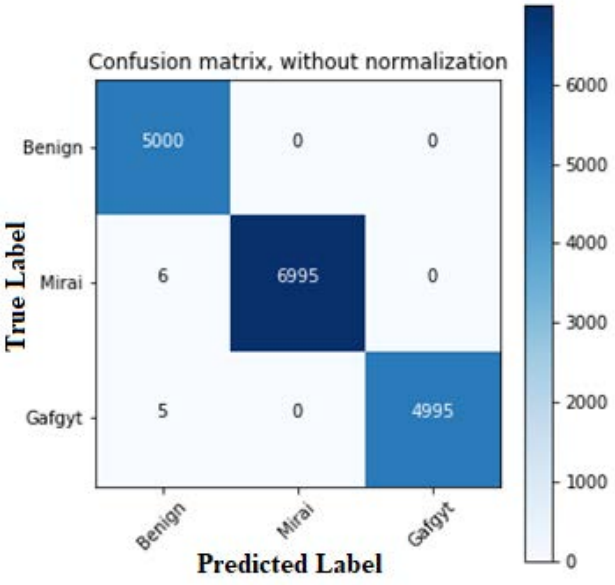


FIGURE 4. Confusion matrix of CNN2D-LSTM.

follows:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3}$$

4) PRECISION

The precision metric, which is the proportion of accurate positive to true positive detection, should also be considered while assessing the proposed model. The following formula can be used to calculate precision:

$$Precision = \frac{TP}{TP + FP} \tag{4}$$

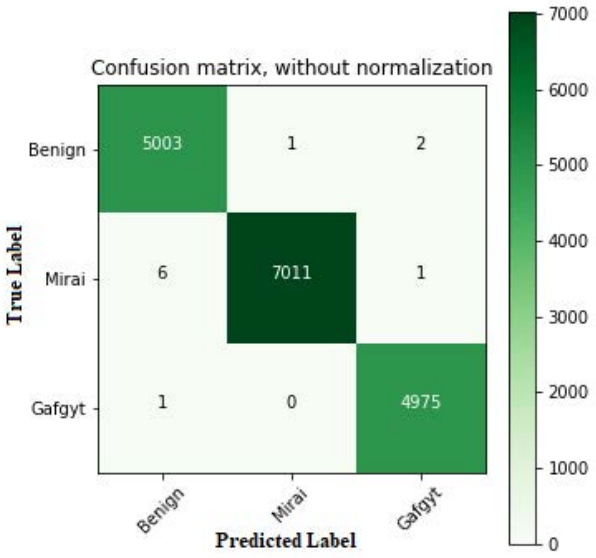


FIGURE 5. Confusion matrix of hybrid CNN2D-CNN3D.

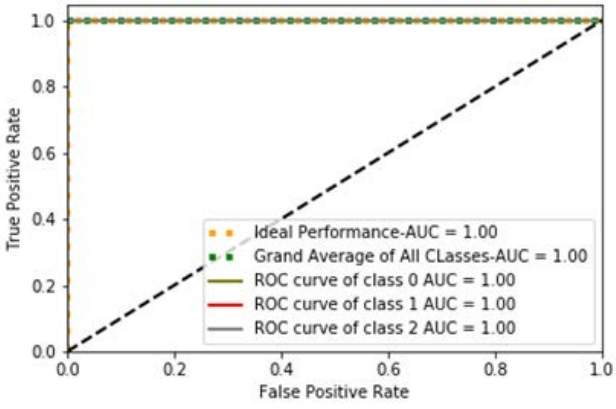


FIGURE 6. ROC curve of LSTM-DNN.

5) RECALL

It is the proportion of exact positive tests to exact malware samples. The following formula can be used to calculate recall:

$$Recall = \frac{TP}{TP + FN} \tag{5}$$

6) F1-SCORE

The accuracy of a model on a dataset is measured by the F-score, also known as the F1-score. The f1-score value is calculated using a formula that weights the precision and recall rates. The following formula is used for the calculation of F1-score:

$$F - score = \frac{2 * TP}{2 * TP + FP + FN} \tag{6}$$

VI. RESULT AND DISCUSSION

We also compare our proposed algorithm with other constructed algorithms to evaluate the proposed system. We used a cross-validation method that divided the training data into several folds (k), using a subset of the data as a test set and the remaining (k-1) subset as a training set each time to gather precise information about the performance of our model. As a result, the data over-or under-fitting problem was resolved, and detailed information on the model's performance for unexpected data during training was provided. Table 7 shows the 10-fold cross-validated findings of our hybrid algorithms compared to other proposed hybrid DL models.

A. F1-SCORE, RECALL, PRECISION AND ACCURACY

The DL-based botnet detection in fog computing using SDN provides an efficient and scalable solution. In order to attain a high accuracy rate, hybrid DL models are used. We employed 10-fold cross-validation to acquire the average detection accuracy, precision, recall, and F1-score to get objective results. The results are presented in a ROC Curve, which can maximize the true positive value while minimizing the false positive value. The detection accuracy, precision, recall, and f1-score of hybrid DNN-LSTM, hybrid CNN2D-LSTM, and CNN2D-CNN3D are also evaluated as part of the evaluation matrix. For Hybrid DNN-LSTM, Hybrid CNN2D-LSTM, and Hybrid CNN2D-CNN3D, Figure 7 shows accuracy, recall, precision, and F1-score. Hybrid DNN-LSTM outperforms other contemporary models in terms of accuracy. For the proposed hybrid DNN-LSTM the accuracy is 99.98%, precision is 99.97%, recall is 99.87%, and f1-score is 99.87%. Likewise, for the hybrid CNN2D-LSTM, the accuracy is 99.95%, precision is 99.94%, and for the recall and f1-score, it is 99.85%. Additionally, for the hybrid CNN2D-CNN3D algorithm, the accuracy is 99.93%, precision is 99.91%, and for the recall and f1-score, it is 99.86%.

B. FPR, FDR, FNR AND FOR

To further enhance the evaluation criteria and evaluate models with better insight, advanced parameters such as FOR, FDR, FPR and FNR have also been used in the evaluation, presented in Figure 8. The high result values of TNR and low values of FNR signify the importance of the proposed Hybrid models for attack detection. The Hybrid DNN-LSTM gain FPR of 0.00009%, FDR 0.0002%, FNR 0.0012% and FOR 0.0005%. Furthermore, the hybrid CNN2D-LSTM gained an FPR of 0005%, FDR 0.0001%, FNR 0.0014% and FOR 0.0005%. Additionally, for the hybrid CNN2D-CNN3D gain FPR of 0.0002%, FDR 0.0006%, FNR 0.0014% and FOR 0.0007%.

For a more detailed examination of the suggested framework, the authors calculated FNR, FPR, FDR, and FOR values. Furthermore, the low rates of the early indicators indicate that the algorithm is performing well. Where the FPR measures the percentage of false positives against all positive predictions, The FDR is the proportion of hypotheses that we falsely think are true, FNR is negative outcomes that the model predicted incorrectly, and The FOR measures most individuals whose test results were negative but whose actual condition was positive.

TABLE 7. 10 Fold of hybrid DNN-LSTM, hybrid CNN2D-LSTM, and hybrid CNN2D-CNN3D.

	Accuracy (%)			Recall (%)			Precision (%)			F1-Score (%)		
Folds	!!	@@	++	!!	@@	++	!!	@@	++	!!	@@	++
1	**99.99**	99.96	99.96	**99.99**	**99.99**	**99.99**	**99.99**	99.90	99.90	**99.99**	**99.99**	**99.99**
2	99.93	99.73	99.72	99.90	**99.99**	**99.99**	99.93	99.90	99.90	99.90	**99.99**	**99.99**
3	**99.99**	**99.99**	**99.99**	**99.99**	**99.99**	**99.99**	**99.99**	**99.99**	**99.99**	**99.99**	**99.99**	**99.99**
4	**99.99**	99.86	99.86	**99.99**	99.70	99.70	**99.99**	99.90	99.90	**99.99**	99.70	99.70
5	99.96	99.93	99.93	99.89	99.90	99.90	**99.99**	99.90	99.90	99.89	99.90	99.90
6	**99.99**	**99.99**	**99.99**	**99.99**	**99.99**	**99.99**	**99.99**	**99.99**	**99.99**	**99.99**	**99.99**	**99.99**
7	99.96	**99.99**	**99.99**	99.90	**99.99**	**99.99**	**99.99**	**99.99**	**99.99**	99.90	**99.99**	**99.99**
8	**99.99**	**99.99**	**99.99**	**99.99**	**99.99**	**99.99**	**99.99**	**99.99**	**99.99**	**99.99**	**99.99**	**99.99**
9	**99.99**	**99.99**	**99.99**	**99.99**	**99.99**	**99.99**	**99.99**	**99.99**	**99.99**	**99.99**	**99.99**	**99.99**
10	**99.99**	99.96	99.96	**99.99**	99.90	9.90	**99.99**	**99.99**	**99.99**	**99.99**	99.90	99.90

Abbreviation Terms: !! Hybrid (DNN & LSTM), @@ Hybrid (CNN2D & LSTM), ++ Hybrid (CNN2D & CNN3D)

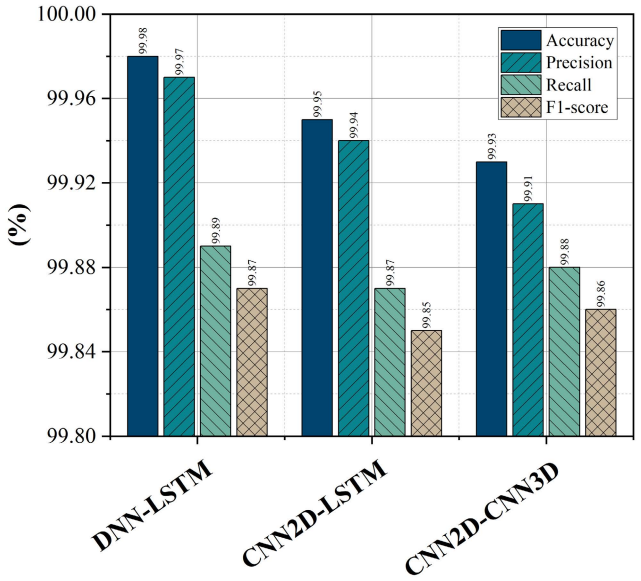


FIGURE 7. Accuracy, Precision, Recall and F1-Score of DNN-LSTM, CNN2D-LSTM, and CNN2D-CNN3D.

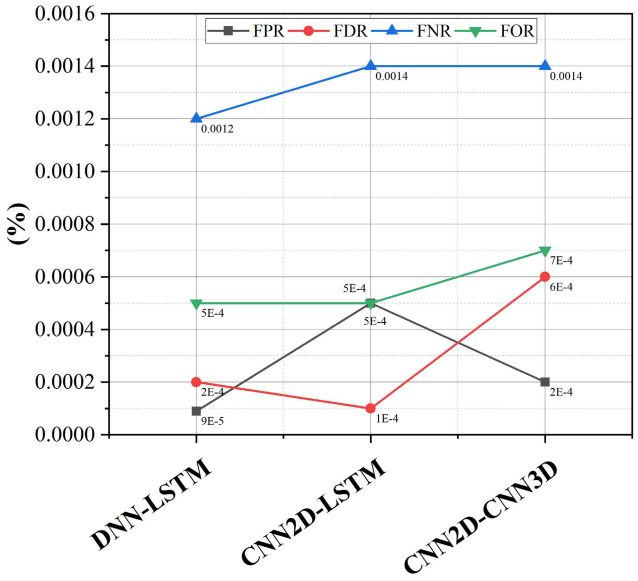


FIGURE 8. FPR, FDR, FNR, and FOR of DNN-LSTM, CNN2D-LSTM, and CNN2D-CNN3D.

C. TNR, NPV AND MCC

We calculated additional measures (i.e., TNR, NPV, MCC) from the confusion matrix to thoroughly analyze our proposed algorithm. The confusion matrix depicts detection performance with TP, TN, FP, and FN values, providing precise results where our proposed algorithm performs best.

The parameter's i.e. TNR, NPV, and MCC are presented in Figure 9. For the hybrid DNN-LSTM algorithm, the value of TNR is 99.99%, the MCC value is 99.93%, and the NPV value is 99.94%. For the CNN2D-LSTM algorithm, the value of TNR is 99.97%, MCC is 99.92%, and NPV is 99.94%. Likewise, for the hybrid CNN2D-CNN3D algorithm, the value of TNR is 99.96%, MCC is 99.91%, and NPV is 99.93%. Our proposed algorithm has the highest value for TNR and MCC while having the same NPV value as CNN2D-LSTM.

D. BM, MK AND TS

BM is a worldwide test performance measure used to assess a diagnostic procedure's overall discriminative power. MK is the state of standing out as divergent compared to the standard form. TS measures the ability of an algorithm and is often used to compute the results over time. The hybrid DNN-LSTM algorithm value for BM is 99.85%, MK is 99.92%, and TS is 99.84%. For the hybrid CNN2D-LSTM, the value for BM is 99.85%, MK is 99.89%, and TS is 99.82%. Additionally, The hybrid CNN2D-CNN3D algorithm shows the value of BM, MK, and TS as 99.82%, 99.86%, and 99.71%, respectively. The proposed algorithm values for BM, MK, and TS are presented in Figure 10, where our proposed algorithm has the highest value for MK and TS while having the same BM value as CNN2D-LSTM.

E. TESTING TIME

Figure 11 depicts the testing time of the three hybrid combinations of DL algorithms used in this research work, i.e., DNN-LSTM, CNN2D-CNN3D, and CNN2D-LSTM. The CNN2D-LSTM and CNN2D-CNN3D hybrid approaches have a 0.14ms and 0.19ms testing time, respectively,

TABLE 8. Comparison of proposed work with other solutions.

Method	Models	Dataset	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)	Detection Time
Proposed	DNN-LSTM	N_BaIoT 2018	99.98	99.97	99.87	99.87	0.22(ms)
[16]	LSTM	N_BaIoT 2018	99.97	99.90	99.90	99.85	2.65
[45]	CNN-RNN	CTU-13, ISOT	99.00	98.00	97.00	97.00	–
[47]	LSTM-CNN	CIDDS2017	99.92	99.85	99.85	99.91	29
[57]	DNN	Network Flow	99.00	–	–	–	–
[58]	MLP-AE	N_BaIoT 2018	99.25	98.84	98.00	98.00	3.75(ms)
[59]	DBN	N_BaIoT 2018	95.60	98.27	92.82	92.82	–
[60]	CNN, LSTM	N_BaIoT 2018	94.30	93.48	93.67	93.58	–
[61]	KNN, RF, NB	N_BaIoT 2018	99.00	86.65	99.00	99.00	–

Abbreviation Terms: ms– millisecond.

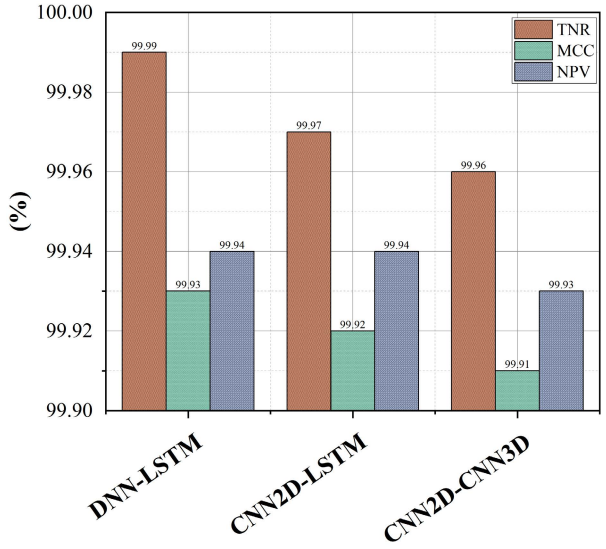


FIGURE 9. TNR, NPV and MCC of DNN-LSTM, CNN2D-LSTM, and CNN2D-CNN3D.

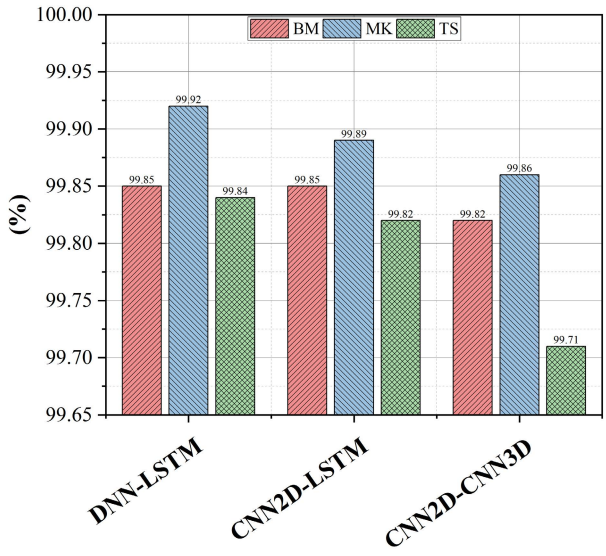


FIGURE 10. BM, MK and TS of DNN-LSTM, CNN2D-LSTM, and CNN2D-CNN3D.

as compared to our suggested hybrid technique DNN-LSTM, which takes 0.22ms. The suggested hybrid DL DNN-LSTM

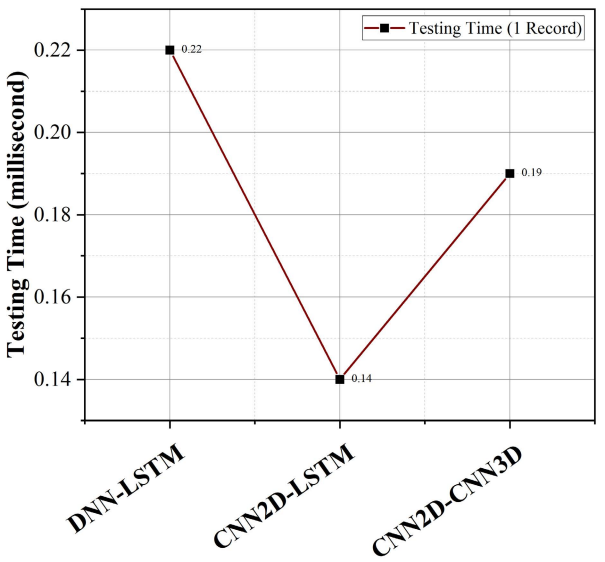


FIGURE 11. Testing time of DNN-LSTM, CNN2D-LSTM, and CNN2D-CNN3D.

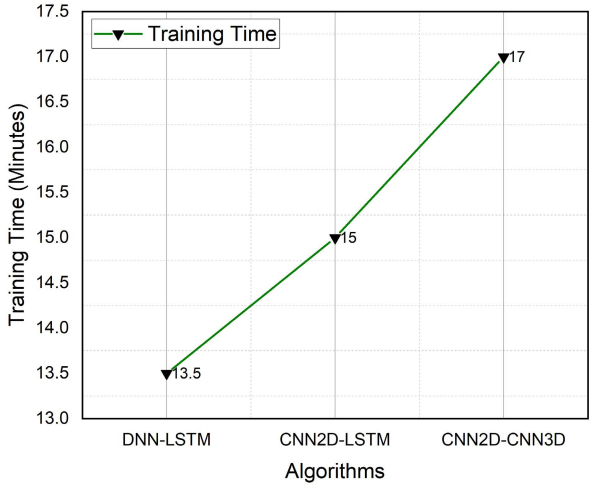


FIGURE 12. Training time of DNN-LSTM, CNN2D-LSTM and CNN2D-CNN3D.

has a negligible compromise in testing time compared to CNN2D-LSTM and CNN2D-CNN3D hybrid models.

Hence, we compared these combinations in various ways in Table 8 to highlight the selection of our suggested model

in terms of the f1-score, accuracy, precision and recall. The findings show that the suggested hybrid model, which uses the DNN-LSTM approach, achieves better f1-score, recall, and accuracy while precision is equivalent to CNN2D-LSTM and CNN2D-CNN3D hybrid models.

F. TRAINING TIME

The training time for each algorithm is presented in Figure 12. Training our proposed algorithm took 13.5 minutes using the N_BaIoT dataset. This time is mentioned here to show that the proposed model takes minimum time for training. The results depicted in Figure 12 show the difference between various algorithms according to their time utilization for training. It shows that the proposed scheme DNN-LSTM utilizes minimum time compared to other counterparts.

VII. CONCLUSION

SDN-based fog computing architectures are the trending networking paradigms for several applications based on the IoT infrastructure. Fog computing systems are vulnerable to various types of Botnet attacks. Hence, there is a need to integrate a security framework that empowers the SDN to monitor the network anomalies against the Botnet attacks. DL algorithms are considered more effective for the IoT-based infrastructures that work on unstructured and large amounts of data. DL-based intrusion detection schemes can detect Botnet attacks in the SDN-enabled fog computing IoT system.

We created a framework that utilizes a hybrid DL detection scheme to identify the IoT botnet attacks. It is trained against the dataset that contains normal and malicious data, and then we used this framework to identify botnet attacks that targeted different IoT devices. Our methodology comprises a botnet dataset, a botnet training paradigm, and a botnet detection paradigm.

Our botnet dataset was built using the N_BaIoT dataset, which was produced by driving botnet attacks from the Gafgyt and Mirai botnets into six distinct types of IoT devices. Five attack types, including UDP, TCP, and ACK, are included in both Gafgyt and Mirai attacks. We developed a botnet detection based on three hybrid models—DNN-LSTM, CNN2D-LSTM, and CNN2D-CNN3D. Using this training model as a foundation, we developed a botnet detection paradigm that can recognise significant botnet attacks. The botnet detection approach is part of a multi-class classification model that can distinguish between the sub-attacks and innocuous data. The fact-finding analysis showed that our hybrid framework DNN-LSTM model had the highest accuracy of 99.98% at identifying the gafgyt and Mirai botnets in the N_BaIoT environment. In 2014 and 2016, the gafgyt and Mirai botnets essentially targeted home routers and IP cameras. The N_BaIoT dataset we used for our experiments revealed that rather than the type of IoT devices, the type of training models has a more significant impact on botnet detection performance. We think creating DNN-LSTM-based IoT botnet detection models would be an excellent strategy to enhance botnet identification for different IoT devices.

In the future, we have in mind to compare the performance of the proposed hybrid algorithm to that of other IoT datasets with a more considerable number of nodes. Further, there is a need to test more combinations of DL algorithms and traditional machine learning algorithms.

REFERENCES

[1] Z. Hussain, A. Akhunzada, J. Iqbal, I. Bibi, and A. Gani, "Secure IIoT-enabled industry 4.0," *Sustainability*, vol. 13, no. 22, p. 12384, Nov. 2021.

[2] R. K. Barik, H. Dubey, K. Mankodiya, S. A. Sasane, and C. Misra, "GeoFog4Health: A fog-based SDI framework for geospatial health big data analysis," *J. Ambient Intell. Hum. Comput.*, vol. 10, no. 2, pp. 551–567, Feb. 2019.

[3] S. Khan, S. Parkinson, and Y. Qin, "Fog computing security: A review of current applications and security solutions," *J. Cloud Comput.*, vol. 6, no. 1, pp. 1–22, Dec. 2017.

[4] J. Malik, A. Akhunzada, I. Bibi, M. Talha, M. A. Jan, and M. Usman, "Security-aware data-driven intelligent transportation systems," *IEEE Sensors J.*, vol. 21, no. 14, pp. 15859–15866, Jul. 2021.

[5] Z. Ning, X. Hu, Z. Chen, M. Zhou, B. Hu, J. Cheng, and M. S. Obaidat, "A cooperative quality-aware service access system for social internet of vehicles," *IEEE Internet Things J.*, vol. 5, no. 4, pp. 2506–2517, Aug. 2018.

[6] X. Wang, Z. Ning, M. C. Zhou, X. Hu, L. Wang, Y. Zhang, F. R. Yu, and B. Hu, "Privacy-preserving content dissemination for vehicular social networks: Challenges and solutions," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 2, pp. 1314–1345, 2nd Quart., 2019.

[7] Z. Ning, Y. Li, P. Dong, X. Wang, M. S. Obaidat, X. Hu, L. Guo, Y. Guo, J. Huang, and B. Hu, "When deep reinforcement learning meets 5G-enabled vehicular networks: A distributed offloading framework for traffic big data," *IEEE Trans. Ind. Informat.*, vol. 16, no. 2, pp. 1352–1361, Feb. 2020.

[8] X. Wang, Z. Ning, and L. Wang, "Offloading in internet of vehicles: A fog-enabled real-time traffic management system," *IEEE Trans. Ind. Informat.*, vol. 14, no. 10, pp. 4568–4578, Oct. 2018.

[9] H. Dubey, J. Yang, N. Constant, A. M. Amiri, Q. Yang, and K. Makodiya, "Fog data: Enhancing telehealth big data through fog computing," in *Proc. ASE BigData Socialinform.*, 2015, pp. 1–6.

[10] W. U. Khan, T. N. Nguyen, F. Jameel, M. A. Jamshed, H. Pervaiz, M. A. Javed, and R. Jäntti, "Learning-based resource allocation for backscatter-aided vehicular networks," *IEEE Trans. Intell. Transp. Syst.*, early access, Nov. 18, 2021, doi: 10.1109/TITS.2021.3126766.

[11] A. M. Rahmani, T. N. Gia, B. Negash, A. Anzanpour, I. Azimi, M. Jiang, and P. Liljeberg, "Exploiting smart e-health gateways at the edge of healthcare Internet-of-Things: A fog computing approach," *Future Gener. Comput. Syst.*, vol. 78, pp. 641–658, Jan. 2018.

[12] Z. Xiao and Y. Xiao, "Security and privacy in cloud computing," *IEEE Commun. Surveys Tuts.*, vol. 15, no. 2, pp. 843–859, 2nd Quart., 2013.

[13] Q. Yan, F. R. Yu, Q. Gong, and J. Li, "Software-defined networking (SDN) and distributed denial of service (DDoS) attacks in cloud computing environments: A survey, some research issues, and challenges," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 1, pp. 602–622, 1st Quart., 2016.

[14] S. S. C. Silva, R. M. P. Silva, R. C. G. Pinto, and R. M. Salles, "Botnets: A survey," *Comput. Netw.*, vol. 57, no. 2, pp. 378–403, 2013.

[15] J. Malik, A. Akhunzada, I. Bibi, M. Imran, A. Musaddiq, and S. W. Kim, "Hybrid deep learning: An efficient reconnaissance and surveillance detection mechanism in SDN," *IEEE Access*, vol. 8, pp. 134695–134706, 2020.

[16] T. Hasan, A. Adnan, T. Giannetsos, and J. Malik, "Orchestrating SDN control plane towards enhanced IoT security," in *Proc. 6th IEEE Conf. Netw. Softw. (NetSoft)*, Jun. 2020, pp. 457–464.

[17] W. U. Khan, J. Liu, F. Jameel, V. Sharma, R. Jäntti, and Z. Han, "Spectral efficiency optimization for next generation NOMA-enabled IoT networks," *IEEE Trans. Veh. Technol.*, vol. 69, no. 12, pp. 15284–15297, Dec. 2020.

[18] L. Yu, Q. Wang, G. Barrineau, J. Oakley, R. R. Brooks, and K.-C. Wang, "TARN: A SDN-based traffic analysis resistant network architecture," in *Proc. 12th Int. Conf. Malicious Unwanted Softw. (MALWARE)*, Oct. 2017, pp. 91–98.

[19] E. Rodríguez, B. Otero, N. Gutiérrez, and R. Canal, "A survey of deep learning techniques for cybersecurity in mobile networks," *IEEE Commun. Surveys Tuts.*, vol. 23, no. 3, pp. 1920–1955, 3rd Quart., 2021.

[20] T. A. Tang, L. Mhamdi, D. McLernon, S. A. R. Zaidi, and M. Ghogho, "Deep recurrent neural network for intrusion detection in SDN-based networks," in *Proc. 4th IEEE Conf. Netw. Softw. Workshops (NetSoft)*, Jun. 2018, pp. 202–206.

[21] R. Chen, W. Niu, X. Zhang, Z. Zhuo, and F. Lv, "An effective conversation-based botnet detection method," *Math. Problems Eng.*, vol. 2017, pp. 1–9, Apr. 2017.

[22] S. Al-mashhadi, M. Anbar, I. Hasbullah, and T. A. Alamiedy, "Hybrid rule-based botnet detection approach using machine learning for analysing DNS traffic," *PeerJ Comput. Sci.*, vol. 7, p. e640, Aug. 2021.

[23] A. O. Prokofiev, Y. S. Smirnova, and V. A. Surov, "A method to detect Internet of Things botnets," in *Proc. IEEE Conf. Russian Young Res. Electr. Electron. Eng. (EIConRus)*, Jan. 2018, pp. 105–108.

[24] M. Waqas, K. Kumar, A. A. Laghari, U. Saeed, M. M. Rind, A. A. Shaikh, F. Hussain, A. Rai, and A. Q. Qazi, "Botnet attack detection in Internet of Things devices over cloud environment via machine learning," *Concurrency Comput., Pract. Exp.*, vol. 34, no. 4, Feb. 2022, Art. no. e6662.

[25] J. A. Faysal, S. T. Mostafa, J. S. Tamanna, K. M. Mumenin, M. M. Arifin, M. A. Awal, A. Shome, and S. S. Mostafa, "XGB-RF: A hybrid machine learning approach for IoT intrusion detection," *Telecom*, vol. 3, no. 1, pp. 52–69, Jan. 2022.

[26] A. O. Khadidos, H. Manoharan, S. Selvarajan, A. O. Khadidos, K. H. Alyoubi, and A. Yafoz, "A classy multifacet clustering and fused optimization based classification methodologies for SCADA security," *Energies*, vol. 15, no. 10, p. 3624, May 2022.

[27] S. Shitharth, K. M. Prasad, K. Sangeetha, P. R. Kshirsagar, T. S. Babu, and H. H. Alhelou, "An enriched RPCO-BCNN mechanisms for attack detection and classification in SCADA systems," *IEEE Access*, vol. 9, pp. 156297–156312, 2021.

[28] B. Jo, R. Khan, and Y.-S. Lee, "Hybrid blockchain and Internet-of-Things network for underground structure health monitoring," *Sensors*, vol. 18, no. 12, p. 4268, Dec. 2018.

[29] T. G. Nguyen, T. V. Phan, B. T. Nguyen, C. So-In, Z. A. Baig, and S. Sanguanpong, "SeArch: A collaborative and intelligent NIDS architecture for SDN-based cloud IoT networks," *IEEE Access*, vol. 7, pp. 107678–107694, 2019.

[30] S. M. Umran, S. Lu, Z. A. Abduljabbar, J. Zhu, and J. Wu, "Secure data of industrial Internet of Things in a cement factory based on a blockchain technology," *Appl. Sci.*, vol. 11, no. 14, p. 6376, Jul. 2021.

[31] S. Miller and C. Busby-Earle, "The role of machine learning in botnet detection," in *Proc. 11th Int. Conf. for Internet Technol. Secured Trans. (ICITST)*, Dec. 2016, pp. 359–364.

[32] P. C. Tikekar, S. S. Sherekar, and V. M. Thakre, "Features representation of botnet detection using machine learning approaches," in *Proc. Int. Conf. Comput. Intell. Comput. Appl. (ICCICA)*, Nov. 2021, pp. 1–5.

[33] A. Prakash and R. Priyadarshini, "An intelligent software defined network controller for preventing distributed denial of service attack," in *Proc. 2nd Int. Conf. Inventive Commun. Comput. Technol. (ICICCT)*, Apr. 2018, pp. 585–589.

[34] T. V. Phan, T. Van Toan, D. Van Tuyen, T. T. Huong, and N. H. Thanh, "OpenFlowSIA: An optimized protection scheme for software-defined networks from flooding attacks," in *Proc. IEEE 6th Int. Conf. Commun. Electron. (ICCE)*, Jul. 2016, pp. 13–18.

[35] T. Abhiroop, S. Babu, and B. S. Manoj, "A machine learning approach for detecting DoS attacks in SDN switches," in *Proc. 24th Nat. Conf. Commun. (NCC)*, Feb. 2018, pp. 1–6.

[36] J. Ye, X. Cheng, J. Zhu, L. Feng, and L. Song, "A DDoS attack detection method based on SVM in software defined network," *Secur. Commun. Netw.*, vol. 2018, pp. 1–8, Apr. 2018.

[37] S. Almutairi, S. Mahfoudh, S. Almutairi, and J. S. Alowibdi, "Hybrid botnet detection based on host and network analysis," *J. Comput. Netw. Commun.*, vol. 2020, pp. 1–16, Jan. 2020.

[38] G. Kocher and G. Kumar, "Machine learning and deep learning methods for intrusion detection systems: Recent developments and challenges," *Soft Comput.*, vol. 25, no. 15, pp. 9731–9763, Aug. 2021.

[39] H. Alkahtani and T. H. H. Aldhyani, "Botnet attack detection by using CNN-LSTM model for Internet of Things applications," *Secur. Commun. Netw.*, vol. 2021, pp. 1–23, Sep. 2021.

[40] B. Susilo and R. F. Sari, "Intrusion detection in IoT networks using deep learning algorithm," *Information*, vol. 11, no. 5, p. 279, May 2020.

[41] A. K. Sahu, S. Sharma, M. Tanveer, and R. Raja, "Internet of Things attack detection using hybrid deep learning model," *Comput. Commun.*, vol. 176, pp. 146–154, Aug. 2021.

[42] D. Javeed, T. Gao, M. T. Khan, and I. Ahmad, "A hybrid deep learning-driven SDN enabled mechanism for secure communication in Internet of Things (IoT)," *Sensors*, vol. 21, no. 14, p. 4884, Jul. 2021.

[43] C. D. McDermott, F. Majdani, and A. V. Petrovski, "Botnet detection in the Internet of Things using deep learning approaches," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2018, pp. 1–8.

[44] A. Dawoud, S. Shahristani, and C. Raun, "Deep learning and software-defined networks: Towards secure IoT architecture," *Internet Things*, vols. 3–4, pp. 82–89, Oct. 2018.

[45] A. Pektaş and T. Acarman, "Deep learning to detect botnet via network flow summaries," *Neural Comput. Appl.*, vol. 31, no. 11, pp. 8021–8033, Nov. 2019.

[46] S. K. Dey and M. M. Rahman, "Flow based anomaly detection in software defined networking: A deep learning approach with feature selection method," in *Proc. 4th Int. Conf. Electr. Eng. Inf. Commun. Technol. (iCEEiCT)*, Sep. 2018, pp. 630–635.

[47] I. Ullah, B. Raza, S. Ali, I. A. Abbasi, S. Baseer, and A. Irshad, "Software defined network enabled fog-to-things hybrid deep learning driven cyber threat detection system," *Secur. Commun. Netw.*, vol. 2021, pp. 1–15, Dec. 2021.

[48] P. Jithu, J. Shareena, A. Ramdas, and A. P. Haripriya, "Intrusion detection system for IoT botnet attacks using deep learning," *Social Netw. Comput. Sci.*, vol. 2, no. 3, pp. 1–8, May 2021.

[49] S. Shitharth, P. R. Kshirsagar, P. K. Balachandran, K. H. Alyoubi, and A. O. Khadidos, "An innovative perceptual pigeon galvanized optimization (PPGO) based likelihood Naïve Bayes (LNB) classification approach for network intrusion detection system," *IEEE Access*, vol. 10, pp. 46424–46441, 2022.

[50] A. Canziani, A. Paszke, and E. Culurciello, "An analysis of deep neural network models for practical applications," 2016, *arXiv:1605.07678*.

[51] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[52] *'Top Providers of SDN Services*. Accessed: Jan. 15, 2022. [Online]. Available: https://www.cambridgewireless.co.uk and https://www.cambridgewireless.co.uk/news/2020/jan/28/top-providers-sdn-services/

[53] G. Kaur, "A novel distributed machine learning framework for semi-supervised detection of botnet attacks," in *Proc. 11th Int. Conf. Contemp. Comput. (IC3)*, Aug. 2018, pp. 1–7.

[54] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *Proc. Int. Conf. Eng. Technol. (ICET)*, Aug. 2017, pp. 1–6.

[55] M. M. Hassan, A. Gumaei, A. Alsanad, M. Alrubaian, and G. Fortino, "A hybrid deep learning model for efficient intrusion detection in big data environment," *Inf. Sci.*, vol. 513, pp. 386–396, Mar. 2020.

[56] Y. Meidan, M. Bohadana, Y. Mathov, Y. Mirsky, A. Shabtai, D. Breitenbacher, and Y. Elovici, "N-BaIoT—Network-based detection of IoT botnet attacks using deep autoencoders," *IEEE Pervasive Comput.*, vol. 17, no. 3, pp. 12–22, Jul./Aug. 2018.

[57] A. Pektaş and T. Acarman, "Botnet detection based on network flow summary and deep learning," *Int. J. Netw. Manage.*, vol. 28, no. 6, Nov. 2018, Art. no. e2039.

[58] V. Rey, P. M. S. Sánchez, A. H. Celdrán, and G. Bovet, "Federated learning for malware detection in IoT devices," *Comput. Netw.*, vol. 204, Feb. 2022, Art. no. 108693.

[59] T. V. Khoa, Y. M. Saputra, D. T. Hoang, N. L. Trung, D. Nguyen, N. V. Ha, and E. Dutkiewicz, "Collaborative learning model for cyberattack detection systems in IoT industry 4.0," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, May 2020, pp. 1–6.

[60] G. De La Torre Parra, P. Rad, K.-K.-R. Choo, and N. Beebe, "Detecting Internet of Things attacks using distributed deep learning," *J. Netw. Comput. Appl.*, vol. 163, Aug. 2020, Art. no. 102662.

[61] H. Alazzam, A. Alsmady, and A. A. Shorman, "Supervised detection of IoT botnet attacks," in *Proc. 2nd Int. Conf. Data Sci., E-Learn. Inf. Syst.*, 2019, pp. 1–6.

FRAIDOON SATTARI received the bachelor's degree in computer science from Gomal University, D. I. Khan, Pakistan. He is currently pursuing the M.S. degree in information security with COMSATS University Islamabad, Islamabad Campus, Pakistan. His research interests include network security, cloud computing, malware analysis and detection, software-defined networking, fog security, and the Internet of Things.

ASHFAQ HUSSAIN FAROOQI received the master's and Ph.D. degrees in computer science from the National University of Computer and Emerging Sciences, Islamabad, Pakistan, in 2009 and 2018, respectively. He is currently working as an Assistant Professor with the Department of Computer Science, Air University, Islamabad. He has authored several articles in refereed journals. His research interests include wireless communication, network security, computational intelligence, risk assessment, and risk management.

ZAKRIA QADIR received the M.Sc. degree in sustainable environment and energy systems from Middle East Technical University, Turkey, in 2019. He is currently pursuing the Ph.D. degree in wireless communication and cloud computing with Western Sydney University, Australia. His research interests include sustainable cities, artificial intelligence, machine learning, optimization techniques, wireless communication, the Internet of things, renewable energy technology, and cloud computing.

BASIT RAZA received the master's degree in computer science from the University of Central Punjab, Lahore, Pakistan, and the dual Ph.D. degree in computer science from International Islamic University Islamabad and the University of Technology Malaysia, in 2014. He is currently an Associate Professor with the Department of Computer Science, COMSATS University Islamabad (CUI), Islamabad, Pakistan. He has authored several articles in refereed journals. His research interests include database management systems, data mining, data warehousing, machine learning, deep learning, and artificial intelligence. He has been serving as a Reviewer for prestigious journals, such as *Applied Soft Computing*, *Swarm and Evolutionary Computation*, *Swarm Intelligence*, *Applied Intelligence*, IEEE Access, and *Future Generation Computer Systems*.

HADI NAZARI received the bachelor's degree in computer science from Gomal University, D. I. Khan, Pakistan. He is currently pursuing the M.S. degree in information security with the Department of Computer Science, COMSATS University Islamabad, Islamabad, Pakistan. His research interests include an access control systems, software-defined networking, smart devices security, threat detection and intelligence, malware analysis and detection, application of deep learning and machine learning in cyber defense, and network security.

MUHANNAD ALMUTIRY (Member, IEEE) received the B.Sc. degree in electrical engineering from Umm Al Qura University, Makkah, Saudi Arabia, in 2007, and the M.Sc. and Ph.D. degrees in electrical engineering from the University of Dayton, Dayton, OH, USA, in 2010 and 2016, respectively. From 2013 to 2016, he was a Research Assistant with the Mumma Radar Laboratory, University of Dayton. He has been an Assistant Professor with the Department of Electrical Engineering, Northern Border University, Saudi Arabia, since 2016. His research interests include radar imaging, subsurface sensing, UAV detection, sonar, and intelligent adaptive radar networks.

INTRODUCTION

One of the most significant issues for the network system to be efficient and reliable while doing transactions over the IoT is security [1]. The tremendous growth of IoT in different fields, i.e., surveillance, healthcare, transportation, manufacturing industry, education, and others, encourages securing IoT infrastructure to improve its performance. Earlier IoT devices generate data through various types of sensors, and it becomes tidy for the cloud servers to handle or process these transactions efficiently. Fog computing is among the newly proposed schemes that could be utilized to add preferred features to the IoT infrastructure [2]. Fog computing is competent in doing some regional analysis of information [3] before communicating the aggregated data to the cloud server. It helps in keeping the latency constraints in some time compelled real-time issues, making them appropriate for IoT-based applications such as vehicular ad-hoc networks (VANETs) [4]_[11]. These advancements towards using fog servers in IoT infrastructure motivate the adversaries to target the fog server with malicious intent to lower its performance. Hence, security and protection of the system are among the major issues that can affect the performance of fog computing [12]. In this regard, availability is among the core security requirements for offering services to the actual customer applications according to their interest. However, this is constantly tested by the adversaries by launching different types of attacks, such as DoS or DDoS attacks [13]. An individual or a group can perform these attacks. If a group performs it, it is named ``botnet," while if an individual launches it, it is known as ``bot-master." [14]. The bot-master is the attacker node that can launch several types of attacks on the server, such as Phishing, spam, Click fraud, and others. A command-and-control channel remotely controls a botnet. The command-and-control channel is a system the adversary uses to control by sending

messages and commands to a compromised system. The adversary can steal the data through these commands and manipulate the infected network [8]. In a botnet attack, some `n' number of compromised nodes are controlled by a bot-master, and they launch an attack on the server from different compromised systems.

In the fog computing paradigm security is still challenging task, and various security schemes are proposed to make it resilient against vulnerabilities. However, most of the schemes focus on flexibility and continuous monitoring of the fog server. Software-denied networking (SDN) is used at fog servers to address flexibility, and continuous monitoring issues [15]. SDN is an emerging networking paradigm that assists in making the network more flexible that can help in managing the network, analyzing the traffic, and assisting in the routing control architectures [16], [17] as there is a separate control plan that provides a flexible device management policy. Hence, an SDN-based fog computing environment provides centralized control to the fog computing system. The characteristics of the SDN based fog computing system are discussed below V

_ SDN can manage the secure connection for thousands of devices connected over the fog for data transmission.

_ SDN can provide real-time monitoring and awareness with low latency.

_ SDN can dynamically balance the load with its flexible architecture. _ SDN can customize the policies and applications dues to its programmable nature. [18]

The software-denied network plays a vital role as its network control architecture can be directly programmable through the command requests. SDN based fog computing architecture can assist in analyzing and managing IoT devices. The motivation behind SDN is to give consistency to network management through partitioning the network into the data plane and the control

plane. SDN can add programmability, adaptability, and versatility to the fog computing system. In high-speed networks, discovering the botnet attack is a significant concern [19]. The proposed work shows the methodology through which the botnet attack is identified with a high detection rate which can be used in SDN to enhance the security of fog computing. Deep learning (DL) based detection approach in the SDN-based fog computing application can be a better counterattack to improve the overall performance of the system [20]. DL strategy is adaptable to conditions to recognize the abnormal behavior of the network. We proposed a hybrid deep learning detection policy to improve the efciency and effectiveness of the SDN-based fog computing architecture. Results show that the proposed scheme works better and provides a better detection rate.

Decision tree classifiers

Decision tree classifiers are used successfully in many diverse areas. Their most important feature is the capability of capturing descriptive decision making knowledge from the supplied data. Decision tree can be generated from training sets. The procedure for such generation based on the set of objects (S), each belonging to one of the classes C1, C2, …, Ck is as follows:

Step 1. If all the objects in S belong to the same class, for example Ci, the decision tree for S consists of a leaf labeled with this class

Step 2. Otherwise, let T be some test with possible outcomes O1, O2,…, On. Each object in S has one outcome for T so the test partitions S into subsets S1, S2,… Sn where each object in Si has outcome Oi for T. T becomes the root of the decision tree and for each outcome Oi we build a subsidiary decision tree by invoking the same procedure recursively on the set Si.

Gradient boosting

Gradient boosting is a machine learning technique used in regression and classification tasks, among others. It gives a prediction model in the form of an ensemble of weak prediction models, which are typically decision trees.[1][2] When a decision tree is the weak learner, the resulting algorithm is called gradient-boosted trees; it usually outperforms random forest.A gradient-boosted trees model is built in a stage-wise fashion as in other boosting methods, but it generalizes the other methods by allowing optimization of an arbitrary differentiable loss function.

K-Nearest Neighbors (KNN)

- **Simple, but a very powerful classification algorithm**

- **Classifies based on a similarity measure**
- **Non-parametric**
- **Lazy learning**
- **Does not "learn" until the test example is given**

- **Whenever we have a new data to classify, we find its K-nearest neighbors from the training data**

Example

- **Training dataset consists of k-closest examples in feature space**
- **Feature space means, space with categorization variables (non-metric variables)**
- **Learning based on instances, and thus also works lazily because instance close to the input vector for test or prediction may take time to occur in the training dataset**

Logistic regression Classifiers

Logistic regression analysis studies the association between a categorical dependent variable and a set of independent (explanatory) variables. The name *logistic regression* is used when the dependent variable has only two values, such as 0 and 1 or Yes and No. The name *multinomial logistic regression* is usually reserved for the case when the dependent variable has three or more unique values, such as Married, Single, Divorced, or Widowed. Although the type of data used for the dependent variable is different from that of multiple regression, the practical use of the procedure is similar.

Logistic regression competes with discriminant analysis as a method for analyzing categorical-response variables. Many statisticians feel that logistic regression is more versatile and better suited for modeling most situations than is discriminant analysis. This is because logistic regression does not assume that the independent variables are normally distributed, as discriminant analysis does.

This program computes binary logistic regression and multinomial logistic regression on both numeric and categorical independent variables. It reports on the regression equation as well as the goodness of fit, odds ratios, confidence limits, likelihood, and deviance. It performs a comprehensive residual analysis including diagnostic residual reports and plots. It can perform an independent variable subset selection search, looking for the best regression model with the fewest independent variables. It provides confidence intervals on predicted values and provides ROC curves to help determine the best cutoff point for classification. It allows you to validate your results by automatically classifying rows that are not used during the analysis.

Naïve Bayes

The naive bayes approach is a supervised learning method which is based on a simplistic hypothesis: it assumes that the presence (or absence) of a particular feature of a class is unrelated to the presence (or absence) of any other feature .
Yet, despite this, it appears robust and efficient. Its performance is comparable to other supervised learning techniques. Various reasons have been advanced in the literature. In this tutorial, we highlight an explanation based on the representation bias. The naive bayes classifier is a linear classifier, as well as linear discriminant

analysis, logistic regression or linear SVM (support vector machine). The difference lies on the method of estimating the parameters of the classifier (the learning bias).

While the Naive Bayes classifier is widely used in the research world, it is not widespread among practitioners which want to obtain usable results. On the one hand, the researchers found especially it is very easy to program and implement it, its parameters are easy to estimate, learning is very fast even on very large databases, its accuracy is reasonably good in comparison to the other approaches. On the other hand, the final users do not obtain a model easy to interpret and deploy, they does not understand the interest of such a technique.

Thus, we introduce in a new presentation of the results of the learning process. The classifier is easier to understand, and its deployment is also made easier. In the first part of this tutorial, we present some theoretical aspects of the naive bayes classifier. Then, we implement the approach on a dataset with Tanagra. We compare the obtained results (the parameters of the model) to those obtained with other linear approaches such as the logistic regression, the linear discriminant analysis and the linear SVM. We note that the results are highly consistent. This largely explains the good performance of the method in comparison to others. In the second part, we use various tools on the same dataset (**Weka 3.6.0**, **R 2.9.2**, **Knime 2.1.1**, **Orange 2.0b** and **RapidMiner 4.6.0)**. We try above all to understand the obtained results.

Random Forest

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time. For classification tasks, the output of the random forest is the class selected by most trees. For regression tasks, the mean or average prediction of the individual trees is returned. Random decision forests correct for decision trees' habit of overfitting to their training set. Random forests generally outperform decision trees, but their accuracy is lower than gradient boosted trees. However, data characteristics can affect their performance.

The first algorithm for random decision forests was created in 1995 by Tin Kam Ho[1] using the random subspace method, which, in Ho's formulation, is a way to implement the "stochastic discrimination" approach to classification proposed by Eugene Kleinberg.

An extension of the algorithm was developed by Leo Breiman and Adele Cutler, who registered "Random Forests" as a trademark in 2006 (as of 2019, owned by Minitab, Inc.).The extension combines Breiman's "bagging" idea and random selection of features, introduced first by Ho[1] and later independently by Amit and Geman[13] in order to construct a collection of decision trees with controlled variance.

Random forests are frequently used as "blackbox" models in businesses, as they generate reasonable predictions across a wide range of data while requiring little configuration.

SVM

In classification tasks a discriminant machine learning technique aims at finding, based on an *independent and identically distributed* (*iid*) training dataset, a discriminant function that can correctly predict labels for newly acquired instances. Unlike generative machine learning approaches, which require computations of conditional probability distributions, a discriminant classification function takes a data point *x* and assigns it to one of the different classes that are a part of the classification task. Less powerful than generative approaches, which are mostly used when prediction involves outlier detection, discriminant approaches require fewer computational resources and less training data, especially for a multidimensional feature space and when only posterior probabilities are needed. From a geometric perspective, learning a classifier is equivalent to finding the equation for a multidimensional surface that best separates the different classes in the feature space.

SVM is a discriminant technique, and, because it solves the convex optimization problem analytically, it always returns the same optimal hyperplane parameter—in contrast to *genetic algorithms* (*GAs*) or *perceptrons*, both of which are widely used for classification in machine learning. For perceptrons, solutions are highly dependent on the initialization and termination criteria. For a specific kernel that transforms the data from the input space to the feature space, training returns uniquely defined SVM model parameters for a given training set, whereas the perceptron and GA classifier models are different each time training is initialized. The aim of GAs

and perceptrons is only to minimize error during training, which will translate into several hyperplanes' meeting this requirement.

Modules

Service Provider

In this module, the Service Provider has to login by using valid user name and password. After login successful he can do some operations such as Login, Train & Test Data Sets, View Trained and Tested Accuracy in Bar Chart, View Trained and Tested Accuracy Results, View Prediction Of Attack Type, View Attack Type Ratio, Download Predicted Data Sets, View Attack Type Ratio Results, View All Remote Users.

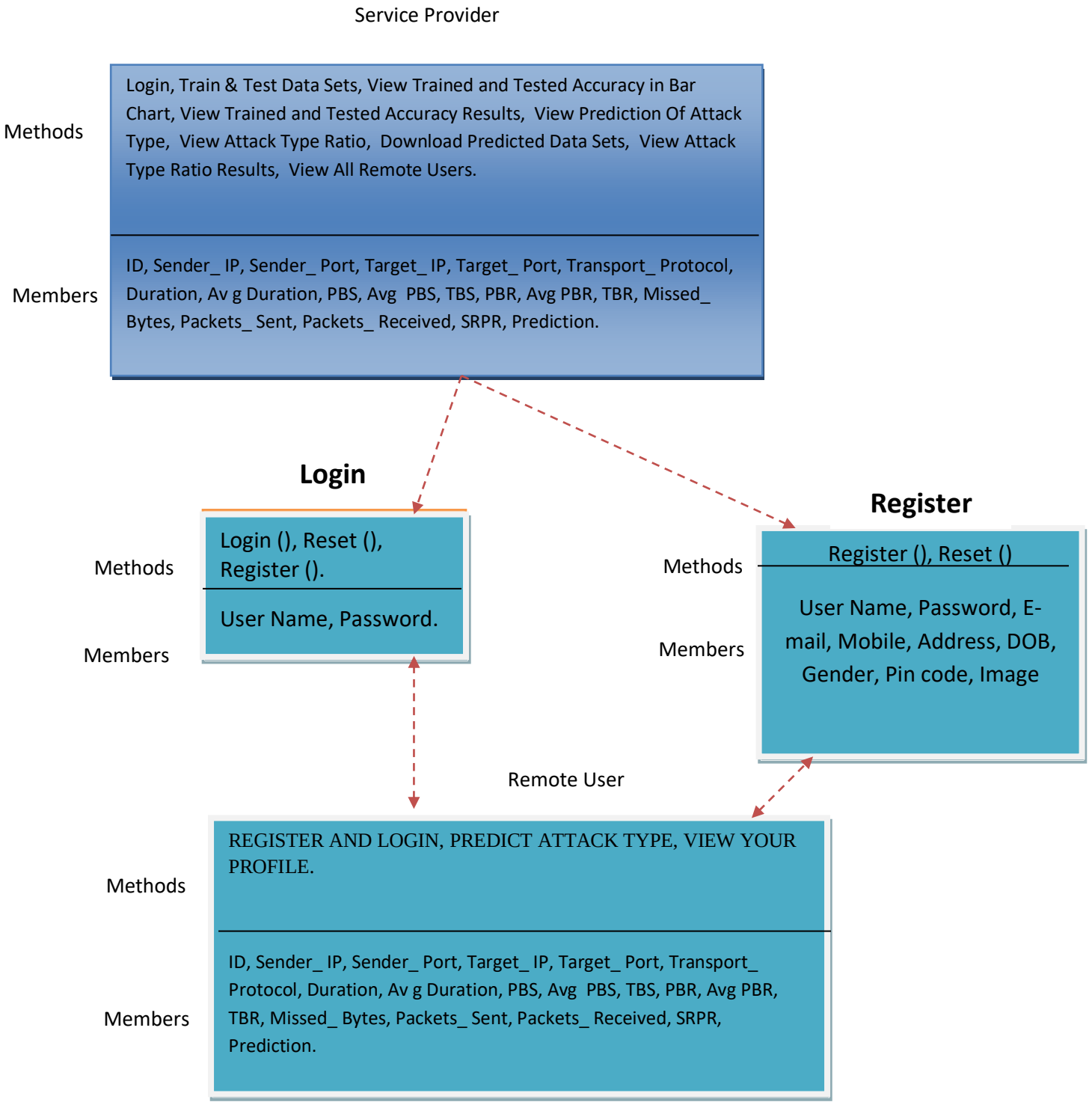
View and Authorize Users

In this module, the admin can view the list of users who all registered. In this, the admin can view the user's details such as, user name, email, address and admin authorizes the users.

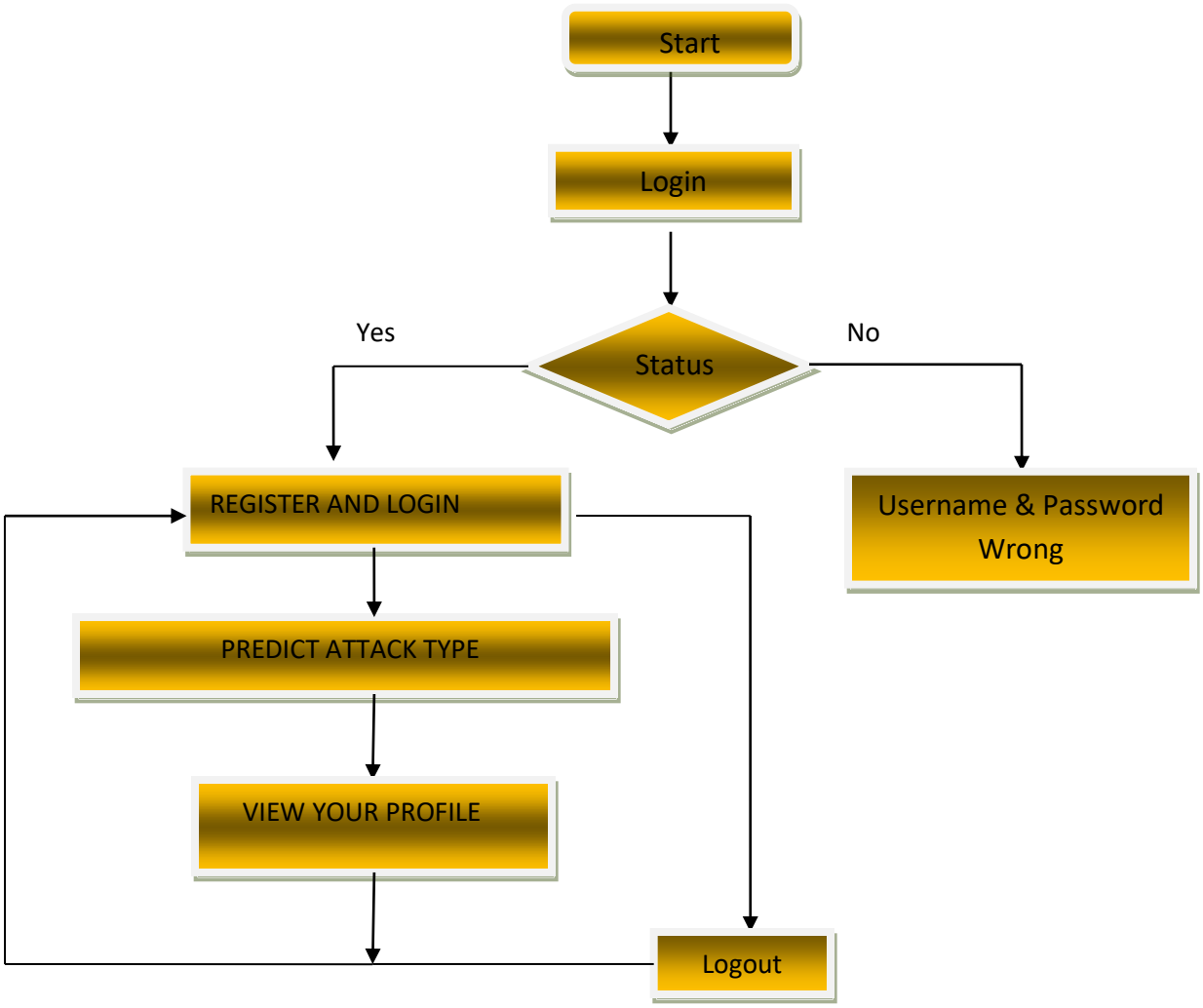
Remote User

In this module, there are n numbers of users are present. User should register before doing any operations. Once user registers, their details will be stored to the database. After registration successful, he has to login by using authorized user name and password. Once Login is successful user will do some operations like REGISTER AND LOGIN, PREDICT ATTACK TYPE, VIEW YOUR PROFILE.

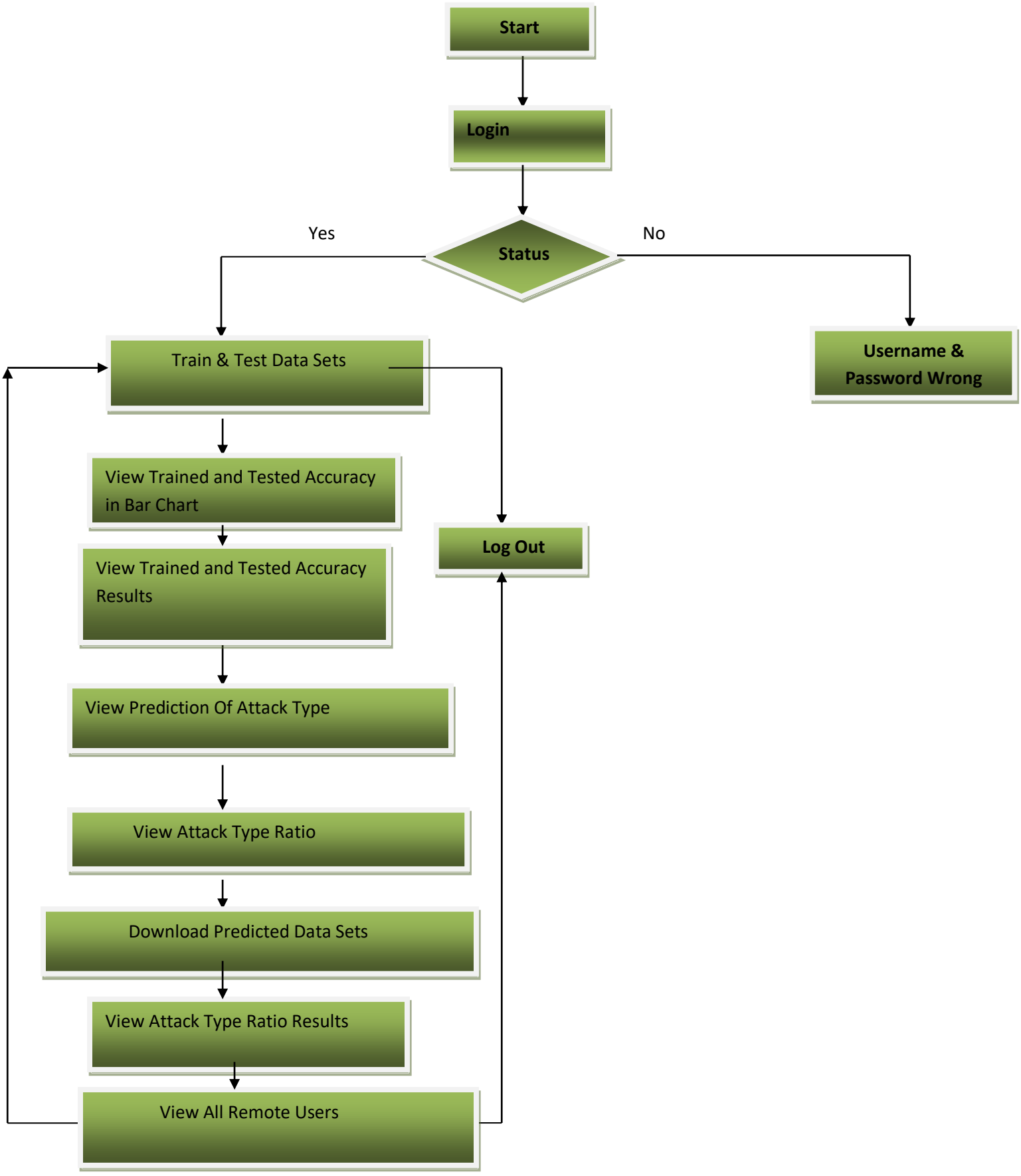
➢ Class Diagram :



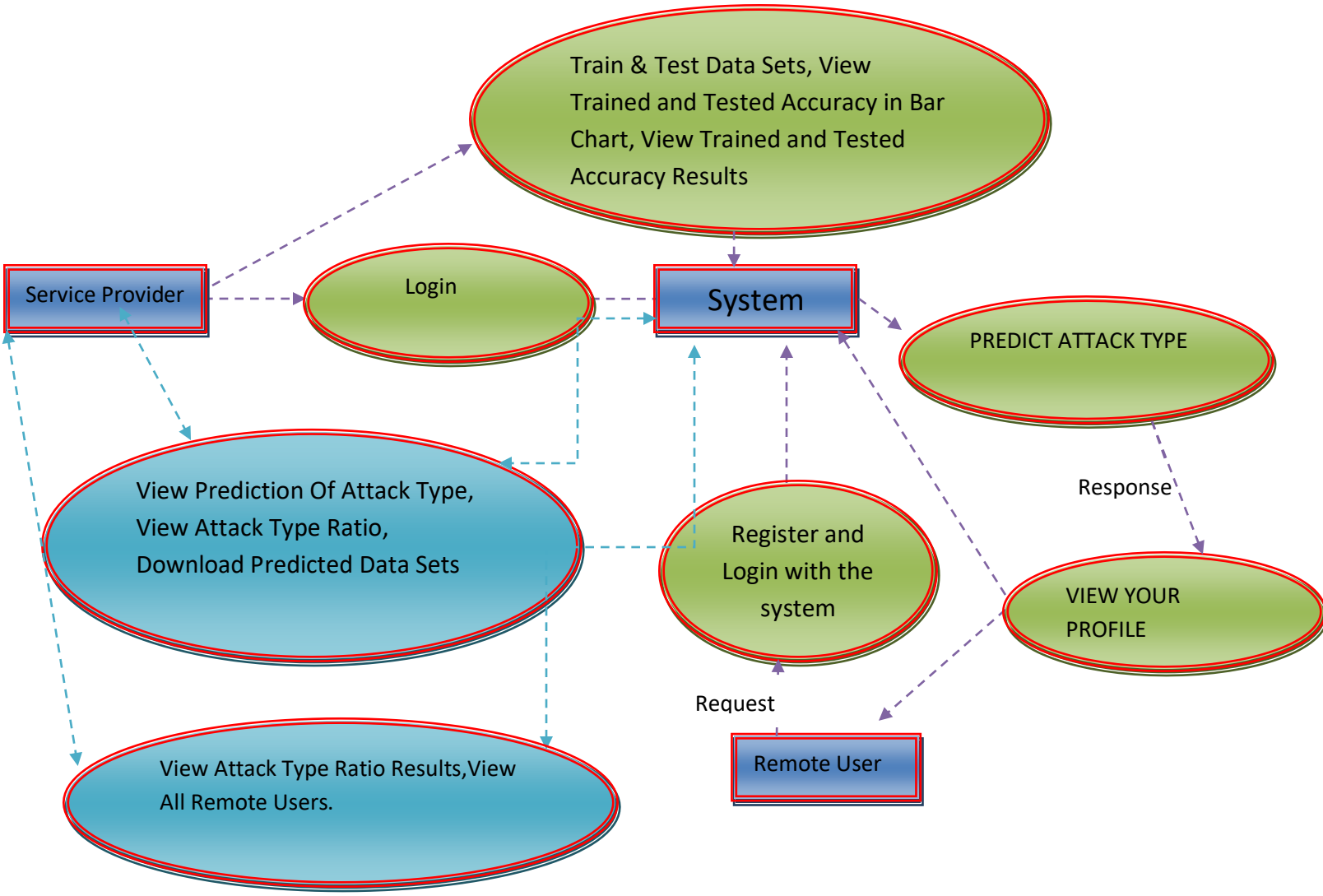
➢ Flow Chart : Remote User



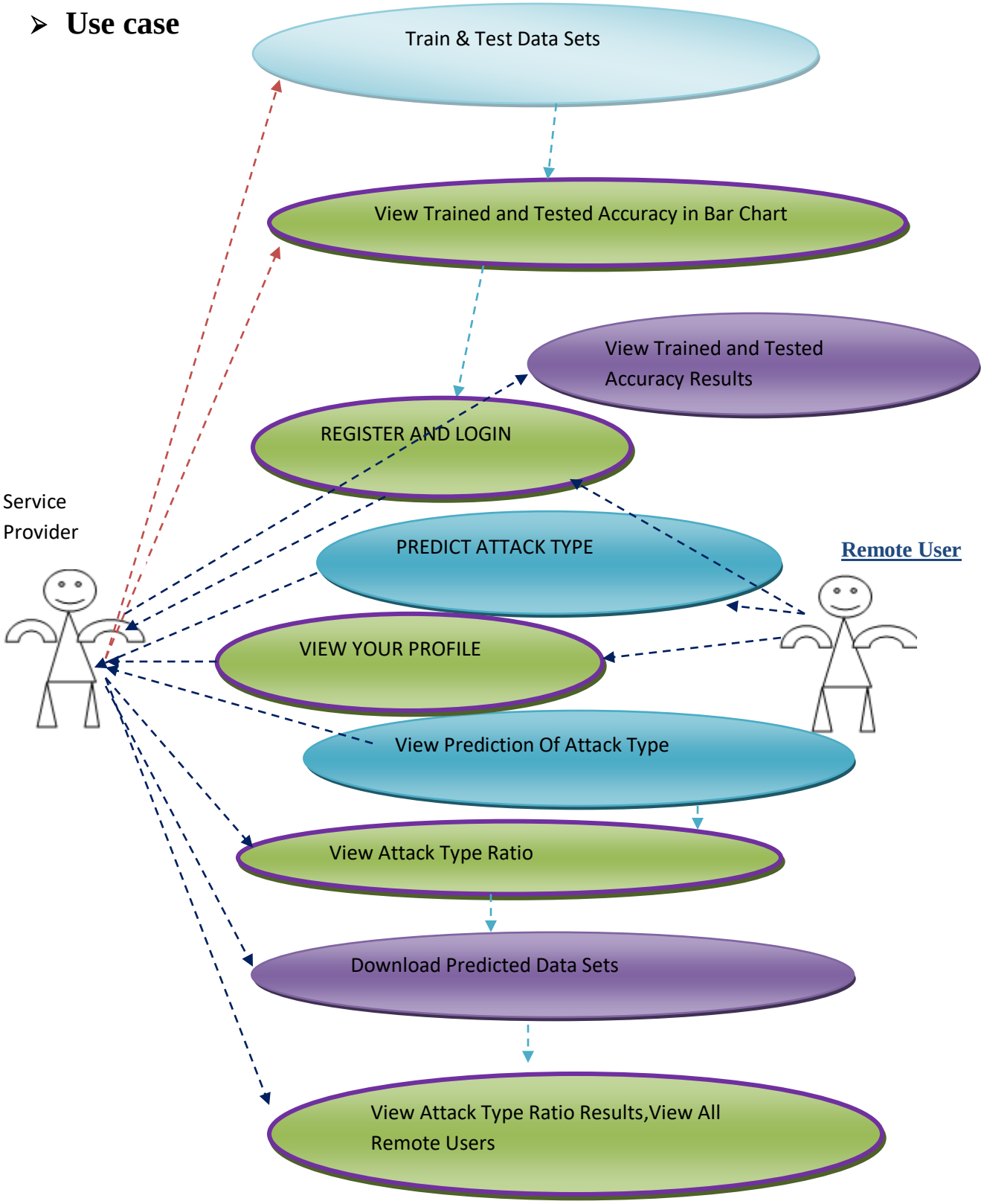
➢ Flow Chart : Service Provider



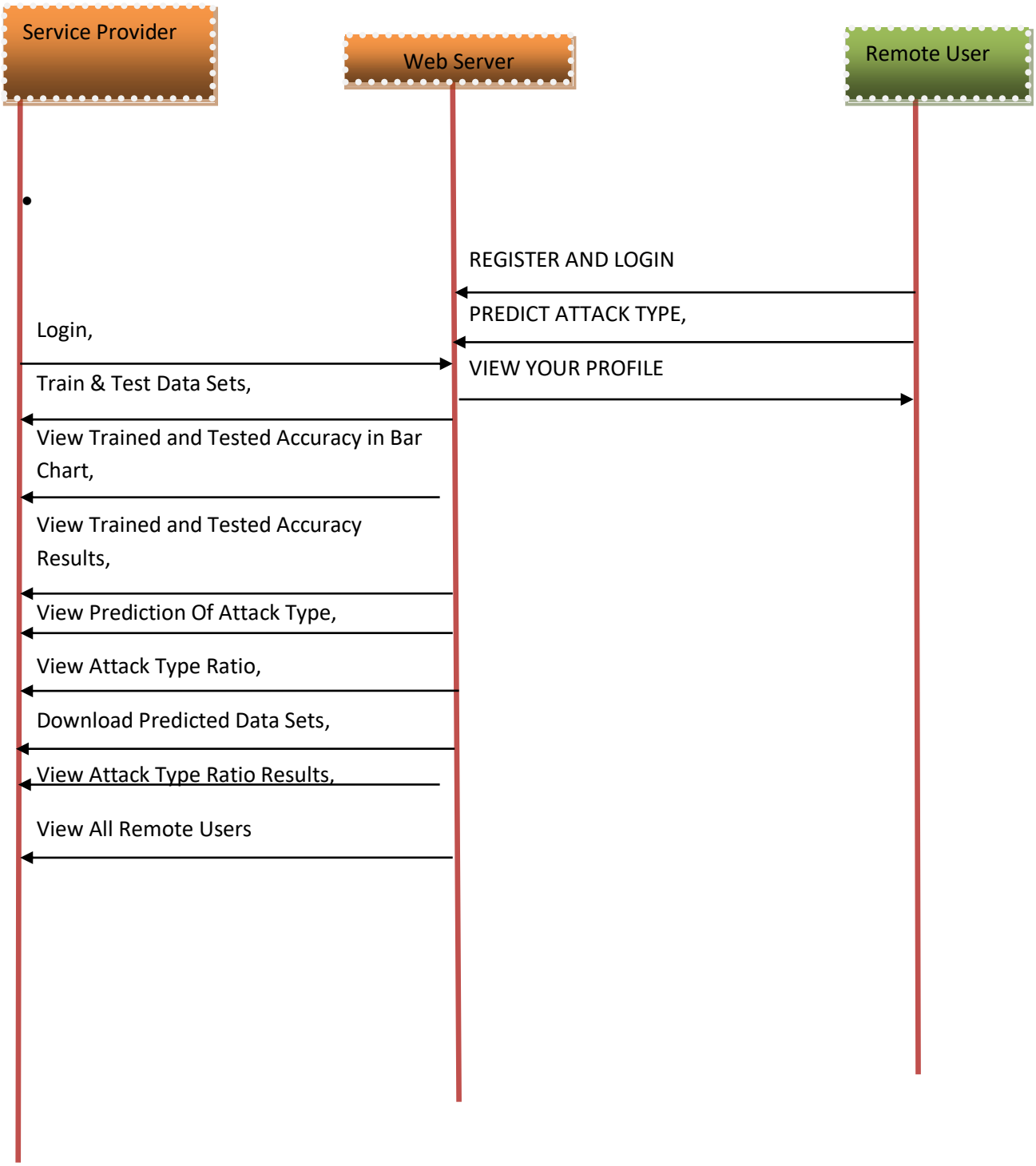
➢ **Data Flow Diagram** :



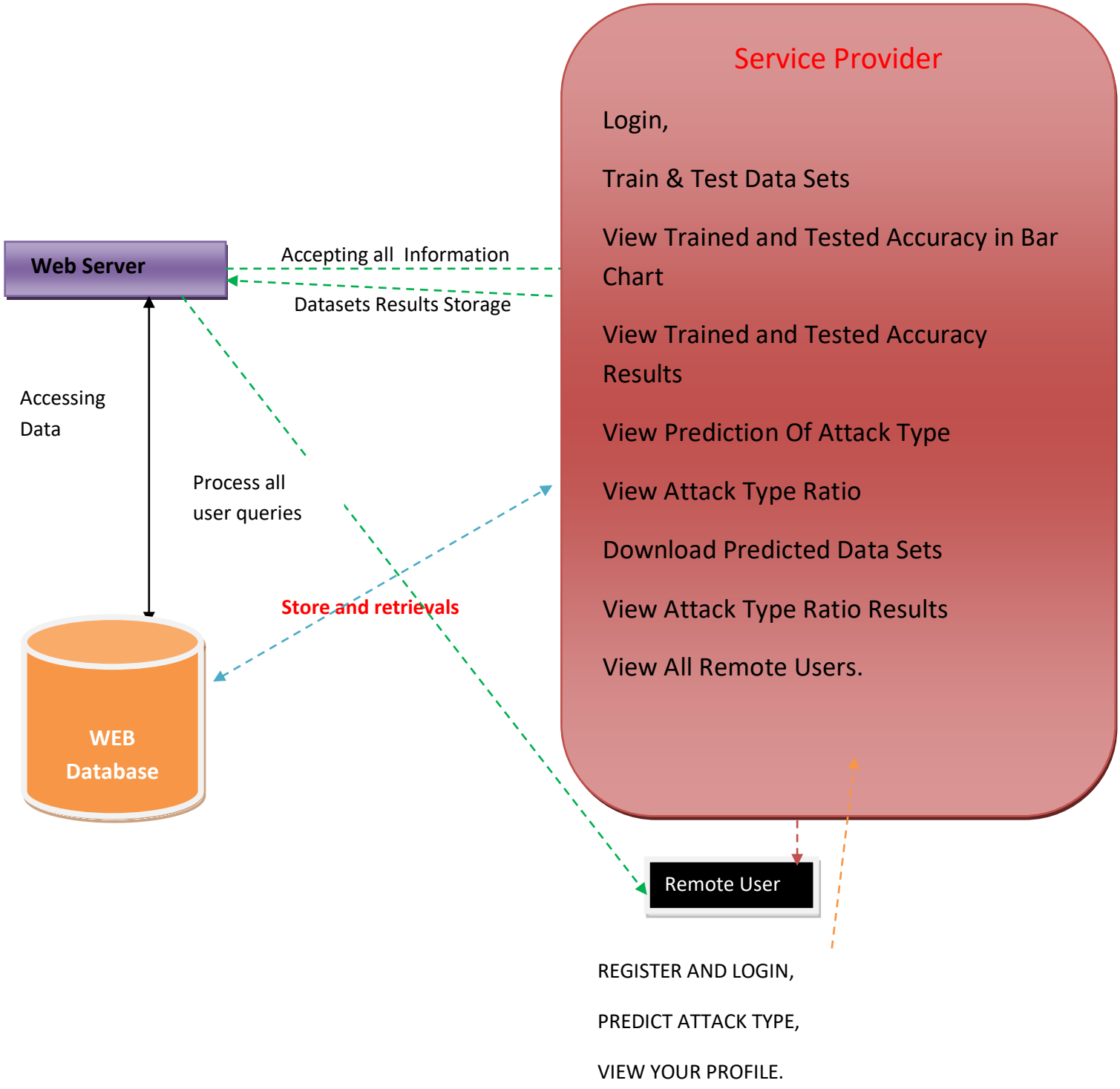
➢ Use case


Train & Test Data Sets
View Trained and Tested Accuracy in Bar Chart
View Trained and Tested Accuracy Results
REGISTER AND LOGIN
Service Provider
PREDICT ATTACK TYPE
Remote User
VIEW YOUR PROFILE
View Prediction Of Attack Type
View Attack Type Ratio
Download Predicted Data Sets
View Attack Type Ratio Results,View All Remote Users

➢ Sequence Diagram



Architecture Diagram



6. SYSTEM TESTING

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the

Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

TYPES OF TESTS

Unit testing

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

Integration testing

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

Functional test

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

Valid Input : identified classes of valid input must be accepted.

Invalid Input : identified classes of invalid input must be rejected.

Functions : identified functions must be exercised.

Output : identified classes of application outputs must be exercised.

Systems/Procedures: interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for

testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

System Test

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

White Box Testing

White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

Black Box Testing

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box .you cannot “see” into it. The test provides inputs and responds to outputs without considering how the software works.

6.1 Unit Testing:

Unit testing is usually conducted as part of a combined code and unit test phase of the software lifecycle, although it is not uncommon for coding and unit testing to be conducted as two distinct phases.

Test strategy and approach

Field testing will be performed manually and functional tests will be written in detail.

Test objectives

- All field entries must work properly.
- Pages must be activated from the identified link.
- The entry screen, messages and responses must not be delayed.

Features to be tested

- Verify that the entries are of the correct format
- No duplicate entries should be allowed
- All links should take the user to the correct page.

6.2 Integration Testing

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at the company level – interact without error.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

6.3 Acceptance Testing

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

SYSTEM TESTING

TESTING METHODOLOGIES

The following are the Testing Methodologies:

- **Unit Testing.**
- **Integration Testing.**
- **User Acceptance Testing.**
- **Output Testing.**
- **Validation Testing.**

Unit Testing

Unit testing focuses verification effort on the smallest unit of Software design that is the module. Unit testing exercises specific paths in a module's control structure to ensure complete coverage and maximum error detection. This test focuses on each module individually, ensuring that it functions properly as a unit. Hence, the naming is Unit Testing.

During this testing, each module is tested individually and the module interfaces are verified for the consistency with design specification. All important processing path are tested for the expected results. All error handling paths are also tested.

Integration Testing

Integration testing addresses the issues associated with the dual problems of verification and program construction. After the software has been integrated a set of high order tests are conducted. The main objective in this testing process is to take unit tested modules and builds a program structure that has been dictated by design.

The following are the types of Integration Testing:

1)Top Down Integration

This method is an incremental approach to the construction of program structure. Modules are integrated by moving downward through the control hierarchy, beginning with the main program module. The module subordinates to the main program module are incorporated into the structure in either a depth first or breadth first manner.

In this method, the software is tested from main module and individual stubs are replaced when the test proceeds downwards.

2. Bottom-up Integration

This method begins the construction and testing with the modules at the lowest level in the program structure. Since the modules are integrated from the bottom up, processing required for modules subordinate to a given level is always available and the need for stubs is eliminated. The bottom up integration strategy may be implemented with the following steps:

- The low-level modules are combined into clusters into clusters that perform a specific Software sub-function.
- A driver (i.e.) the control program for testing is written to coordinate test case input and output.
- The cluster is tested.
- Drivers are removed and clusters are combined moving upward in the program structure

The bottom up approaches tests each module individually and then each module is module is integrated with a main module and tested for functionality.

OTHER TESTING METHODOLOGIES

User Acceptance Testing

User Acceptance of a system is the key factor for the success of any system. The system under consideration is tested for user acceptance by constantly keeping in touch with the prospective system users at the time of developing and making changes wherever required. The system developed provides a friendly user interface that can easily be understood even by a person who is new to the system.

Output Testing

After performing the validation testing, the next step is output testing of the proposed system, since no system could be useful if it does not produce the required output in the specified format. Asking the users about the format required by them tests the outputs generated or displayed by the system under consideration. Hence the output format is considered in 2 ways – one is on screen and another in printed format.

Validation Checking

Validation checks are performed on the following fields.

Text Field:

The text field can contain only the number of characters lesser than or equal to its size. The text fields are alphanumeric in some tables and alphabetic in other tables. Incorrect entry always flashes and error message.

Numeric Field:

The numeric field can contain only numbers from 0 to 9. An entry of any character flashes an error messages. The individual modules are checked for accuracy and what it has to perform. Each module is subjected to test run along with sample data. The individually tested modules are integrated into a single system. Testing involves executing the real data information is used in the program the existence of any program defect is inferred from the output. The testing should be planned so that all the requirements are individually tested.

A successful test is one that gives out the defects for the inappropriate data and produces and output revealing the errors in the system.

Preparation of Test Data

Taking various kinds of test data does the above testing. Preparation of test data plays a vital role in the system testing. After preparing the test data the system under study is tested using that test data. While testing the system by using test data errors are again uncovered and corrected by using above testing steps and corrections are also noted for future use.

Using Live Test Data:

Live test data are those that are actually extracted from organization files. After a system is partially constructed, programmers or analysts often ask users to key in a set of data from their normal activities. Then, the systems person uses this data as a way to partially test the system. In other instances, programmers or analysts extract a set of live data from the files and have them entered themselves.

It is difficult to obtain live data in sufficient amounts to conduct extensive testing. And, although it is realistic data that will show how the system will perform for the typical processing requirement, assuming that the live data entered are in fact

typical, such data generally will not test all combinations or formats that can enter the system. This bias toward typical values then does not provide a true systems test and in fact ignores the cases most likely to cause system failure.

Using Artificial Test Data:

Artificial test data are created solely for test purposes, since they can be generated to test all combinations of formats and values. In other words, the artificial data, which can quickly be prepared by a data generating utility program in the information systems department, make possible the testing of all login and control paths through the program.

The most effective test programs use artificial test data generated by persons other than those who wrote the programs. Often, an independent team of testers formulates a testing plan, using the systems specifications.

The package "Virtual Private Network" has satisfied all the requirements specified as per software requirement specification and was accepted.

USER TRAINING

Whenever a new system is developed, user training is required to educate them about the working of the system so that it can be put to efficient use by those for whom the system has been primarily designed. For this purpose the normal working of the project was demonstrated to the prospective users. Its working is easily understandable and since the expected users are people who have good knowledge of computers, the use of this system is very easy.

MAINTAINENCE

This covers a wide range of activities including correcting code and design errors. To reduce the need for maintenance in the long run, we have more accurately defined the user's requirements during the process of system development. Depending on the requirements, this system has been developed to satisfy the needs to the largest possible extent. With development in technology, it may be possible to add many more features based on the requirements in future. The coding and designing is simple and easy to understand which will make maintenance easier.

TESTING STRATEGY :

A strategy for system testing integrates system test cases and design techniques into a well planned series of steps that results in the successful construction of software. The testing strategy must co-operate test planning, test case design, test execution, and the resultant data collection and evaluation .A strategy for software testing must accommodate low-level tests that are necessary to verify that a small source code segment has been correctly implemented as well as high level tests that validate major system functions against user requirements.

Software testing is a critical element of software quality assurance and represents the ultimate review of specification design and coding. Testing represents an interesting anomaly for the software. Thus, a series of testing are performed for the proposed system before the system is ready for user acceptance testing.

SYSTEM TESTING:

Software once validated must be combined with other system elements (e.g. Hardware, people, database). System testing verifies that all the elements are proper and that overall system function performance is achieved. It also tests to find discrepancies between the system and its original objective, current specifications and system documentation.

UNIT TESTING:

In unit testing different are modules are tested against the specifications produced during the design for the modules. Unit testing is essential for verification of the code produced during the coding phase, and hence the goals to test the internal logic of the modules. Using the detailed design description as a guide, important Conrail paths are tested to uncover errors within the boundary of the modules. This testing is carried out during the programming stage itself. In this type of testing step, each module was found to be working satisfactorily as regards to the expected output from the module.

In Due Course, latest technology advancements will be taken into consideration. As part of technical build-up many components of the networking system will be generic in nature so that future projects can either use or interact with this. The future holds a lot to offer to the development and refinement of this project.

1.1 PYTHON

Python is a **high-level, interpreted**, **interactive** and **object-oriented scripting language**. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

- **Python is Interpreted:** Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- **Python is Interactive:** You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- **Python is Object-Oriented:** Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- **Python is a Beginner's Language:** Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

1.2 History of Python

Python was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands.

Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, SmallTalk, and Unix shell and other scripting languages.

Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL).

Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress.

1.3 Python Features

Python's features include:

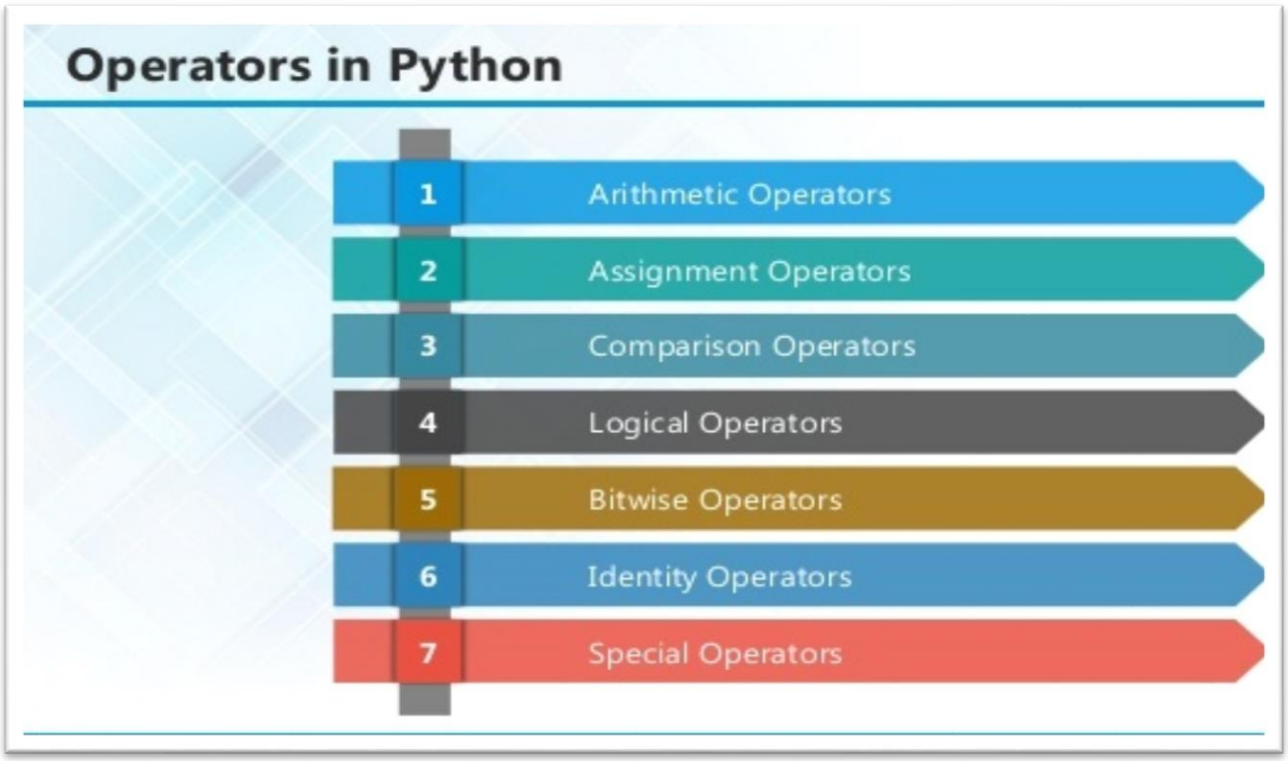
- **Easy-to-learn:** Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
- **Easy-to-read:** Python code is more clearly defined and visible to the eyes.
- **Easy-to-maintain:** Python's source code is fairly easy-to-maintain.
- **A broad standard library:** Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.

- **Interactive Mode:** Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
- **Portable:** Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- **Extendable:** You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- **Databases:** Python provides interfaces to all major commercial databases.
- **GUI Programming:** Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
- **Scalable:** Python provides a better structure and support for large programs than shell scripting.

Python has a big list of good features:

- It supports functional and structured programming methods as well as OOP.

- It can be used as a scripting language or can be compiled to byte-code for building large applications.
- It provides very high-level dynamic data types and supports dynamic type checking.
- IT supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.



2.1 ARITHMETIC OPERATORS

Operator	Description	Example

+ Addition	Adds values on either side of the operator.	a + b = 30
- Subtraction	Subtracts right hand operand from left hand operand.	a – b = -10
* Multiplication	Multiplies values on either side of the operator	a * b = 200
/ Division	Divides left hand operand by right hand operand	b / a = 2
% Modulus	Divides left hand operand by right hand operand and returns remainder	b % a = 0
** Exponent	Performs exponential (power) calculation on operators	a**b =10 to the power 20
//	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed. But if one of the operands is negative, the result is floored, i.e., rounded away from zero (towards negative infinity):	9//2 = 4 and 9.0//2.0 = 4.0, -11//3 = -4, -11.0//3 = -4.0

2.2ASSIGNMENT OPERATOR

Operator	Description	Example

=	Assigns values from right side operands to left side operand	c = a + b assigns value of a + b into c
+= Add AND	It adds right operand to the left operand and assign the result to left operand	c += a is equivalent to c = c + a
-= Subtract AND	It subtracts right operand from the left operand and assign the result to left operand	c -= a is equivalent to c = c - a
*= Multiply AND	It multiplies right operand with the left operand and assign the result to left operand	c *= a is equivalent to c = c * a
/= Divide AND	It divides left operand with the right operand and assign the result to left operand	c /= a is equivalent to c = c / ac /= a is equivalent to c = c / a

%= Modulus AND	It takes modulus using two operands and assign the result to left operand	c %= a is equivalent to c = c % a

**= Exponent AND	Performs exponential (power) calculation on operators and assign value to the left operand	c **= a is equivalent to c = c ** a
//= Floor Division	It performs floor division on operators and assign value to the left operand	c //= a is equivalent to c = c // a

2.3 IDENTITY OPERATOR

Operator	Description	Example
is	Evaluates to true if the variables on either side of the operator point to the same object and false otherwise.	x is y, here **is** results in 1 if id(x) equals id(y).
is not	Evaluates to false if the variables on either side of the operator point to the same object and true otherwise.	x is not y, here **is not** results in 1 if id(x) is not equal to id(y

2.4 COMPARISON OPERATOR

Operator	Description	Example

& Binary AND	Operator copies a bit to the result if it exists in both operands	(a & b) (means 0000 1100)
| Binary OR	It copies a bit if it exists in either operand.	(a | b) = 61 (means 0011 1101)
^ Binary XOR	It copies the bit if it is set in one operand but not both.	(a ^ b) = 49 (means 0011 0001)
~ Binary Ones Complement	It is unary and has the effect of 'flipping' bits.	(~a) = -61 (means 1100 0011 in 2's complement form due to a signed binary number.
<< Binary Left Shift	The left operands value is moved left by the number of bits specified by the right operand.	a << 2 = 240 (means 1111 0000)
>> Binary Right Shift	The left operands value is moved right by the number of bits specified by the right operand.	a >> 2 = 15 (means 0000 1111)

2.5 LOGICAL OPERATOR

Operator	Description	Example
and Logical AND	If both the operands are true then condition becomes true.	(a and b) is true.
or Logical OR	If any of the two operands are non-zero then condition becomes true.	(a or b) is true.
not Logical NOT	Used to reverse the logical state of its operand.	Not(a and b) is false.

2.6 Membership Operators

Operator	Description	Example
in	Evaluates to true if it finds a variable in the specified sequence and false otherwise.	x in y, here in results in a 1 if x is a member of sequence y.
not in	Evaluates to true if it does not finds a variable in the specified sequence and false otherwise.	x not in y, here not in results in a 1 if x is not a member of sequence y.

Python Operators Precedence

Operator	Description

**	Exponentiation (raise to the power)
~ + -	Complement, unary plus and minus (method names for the last two are +@ and -@)
* / % //	Multiply, divide, modulo and floor division
+ -	Addition and subtraction
>> <<	Right and left bitwise shift
&	Bitwise 'AND'
^ |	Bitwise exclusive `OR' and regular `OR'
<= < > >=	Comparison operators
<> == !=	Equality operators
= %= /= //= -= += *= **=	Assignment operators
is is not	Identity operators
in not in	Membership operators
not or and	Logical operators

3.1 LIST

The list is a most versatile data type available in Python which can be written as a list of comma-separated values (items) between square brackets. Important thing about a list is that items in a list need not be of the same type.

Creating a list is as simple as putting different comma-separated values between square brackets. For example −

```
list1 = ['physics', 'chemistry', 1997, 2000];
list2 = [1, 2, 3, 4, 5 ];
list3 = ["a", "b", "c", "d"]
```

Basic List Operations

Lists respond to the + and * operators much like strings; they mean concatenation and repetition here too, except that the result is a new list, not a string.

Python Expression	Results	Description
len([1, 2, 3])	3	Length
[1, 2, 3] + [4, 5, 6]	[1, 2, 3, 4, 5, 6]	Concatenation
['Hi!'] * 4	['Hi!', 'Hi!', 'Hi!', 'Hi!']	Repetition
3 in [1, 2, 3]	True	Membership
for x in [1, 2, 3]: print x,	1 2 3	Iteration

Built-in List Functions & Methods:

Python includes the following list functions −

SN	Function with Description
1	cmp(list1, list2) Compares elements of both lists.
2	len(list) Gives the total length of the list.
3	max(list) Returns item from the list with max value.
4	min(list) Returns item from the list with min value.
5	list(seq) Converts a tuple into list.

Python includes following list methods

SN	Methods with Description
1	list.append(obj) Appends object obj to list
2	list.count(obj) Returns count of how many times obj occurs in list

3	list. extend(seq) Appends the contents of seq to list
4	list.index(obj) Returns the lowest index in list that obj appears
5	list.insert(index, obj) Inserts object obj into list at offset index
6	list.pop(obj=list[-1]) Removes and returns last object or obj from list
7	list.remove(obj) Removes object obj from list
8	list.reverse() Reverses objects of list in place
9	list.sort([func]) Sorts objects of list, use compare function if given

3.2 TUPLES

A tuple is a sequence of immutable Python objects. Tuples are sequences, just like lists. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.

Creating a tuple is as simple as putting different comma-separated values. Optionally we can put these comma-separated values between parentheses also. For example −

```
tup1 = ('physics', 'chemistry', 1997, 2000);
```

```
tup2 = (1, 2, 3, 4, 5 );

tup3 = "a", "b", "c", "d";
```

The empty tuple is written as two parentheses containing nothing −

```
tup1 = ();
```

To write a tuple containing a single value you have to include a comma, even though there is only one value −

```
tup1 = (50,);
```

Like string indices, tuple indices start at 0, and they can be sliced, concatenated, and so on.

- **Accessing Values in Tuples:**

To access values in tuple, use the square brackets for slicing along with the index or indices to obtain value available at that index. For example –

```
tup1 = ('physics', 'chemistry', 1997, 2000);

tup2 = (1, 2, 3, 4, 5, 6, 7 );

print "tup1[0]: ", tup1[0]

print "tup2[1:5]: ", tup2[1:5]
```

When the code is executed, it produces the following result −

```
tup1[0]:  physics
tup2[1:5]:  [2, 3, 4, 5]
```

Updating Tuples:

Tuples are immutable which means you cannot update or change the values of tuple elements. We are able to take portions of existing tuples to create new tuples as the following example demonstrates −

```
tup1 = (12, 34.56);

tup2 = ('abc', 'xyz');
```

```
tup3 = tup1 + tup2;
print tup3
```

When the above code is executed, it produces the following result −

```
(12, 34.56, 'abc', 'xyz')
```

Delete Tuple Elements

Removing individual tuple elements is not possible. There is, of course, nothing wrong with putting together another tuple with the undesired elements discarded.

To explicitly remove an entire tuple, just use the **del** statement. For example:

```
tup = ('physics', 'chemistry', 1997, 2000);
print tup
del tup;
print "After deleting tup : "
print tup
```

Basic Tuples Operations:

Python Expression	Results	Description
len((1, 2, 3))	3	Length
(1, 2, 3) + (4, 5, 6)	(1, 2, 3, 4, 5, 6)	Concatenation
('Hi!',) * 4	('Hi!', 'Hi!', 'Hi!', 'Hi!')	Repetition

3 in (1, 2, 3)	True	Membership
for x in (1, 2, 3): print x,	1 2 3	Iteration

Built-in Tuple Functions

SN	Function with Description
1	**cmp(tuple1, tuple2)**:Compares elements of both tuples.
2	**len(tuple)**:Gives the total length of the tuple.
3	**max(tuple)**:Returns item from the tuple with max value.
4	**min(tuple)**:Returns item from the tuple with min value.
5	**tuple(seq)**:Converts a list into tuple.

3.2 DICTIONARY

Each key is separated from its value by a colon (:), the items are separated by commas, and the whole thing is enclosed in curly braces. An empty dictionary without any items is written with just two curly braces, like this: {}.

Keys are unique within a dictionary while values may not be. The values of a dictionary can be of any type, but the keys must be of an immutable data type such as strings, numbers, or tuples.

Accessing Values in Dictionary:

To access dictionary elements, you can use the familiar square brackets along with the key to obtain its value. Following is a simple example −

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}


print "dict['Name']: ", dict['Name']
print "dict['Age']: ", dict['Age']
```

Result –

```
dict['Name']:  Zara
dict['Age']:  7
```

Updating Dictionary

We can update a dictionary by adding a new entry or a key-value pair, modifying an existing entry, or deleting an existing entry as shown below in the simple example −

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}


dict['Age'] = 8; # update existing entry
dict['School'] = "DPS School"; # Add new entry
print "dict['Age']: ", dict['Age']
print "dict['School']: ", dict['School']
```

Result −

```
dict['Age']:  8
dict['School']:  DPS School
```

Delete Dictionary Elements

We can either remove individual dictionary elements or clear the entire contents of a dictionary. You can also delete entire dictionary in a single operation.

To explicitly remove an entire dictionary, just use the **del** statement. Following is a simple example –

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}


del dict['Name']; # remove entry with key 'Name'
dict.clear();     # remove all entries in dict
del dict ;        # delete entire dictionary


print "dict['Age']: ", dict['Age']
print "dict['School']: ", dict['School']
```

Built-in Dictionary Functions & Methods –

Python includes the following dictionary functions −

SN	Function with Description
1	cmp(dict1, dict2) Compares elements of both dict.

2	len(dict) Gives the total length of the dictionary. This would be equal to the number of items in the dictionary.
3	str(dict) Produces a printable string representation of a dictionary
4	type(variable) Returns the type of the passed variable. If passed variable is dictionary, then it would return a dictionary type.

Python includes following dictionary methods −

SN	Methods with Description
1	**dict.clear()**:Removes all elements of dictionary *dict*
2	**dict. Copy()**:Returns a shallow copy of dictionary *dict*
3	**dict.fromkeys()**:Create a new dictionary with keys from seq and values *set* to *value*.
4	**dict.get(key, default=None)**:For *key* key, returns value or default if key not in dictionary
5	**dict.has_key(key)**:Returns *true* if key in dictionary *dict*, *false* otherwise

6	**dict.items()**:Returns a list of *dict*'s (key, value) tuple pairs
7	**dict.keys()**:Returns list of dictionary dict's keys
8	**dict.setdefault(key, default=None)**:Similar to get(), but will set dict[key]=default if *key* is not already in dict
9	**dict.update(dict2)**:Adds dictionary *dict2*'s key-values pairs to *dict*
10	**dict.values()**:Returns list of dictionary *dict*'s values

A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing. Python gives you many built-in functions like print(), etc. but you can also create your own functions. These functions are called *user-defined functions.*

Defining a Function

Simple rules to define a function in Python.

- Function blocks begin with the keyword def followed by the function name and parentheses (( )).
- Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.
- The first statement of a function can be an optional statement - the documentation string of the function or *docstring*.
- The code block within every function starts with a colon (:) and is indented.
- The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

```
def functionname( parameters ):
   "function_docstring"
   function_suite
   return [expression]
```

Calling a Function

Defining a function only gives it a name, specifies the parameters that are to be included in the function and structures the blocks of code.Once the basic structure of a function is finalized, you can execute it by calling it from another function or directly from the Python prompt. Following is the example to call printme() function −

```
# Function definition is here
def printme( str ):
   "This prints a passed string into this function"
   print str
   return;
# Now you can call printme function
printme("I'm first call to user defined function!")
printme("Again second call to the same function")
```

When the above code is executed, it produces the following result −

```
I'm first call to user defined function!
Again second call to the same function
```

Function Arguments

You can call a function by using the following types of formal arguments:

- Required arguments
- Keyword arguments
- Default arguments
- Variable-length arguments

Scope of Variables

All variables in a program may not be accessible at all locations in that program. This depends on where you have declared a variable.

The scope of a variable determines the portion of the program where you can access a particular identifier. There are two basic scopes of variables in Python −

Global variables Local variables

Global vs. Local variables

Variables that are defined inside a function body have a local scope, and those defined outside have a global scope.

This means that local variables can be accessed only inside the function in which they are declared, whereas global variables can be accessed throughout the program body by all functions. When you call a function, the variables declared inside it are brought into scope. Following is a simple example −

```
total = 0; # This is global variable.
# Function definition is here
def sum( arg1, arg2 ):
   # Add both the parameters and return them."
   total = arg1 + arg2; # Here total is local variable.
   print "Inside the function local total : ", total
```

```
   return total;
sum( 10, 20 );
print "Outside the function global total : ", total
```

Result −

```
Inside the function local total :  30
Outside the function global total :  0
```

A module allows you to logically organize your Python code. Grouping related code into a module makes the code easier to understand and use. A module is a Python object with arbitrarily named attributes that you can bind and reference.Simply, a module is a file consisting of Python code. A module can define functions, classes and variables. A module can also include runnable code.

Example:

The Python code for a module named *aname* normally resides in a file named *aname.py*. Here's an example of a simple module, support.py

```
def print_func( par ):
   print "Hello : ", par
   return
```

The *import* Statement

The *import* has the following syntax:

```
import module1[, module2[,... moduleN]
```

When the interpreter encounters an import statement, it imports the module if the module is present in the search path. A search path is a list of directories that the interpreter searches

before importing a module. For example, to import the module support.py, you need to put the following command at the top of the script −

A module is loaded only once, regardless of the number of times it is imported. This prevents the module execution from happening over and over again if multiple imports occur.

Packages in Python

A package is a hierarchical file directory structure that defines a single Python application environment that consists of modules and sub packages and sub-sub packages.

Consider a file *Pots.py* available in *Phone* directory. This file has following line of source code −

```
def Pots():
   print "I'm Pots Phone"
```

Similar way, we have another two files having different functions with the same name as above −

- *Phone/Isdn.py* file having function Isdn()
- *Phone/G3.py* file having function G3()

Now, create one more file __init__.py in *Phone* directory −

- Phone/__init__.py

To make all of your functions available when you've imported Phone,to put explicit import statements in __init__.py as follows −

```
from Pots import Pots
from Isdn import Isdn
from G3 import G3
```

After you add these lines to __init__.py, you have all of these classes available when you import the Phone package.

```
# Now import your Phone Package.
import Phone
Phone.Pots()
Phone.Isdn()
Phone.G3()
```

RESULT:

```
I'm Pots Phone
I'm 3G Phone
I'm ISDN Phone
```

In the above example, we have taken example of a single functions in each file, but you can keep multiple functions in your files. You can also define different Python classes in those files and then you can create your packages out of those classes.

This chapter covers all the basic I/O functions available in Python.

Printing to the Screen

The simplest way to produce output is using the *print* statement where you can pass zero or more expressions separated by commas. This function converts the expressions you pass into a string and writes the result to standard output as follows −

```
print "Python is really a great language,", "isn't it?"
```

Result:

```
Python is really a great language, isn't it?
```

Reading Keyboard Input

Python provides two built-in functions to read a line of text from standard input, which by default comes from the keyboard. These functions are −

- raw_input
- input

The *raw_input* Function

The *raw_input([prompt])* function reads one line from standard input and returns it as a string (removing the trailing newline).

```
str = raw_input("Enter your input: ");
print "Received input is : ", str
```

This prompts you to enter any string and it would display same string on the screen. When I typed "Hello Python!", its output is like this −

```
Enter your input: Hello Python
Received input is :  Hello Python
```

The *input* Function

The *input([prompt])* function is equivalent to raw_input, except that it assumes the input is a valid Python expression and returns the evaluated result to you.

```
str = input("Enter your input: ");
print "Received input is : ", str
```

This would produce the following result against the entered input −

```
Enter your input: [x*5 for x in range(2,10,2)]
Recieved input is :  [10, 20, 30, 40]
```

Opening and Closing Files

Until now, you have been reading and writing to the standard input and output. Now, we will see how to use actual data files.

Python provides basic functions and methods necessary to manipulate files by default. You can do most of the file manipulation using a **file** object.

The *open* Function

Before you can read or write a file, you have to open it using Python's built-in *open()* function. This function creates a **file** object, which would be utilized to call other support methods associated with it.

Syntax

```
file object = open(file_name [, access_mode][, buffering])
```

Here are parameter details:

- **file_name:** The file_name argument is a string value that contains the name of the file that you want to access.
- **access_mode:** The access_mode determines the mode in which the file has to be opened, i.e., read, write, append, etc. A complete list of possible values is given below in the table. This is optional parameter and the default file access mode is read (r).
- **buffering:** If the buffering value is set to 0, no buffering takes place. If the buffering value is 1, line buffering is performed while accessing a file. If you specify the buffering

value as an integer greater than 1, then buffering action is performed with the indicated buffer size. If negative, the buffer size is the system default(default behavior).

Here is a list of the different modes of opening a file −

Modes	Description
r	Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode.
rb	Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file. This is the default mode.
r+	Opens a file for both reading and writing. The file pointer placed at the beginning of the file.
rb+	Opens a file for both reading and writing in binary format. The file pointer placed at the beginning of the file.
w	Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
wb	Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
w+	Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
wb+	Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.

a	Opens a file for appending. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
ab	Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
a+	Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.
ab+	Opens a file for both appending and reading in binary format. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.

The *file* Object Attributes

Once a file is opened and you have one *file* object, you can get various information related to that file.

Here is a list of all attributes related to file object:

Attribute	**Description**
file.closed	Returns true if file is closed, false otherwise.
file.mode	Returns access mode with which file was opened.

file.name	Returns name of the file.
file.softspace	Returns false if space explicitly required with print, true otherwise.

Example

```
# Open a file
fo = open("foo.txt", "wb")
print "Name of the file: ", fo.name
print "Closed or not : ", fo.closed
print "Opening mode : ", fo.mode
print "Softspace flag : ", fo.softspace
```

This produces the following result −

```
Name of the file:  foo.txt
Closed or not :  False
Opening mode :  wb
Softspace flag :  0
```

The *close()* Method

The close() method of a *file* object flushes any unwritten information and closes the file object, after which no more writing can be done.Python automatically closes a file when the reference object of a file is reassigned to another file. It is a good practice to use the close() method to close a file.

Syntax

```
fileObject.close();
```

Example

```
# Open a file
fo = open("foo.txt", "wb")
```

```
print "Name of the file: ", fo.name

# Close opend file

fo.close()
```

Result −

```
Name of the file:  foo.txt
```

Reading and Writing Files

The *file* object provides a set of access methods to make our lives easier. We would see how to use *read()* and *write()* methods to read and write files.

The *write()* Method

The *write()* method writes any string to an open file. It is important to note that Python strings can have binary data and not just text.The write() method does not add a newline character ('\n') to the end of the string **Syntax**

```
fileObject.write(string);
```

Here, passed parameter is the content to be written into the opened file. **Example**

```
# Open a file

fo = open("foo.txt", "wb")

fo.write( "Python is a great language.\nYeah its great!!\n");


# Close opend file

fo.close()
```

The above method would create *foo.txt* file and would write given content in that file and finally it would close that file. If you would open this file, it would have following content.

```
Python is a great language.
Yeah its great!!
```

The *read()* Method

The *read()* method reads a string from an open file. It is important to note that Python strings can have binary data. apart from text data.

Syntax

```
fileObject.read([count]);
```

Here, passed parameter is the number of bytes to be read from the opened file. This method starts reading from the beginning of the file and if *count* is missing, then it tries to read as much as possible, maybe until the end of file.

Example

Let's take a file *foo.txt*, which we created above.

```
# Open a file
fo = open("foo.txt", "r+")
str = fo.read(10);
print "Read String is : ", str
# Close opend file
fo.close()
```

This produces the following result −

```
Read String is :  Python is
```

File Positions

The *tell()* method tells you the current position within the file; in other words, the next read or write will occur at that many bytes from the beginning of the file.

The *seek(offset[, from])* method changes the current file position. The *offset* argument indicates the number of bytes to be moved. The *from* argument specifies the reference position from where the bytes are to be moved.

If *from* is set to 0, it means use the beginning of the file as the reference position and 1 means use the current position as the reference position and if it is set to 2 then the end of the file would be taken as the reference position.

Example

Let us take a file *foo.txt*, which we created above.

```
# Open a file
fo = open("foo.txt", "r+")
str = fo.read(10);
print "Read String is : ", str

# Check current position
position = fo.tell();
print "Current file position : ", position

# Reposition pointer at the beginning once again
position = fo.seek(0, 0);
str = fo.read(10);
print "Again read String is : ", str
# Close opend file
fo.close()
```

This produces the following result −

```
Read String is :  Python is
Current file position :  10
```

```
Again read String is :  Python is
```

Renaming and Deleting Files

Python **os** module provides methods that help you perform file-processing operations, such as renaming and deleting files.

To use this module you need to import it first and then you can call any related functions.

The rename() Method

The *rename()* method takes two arguments, the current filename and the new filename.

Syntax

```
os.rename(current_file_name, new_file_name)
```

Example

Following is the example to rename an existing file *test1.txt*:

```
import os


# Rename a file from test1.txt to test2.txt
os.rename( "test1.txt", "test2.txt" )
```

The *remove()* Method

You can use the *remove()* method to delete files by supplying the name of the file to be deleted as the argument.

Syntax

```
os.remove(file_name)
```

Example

Following is the example to delete an existing file *test2.txt* −

```
#!/usr/bin/python
import os
```

```
# Delete file test2.txt
os.remove("text2.txt")
```

Directories in Python

All files are contained within various directories, and Python has no problem handling these too. The **os** module has several methods that help you create, remove, and change directories.

The *mkdir()* Method

You can use the *mkdir()* method of the **os** module to create directories in the current directory. You need to supply an argument to this method which contains the name of the directory to be created.

Syntax

```
os.mkdir("newdir")
```

Example

Following is the example to create a directory *test* in the current directory −

```
#!/usr/bin/python
import os

# Create a directory "test"
os.mkdir("test")
```

The *chdir()* Method

You can use the *chdir()* method to change the current directory. The chdir() method takes an argument, which is the name of the directory that you want to make the current directory.

Syntax

```
os.chdir("newdir")
```

Example

Following is the example to go into "/home/newdir" directory −

```
#!/usr/bin/python
import os

# Changing a directory to "/home/newdir"
os.chdir("/home/newdir")
```

The *getcwd()* Method

The *getcwd()* method displays the current working directory.

Syntax

```
os.getcwd()
```

Example

Following is the example to give current directory −

```
import os

# This would give location of the current directory
os.getcwd()
```

The *rmdir()* Method

The *rmdir()* method deletes the directory, which is passed as an argument in the method.

Before removing a directory, all the contents in it should be removed.

Syntax:

```
os.rmdir('dirname')
```

Example

Following is the example to remove "/tmp/test" directory. It is required to give fully qualified name of the directory, otherwise it would search for that directory in the current directory.

```
import os
# This would  remove "/tmp/test"  directory.
os.rmdir( "/tmp/test"  )
```

File & Directory Related Methods

There are three important sources, which provide a wide range of utility methods to handle and manipulate files & directories on Windows and Unix operating systems. They are as follows −

- File Object Methods: The *file* object provides functions to manipulate files.
- OS Object Methods: This provides methods to process files as well as directories.

Python provides two very important features to handle any unexpected error in your Python programs and to add debugging capabilities in them −

- **Exception Handling:** This would be covered in this tutorial. Here is a list standard Exceptions available in Python: Standard Exceptions.
- **Assertions:** This would be covered in Assertions in Python

List of Standard Exceptions −

EXCEPTION NAME	DESCRIPTION
Exception	Base class for all exceptions

StopIteration	Raised when the next() method of an iterator does not point to any object.
SystemExit	Raised by the sys.exit() function.
StandardError	Base class for all built-in exceptions except StopIteration and SystemExit.
ArithmeticError	Base class for all errors that occur for numeric calculation.
OverflowError	Raised when a calculation exceeds maximum limit for a numeric type.
FloatingPointError	Raised when a floating point calculation fails.
ZeroDivisionError	Raised when division or modulo by zero takes place for all numeric types.
AssertionError	Raised in case of failure of the Assert statement.
AttributeError	Raised in case of failure of attribute reference or assignment.
EOFError	Raised when there is no input from either the raw_input() or input() function and the end of file is reached.
ImportError	Raised when an import statement fails.

KeyboardInterrupt	Raised when the user interrupts program execution, usually by pressing Ctrl+c.
LookupError	Base class for all lookup errors.
IndexError KeyError	Raised when an index is not found in a sequence. Raised when the specified key is not found in the dictionary.
NameError	Raised when an identifier is not found in the local or global namespace.
UnboundLocalError EnvironmentError	Raised when trying to access a local variable in a function or method but no value has been assigned to it. Base class for all exceptions that occur outside the Python environment.
IOError IOError	Raised when an input/ output operation fails, such as the print statement or the open() function when trying to open a file that does not exist. Raised for operating system-related errors.
SyntaxError IndentationError	Raised when there is an error in Python syntax. Raised when indentation is not specified properly.
SystemError	Raised when the interpreter finds an internal problem, but when this error is encountered the Python interpreter does not exit.

SystemExit	Raised when Python interpreter is quit by using the sys.exit() function. If not handled in the code, causes the interpreter to exit.
TypeError	Raised when an operation or function is attempted that is invalid for the specified data type.
ValueError	Raised when the built-in function for a data type has the valid type of arguments, but the arguments have invalid values specified.
RuntimeError	Raised when a generated error does not fall into any category.
NotImplementedError	Raised when an abstract method that needs to be implemented in an inherited class is not actually implemented.

What is Exception?

An exception is an event, which occurs during the execution of a program that disrupts the normal flow of the program's instructions. In general, when a Python script encounters a situation that it cannot cope with, it raises an exception. An exception is a Python object that represents an error.

When a Python script raises an exception, it must either handle the exception immediately otherwise it terminates and quits.

Handling an exception

If you have some *suspicious* code that may raise an exception, you can defend your program by placing the suspicious code in a **try:** block. After the try: block, include an **except:** statement, followed by a block of code which handles the problem as elegantly as possible.

The Python standard for database interfaces is the Python DB-API. Most Python database interfaces adhere to this standard.

You can choose the right database for your application. Python Database API supports a wide range of database servers such as −

- GadFly
- mSQL
- MySQL
- PostgreSQL
- Microsoft SQL Server 2000
- Informix
- Interbase
- Oracle
- Sybase

The DB API provides a minimal standard for working with databases using Python structures and syntax wherever possible. This API includes the following:

- Importing the API module.
- Acquiring a connection with the database.
- Issuing SQL statements and stored procedures.
- Closing the connection

2. SYSTEM STUDY

2.1 FEASIBILITY STUDY

The feasibility of the project is analyzed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential.

Three key considerations involved in the feasibility analysis are

- ECONOMICAL FEASIBILITY
- TECHNICAL FEASIBILITY
- SOCIAL FEASIBILITY

ECONOMICAL FEASIBILITY

This study is carried out to check the economic impact that the system will have on the organization. The amount of fund that the company can pour into the research and development of the system is limited. The expenditures must be justified. Thus the developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products had to be purchased.

TECHNICAL FEASIBILITY

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. This will lead to high demands on the available technical resources. This will lead to high demands being placed on the client. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

SOCIAL FEASIBILITY

The aspect of study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system.