**ABSTRACT:**

In the era of big data, network security faces escalating threats that demand innovative solutions. This project introduces a Hybrid Network Intrusion Detection System (HNIDS) that combines the power of Convolutional Neural Networks (CNNs) and Support Vector Machines (SVMs) to tackle the challenges of network intrusion detection in environments characterized by imbalanced datasets. HNIDS leverages CNNs for automatic feature extraction, providing an effective means of discerning subtle intrusion patterns in network traffic data. It also harnesses SVMs to address the issue of imbalanced data using a combination of SMOTE oversampling and Tomek-Links undersampling techniques, ensuring balanced representation of minority classes. The results demonstrate a significant improvement in detection accuracy when compared to standalone models. This hybrid approach offers a potent solution to enhance intrusion detection in complex network landscapes, ensuring accurate and robust security measures, regardless of data volume and class distribution intricacies.

**INTRODUCTION:**

In the rapidly evolving landscape of information technology and data communication, network security has become a paramount concern. With the proliferation of interconnected devices, cloud computing, and the vast amounts of data generated and transmitted daily, the risk of network intrusions has grown exponentially. To mitigate these threats effectively, there is a pressing need for advanced Intrusion Detection Systems (IDS) that can adapt to the complexities of modern network environments. This introduction provides an in-depth overview of the research project, "A Hybrid Network Intrusion Detection System (HNIDS)," which synergizes Convolutional Neural Networks (CNNs) and Support Vector Machines (SVMs) to address network intrusion detection challenges in big data environments with imbalanced datasets.

**OVERVIEW:**

The network intrusion landscape has witnessed a significant transformation over the years, driven by the rapid advancement of technology and the increasing sophistication of malicious actors. Intrusion Detection Systems (IDS) play a pivotal role in safeguarding network infrastructures by identifying and responding to unauthorized access, malicious activities, and potential security breaches. Traditional IDS approaches predominantly rely on rule-based systems and signature-based detection methods, which are often insufficient in dealing with the ever-evolving nature of cyber threats.To address the shortcomings of conventional IDS, this research project introduces a Hybrid Network Intrusion Detection System (HNIDS) that combines the strengths of CNNs and SVMs. HNIDS is designed to overcome the challenges presented by modern network environments characterized by big data volumes and imbalanced datasets. This innovation seeks to revolutionize the field of network intrusion detection by offering a more robust and adaptive approach to identifying and mitigating security threats.

**INTRUSION DETECTION:**

Intrusion detection is a critical component of network security. It involves the monitoring and analysis of network traffic and system activities to identify unauthorized access, unauthorized activities, and potential security threats. IDS can be broadly categorized into two main types: host-based IDS (HIDS) and network-based IDS (NIDS).

1. Host-Based IDS (HIDS): HIDS is deployed on individual network hosts or devices to monitor their internal activities and detect anomalies. These anomalies can include unauthorized access, unusual file modifications, or suspicious processes running on a host.

2. Network-Based IDS (NIDS): NIDS is strategically placed at key points within a network to analyze network traffic and identify abnormal patterns or signatures associated with known attacks or emerging threats.

The primary goal of intrusion detection is to provide early warning and response to potential security breaches, thereby safeguarding network integrity, confidentiality, and availability. To achieve this goal, intrusion detection systems rely on various detection techniques, including signature-based detection, anomaly-based detection, and hybrid approaches.

**PROBLEM STATEMENT:**

Despite the importance of intrusion detection, several challenges hinder its effectiveness, especially in modern network environments characterized by big data and imbalanced datasets. The traditional intrusion detection methods, which are rule-based or signature-based, often struggle to keep pace with the rapidly evolving tactics employed by cybercriminals. These methods heavily rely on predefined patterns or signatures of known attacks and are limited in their ability to detect previously unseen or zero-day attacks.Moreover, imbalanced datasets pose a significant problem in intrusion detection. In many real-world scenarios, benign network traffic far outweighs malicious traffic. As a result, intrusion detection models trained on imbalanced datasets tend to prioritize the majority class (normal traffic) and underperform in identifying the minority class (malicious traffic). This bias towards the majority class can lead to false negatives, where actual intrusions go undetected, posing a serious security risk.

**The necessity for a Hybrid Network Intrusion Detection System (HNIDS):**

The necessity for an advanced and adaptive Intrusion Detection System like HNIDS is driven by several compelling factors:

1. Evolving Threat Landscape: The nature of network intrusions is continually evolving, with cybercriminals employing sophisticated techniques to circumvent traditional detection mechanisms. HNIDS is designed to address emerging threats effectively by leveraging advanced machine learning techniques.

2. Big Data Environments: In the age of big data, network traffic data has grown exponentially, making it increasingly challenging to process and analyze for potential security threats. HNIDS is tailored to handle vast volumes of data efficiently, ensuring that no threats go unnoticed.

3. Imbalanced Datasets: Imbalanced datasets are a common issue in intrusion detection, and they hinder the accurate identification of malicious traffic. HNIDS employs techniques such as Synthetic Minority Over-sampling Technique (SMOTE) and Tomek-Links undersampling to balance the dataset, enhancing the detection of both majority and minority classes.

4. Real-time Detection: Network intrusions can have severe consequences, and rapid detection is crucial for effective response. HNIDS is designed to provide real-time intrusion detection, minimizing the time between threat identification and response.

5. Reduced False Positives: The combination of CNNs for feature extraction and SVMs for classification in HNIDS aims to reduce false positives, ensuring that legitimate network traffic is not mistakenly flagged as malicious.

Advantages of HNIDS:

The Hybrid Network Intrusion Detection System (HNIDS) offers several advantages over traditional IDS approaches, making it an invaluable tool for securing modern network environments:

1. Enhanced Accuracy: HNIDS leverages Convolutional Neural Networks (CNNs) for automatic feature extraction, enabling it to identify complex patterns and anomalies in network traffic accurately. This advanced feature extraction mechanism significantly enhances detection accuracy.

2. Adaptability: HNIDS can adapt to evolving threats and zero-day attacks since it is not solely dependent on predefined signatures. It can learn from new data and continuously improve its detection capabilities.

3. Big Data Handling: In the era of big data, HNIDS efficiently processes and analyzes vast volumes of network traffic data, ensuring that no threat goes undetected. Its scalability makes it suitable for large and complex network environments.

4. Imbalanced Data Mitigation: HNIDS employs techniques such as Synthetic Minority Over-sampling Technique (SMOTE) and Tomek-Links undersampling to address the problem of imbalanced datasets. This ensures balanced representation of both normal and malicious traffic, reducing false negatives.

5. Real-time Detection and Response: HNIDS is designed for real-time intrusion detection, allowing for swift responses to potential threats. This real-time capability minimizes the window of opportunity for attackers.

6. Reduced False Positives: By combining the feature extraction capabilities of CNNs with the classification power of Support Vector Machines (SVMs), HNIDS reduces the occurrence of false positives, ensuring that legitimate network traffic is not mistakenly labeled as malicious.

7. Comprehensive Security: HNIDS offers a comprehensive security solution, capable of detecting a wide range of network intrusions, from known attacks to emerging threats.

**LITERATURE SURVEY:**

1.Title: Machine Learning Based Intrusion Detection System

Authors: Anish Halimaa A.,K. Sundarakantham,

Publication: 2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI)

In order to examine malicious activity that occurs in a network or a system, intrusion detection system is used. Intrusion Detection is software or a device that scans a system or a network for a distrustful activity. Due to the growing connectivity between computers, intrusion detection becomes vital to perform network security. Various machine learning techniques and statistical methodologies have been used to build different types of Intrusion Detection Systems to protect the networks. Performance of an Intrusion Detection is mainly depends on accuracy. Accuracy for Intrusion detection must be enhanced to reduce false alarms and to increase the detection rate. In order to improve the performance, different techniques have been used in recent works. Analyzing huge network traffic data is the main work of intrusion detection system. A well-organized classification methodology is required to overcome this issue. This issue is taken in proposed approach. Machine learning techniques like Support Vector Machine (SVM) and Naive Bayes are applied. These techniques are well-known to solve the classification problems. For evaluation of intrusion detection system, NSL- KDD knowledge discovery Dataset is taken. The outcomes show that SVM works better than Naive Bayes. To perform comparative analysis, effective classification methods like Support Vector Machine and Naive Bayes are taken, their accuracy and misclassification rate get calculated

2.Title: Network intrusion detection

Authors: B. Mukherjee,L.T. Heberlein,K.N. Levitt,

Publication: IEEE Network

Intrusion detection is a new, retrofit approach for providing a sense of security in existing computers and data networks, while allowing them to operate in their current "open" mode. The goal of intrusion detection is to identify unauthorized use, misuse, and abuse of computer systems by both system insiders and external penetrators. The intrusion detection problem is becoming a challenging task due to the proliferation of heterogeneous computer networks since the increased connectivity of computer systems gives greater access to outsiders and makes it easier for intruders to avoid identification. Intrusion detection systems (IDSs) are based on the beliefs that an intruder's behavior will be noticeably different from that of a legitimate user and that many unauthorized actions are detectable. Typically, IDSs employ statistical anomaly and rulebased misuse models in order to detect intrusions. A number of prototype IDSs have been developed at several institutions, and some of them have also been deployed on an experimental basis in operational systems. In the present paper, several host-based and network-based IDSs are surveyed, and the characteristics of the corresponding systems are identified. The host-based systems employ the host operating system's audit trails as the main source of input to detect intrusive activity, while most of the network-based IDSs build their detection mechanism on monitored network traffic, and some employ host audit trails as well.

3.Title: Intrusion Detection System using Machine Learning Techniques: A Review

Authors: Usman Shuaibu Musa,Megha Chhabra,Aniso Ali,Mandeep Kaur,

Publication: 2020 International Conference on Smart Electronics and Communication (ICOSEC)

The rapid growth in the use of computer networks results in the issues of maintaining the network availability, integrity, and confidentiality. This necessitates the network administrators to adopt various types of intrusion detection systems (IDS) that help in monitoring the network traffics for unauthorized and malicious activities. Intrusion is the breach of security policy with malicious intent. Therefore, intrusion detection system monitors traffic flowing on a network through computer systems to search for malicious activities and known threats, sending up alerts when it finds those threats. The detection of malicious activities is of two types, the misuse or signature-based detection in which the IDS collects information, analyzes it and then compares it to the attack signatures stored in a large database. The second detection is the anomaly detection which assumes malicious activity as any action that deviates from normal behavior. The proposed paper presents an overview of various works being done on building an efficient IDS using single, hybrid and ensemble machine learning (ML) classifiers, evaluated using seven different datasets. The results obtained by various works were discussed and compared which gives a clear path and guide for future work

4.Title: Intrusion Detection: A Survey

Authors: F. Sabahi,A. Movaghar,

Publication: 2008 Third International Conference on Systems and Networks Communications

The rapid proliferation of computer networks has changed the prospect of network security. An easy accessibility condition cause computer networkpsilas vulnerable against several threats from hackers. Threats to networks are numerous and potentially devastating. Up to the moment, researchers have developed Intrusion Detection Systems (IDS) capable of detecting attacks in several available environments. A boundlessness of methods for misuse detection as well as anomaly detection has been applied. Many of the technologies proposed are complementary to each other, since for different kind of environments some approaches perform better than others. This paper presents a taxonomy of intrusion detection systems that is then used to survey and classify them. The taxonomy consists of the detection principle, and second of certain operational aspects of the intrusion detection system

5.Title: Intrusion detection and prevention system: Challenges & opportunities

Authors: Uzair Bashir,Manzoor Chachoo,

Publication: 2014 International Conference on Computing for Sustainable Global Development (INDIACom)

The idea of making everything available readily and universally has led to a revolution in the field of networks. In spite of the tremendous growth of technologies in the field of networks and information technology, we still lack in preventing our resources from theft/attacks. This may not concern small organizations but it is a serious issue as far as industry/companies or national security is concerned. Organizations are facing an increasing number of threats every day in the form of viruses, intrusions, etc. Since many different mechanisms were opted by organizations in the form of intrusion detection and prevention systems to protect themselves from these kinds of attacks, there are many security breaches which go undetected. In order to understand the security risks and IDPS(intrusion detection and prevention system), we will first survey about the common security breaches and then after discuss what are different opportunities and challenges in this particular field. In this paper we have made a survey on the overall progress of intrusion detection systems. We survey the existing types, techniques and architectures of Intrusion Detection Systems in the literature. Finally we outline the present research challenges and issue

6.Title: A Review on Intrusion Detection System using Machine Learning Techniques

Authors: Usman Shuaibu Musa,Sudeshna Chakraborty,Muhammad M. Abdullahi,Tarun Maini,

Publication: 2021 International Conference on Computing Communication and Intelligent Systems (ICCCIS)

Computer networks are exposed to cyber related attacks due to the common usage of internet, as the result of such, several intrusion detection systems (IDSs) were proposed by several researchers. Among key research issues in securing network is detecting intrusions. It helps to recognize unauthorized usage and attacks as a measure to ensure the secure the networku2019s security. Various approaches have been proposed to determine the most effective features and hence enhance the efficiency of intrusion detection systems, the methods include, machine learning-based (ML), Bayesian based algorithm, nature inspired meta-heuristic techniques, swarm smart algorithm, and Markov neural network. Over years, the various works being carried out were evaluated on different datasets. This paper presents a thorough review on various research articles that employed single, hybrid and ensemble classification algorithms. The results metrics, shortcomings and datasets used by the studied articles in the development of IDS were compared. A future direction for potential researches is also given

7.Title: Intelligent intrusion detection system using clustered self organized map

Authors: Muder Almi'ani,Alia Abu Ghazleh,Amer Al-Rahayfeh,Abdul Razaque,

Publication: 2018 Fifth International Conference on Software Defined Systems (SDS)

The impact of information security breaching becomes bigger and complicated to ignore every day. New and more sophisticated attacks are emerging and developed; requiring the information systems and networks be protected in a highly flexible and accurate manner. Intrusion Detection Systems (IDS) are considered one of the basic building blocks of the protection wall against these intrusive activities through detecting it before it hits the network systems. Artificial neural networks have been used successfully for addressing the high accuracy and precision demands of intrusion detection systems. In this paper, we built an intelligent intrusion detection system using clustered version of Self-Organized Map (SOM) network. The proposed system consists of two subsequent stages: first, SOM network was built, then a hierarchical agglomerative clustering using k-means was applied on SOM neurons. The proposed work in this research paper addresses the issues of sensitivity and time consumption for each connection record processing. The proposed system was demonstrated using NSL-KDD benchmark dataset, where it has achieved superior sensitivity reached up to 96.66 % in less than 0.08 milliseconds per connection record

8.Title: Intrusion Detection System Using Machine Learning

Authors: Ajmeera Kiran,S. Wilson Prakash,B Anand Kumar,Likhitha,Tammana Sameeratmaja,Ungarala Satya Surya Ram Charan,

Publication: 2023 International Conference on Computer Communication and Informatics (ICCCI)

The use of computers and the internet has spread rapidly over the course of the past few decades. Every day, more and more people are coming to rely heavily on the internet. When it comes to the field of information security, the subject of security is one that is becoming an increasingly important focus. It is vital to design a powerful intrusion detection system in order to prevent computer hackers and other intruders from effectively getting into computer networks or systems. This can be accomplished by: (IDS). The danger and attack detection capabilities of the computer system are built into the intrusion detection system. Abuse has occurred and can be used to identify invasions when there is a deviation between a preset pattern of intrusion and an observed pattern of intrusion. An intrusion detection system (IDS) is a piece of hardware (or software) that is used to generate reports for a Management Station as well as monitor network and/or system activities for unethical behaviour or policy violations. In the current study, an approach known as machine learning is suggested as a possible paradigm for the development of a network intrusion detection system. The results of the experiment show that the strategy that was suggested improves the capability of intrusion detection

9.Title: Using Deep Learning Techniques for Network Intrusion Detection

Authors: Sara Al-Emadi,Aisha Al-Mohannadi,Felwa Al-Senaid,

Publication: 2020 IEEE International Conference on Informatics IoT and Enabling Technologies (ICIoT)

In recent years, there has been a significant increase in network intrusion attacks which raises a great concern from the privacy and security aspects. Due to the advancement of the technology, cyber-security attacks are becoming very complex such that the current detection systems are not sufficient enough to address this issue. Therefore, an implementation of an intelligent and effective network intrusion detection system would be crucial to solve this problem. In this paper, we use deep learning techniques, namely, Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN) to design an intelligent detection system which is able to detect different network intrusions. Additionally, we evaluate the performance of the proposed solution using different evaluation matrices and we present a comparison between the results of our proposed solution to find the best model for the network intrusion detection system

10.Title: A survey of intrusion detection system

Authors: Loubna Dali,Ahmed Bentajer,Elmoutaoukkil Abdelmajid,Karim Abouelmehdi,Hoda Elsayed,Eladnani Fatiha,Benihssane Abderahim,

Publication: 2015 2nd World Symposium on Web Applications and Networking (WSWAN)

In this paper, we presented a survey on intrusion detection systems (IDS). First, we referred to different mechanisms of intrusion detection. Furthermore, we detailed the types of IDS. We have focused on the application IDS, specifically on the IDS Network, and the IDS in the cloud computing environment. Finally, the contribution of every single type of IDS is described

**METHODOLOGY:**

Label Encoding

Feature Reduction

Normalization

Test Dataset

Train Dataset

Smote +Tomek Link (STL)

**CNN-SVM**

CIDDS-001 Dataset

Results

The methodology of the proposed Hybrid Network Intrusion Detection System (HNIDS) is a comprehensive process that combines data balancing techniques, SMOTE (Synthetic Minority Oversampling Technique) and Tomek-links, with a powerful data classification approach using Convolutional Neural Networks (CNN) and Support Vector Machines (SVM). This section will elaborate on each component of the methodology, including the working platform, dataset used, and the step-by-step workflow.

1. Data Balancing Techniques:

Data imbalance is a common challenge in network intrusion detection, where benign traffic (majority class) significantly outweighs malicious traffic (minority class). This imbalance can lead to biased models that perform poorly in detecting intrusions. To address this issue, HNIDS employs two data balancing techniques:

a. SMOTE (Synthetic Minority Oversampling Technique):

SMOTE is a powerful technique used to balance imbalanced datasets by oversampling the minority class. It generates synthetic instances for the minority class by interpolating between existing instances, effectively increasing the representation of the minority class.

In the context of HNIDS, SMOTE is applied to the training dataset to ensure a more balanced representation of normal and malicious network traffic.

b. Tomek-links:

Tomek-links are used to remove redundant and potentially confusing samples from the dataset. They identify pairs of instances, one from the majority class and one from the minority class, that are nearest neighbors to each other. Removing these pairs helps to improve the separation between the classes.

In HNIDS, Tomek-links are applied after SMOTE to further refine the dataset and enhance the distinction between normal and malicious traffic.

2. Data Classification Approach:

The classification of network traffic into normal or malicious categories is a critical aspect of intrusion detection. HNIDS employs a hybrid classification approach that combines the strengths of Convolutional Neural Networks (CNNs) and Support Vector Machines (SVMs).

a. Convolutional Neural Network (CNN):

CNNs are a type of deep learning model particularly well-suited for image and pattern recognition tasks. In the context of HNIDS, CNNs are used for feature extraction from network traffic data.

CNNs are designed to automatically learn relevant features from the raw data, allowing them to identify complex patterns and anomalies in network traffic.

The feature vectors extracted by the CNN serve as the input to the SVM for classification.

b. Support Vector Machine (SVM):

SVMs are powerful machine learning models for classification tasks. They work by finding the optimal hyperplane that best separates data points of different classes.

In HNIDS, SVMs take the feature vectors extracted by the CNN and perform the final classification into normal or malicious traffic. SVMs are known for their effectiveness in handling high-dimensional data and can provide robust classification results.

3. Working Platform:

HNIDS is implemented on a local development platform using the Thonny Integrated Development Environment (IDE). Thonny is a lightweight Python IDE that provides a user-friendly interface for writing, debugging, and executing Python code. It is an ideal choice for developing and testing machine learning models and intrusion detection systems. The Python programming language is a versatile and widely-used tool for machine learning and data analysis, making Thonny a suitable platform for developing HNIDS.

4. Dataset Used:

The dataset utilized for training and testing the HNIDS is CIDDS-001 (Cybersecurity Intrusion Detection Data Set) published by HS-Coburg, Germany. This dataset is specifically designed for network intrusion detection research and comprises network traffic data captured in a controlled environment.

CIDDS-001 is an invaluable resource for evaluating intrusion detection systems, as it contains a diverse range of network traffic, including normal and malicious activities. The dataset is divided into training and testing subsets, with 80% of the data allocated for training and 20% for testing. This division ensures that the HNIDS model is trained on a substantial portion of the data and rigorously tested on unseen instances, providing a robust assessment of its performance.

Step-by-Step Workflow:

Now, let's delve into the step-by-step workflow of HNIDS, from data preprocessing to the final intrusion detection:

Step 1: Data Preprocessing

The initial step involves data preprocessing, where the CIDDS-001 dataset is loaded and cleaned. Any missing or inconsistent data is handled to ensure the dataset is ready for training and testing.

Step 2: Data Splitting

The dataset is divided into two subsets: 80% for training and 20% for testing. This division is critical to assess the model's performance on unseen data accurately.

Step 3: Data Balancing

The training dataset, which is typically characterized by data imbalance, is subjected to data balancing techniques. First, SMOTE is applied to oversample the minority class, ensuring a balanced representation of normal and malicious traffic. Then, Tomek-links are employed to refine the dataset further by removing redundant instances.

Step 4: Feature Extraction (CNN)

With the balanced training dataset in place, the feature extraction phase begins. Convolutional Neural Networks (CNNs) are employed to automatically learn and extract relevant features from the network traffic data.

The CNN processes the raw network traffic data, capturing intricate patterns and anomalies that would be challenging to discern through traditional methods.

Step 5: Feature Vector Generation

The output of the CNN is a set of feature vectors that represent the network traffic data. These feature vectors are used as input to the Support Vector Machine (SVM) for classification.

Step 6: Data Classification (SVM)

Support Vector Machines (SVMs) take the feature vectors and perform the final classification. SVMs are adept at handling high-dimensional data and are particularly effective in distinguishing between normal and malicious network traffic.

The SVM classifies each data point in the testing dataset, providing intrusion detection results.

Step 7: Evaluation and Metrics

The performance of HNIDS is rigorously evaluated using a range of intrusion detection metrics, including but not limited to accuracy, precision, recall, F1-score, and the ROC curve.

These metrics provide insights into the system's ability to correctly identify intrusions while minimizing false alarms.

Step 8: Fine-Tuning and Optimization

Based on the evaluation results, the system may be fine-tuned and optimized to improve its performance. This step involves adjusting hyperparameters, modifying the network architecture, or enhancing data preprocessing techniques.

Step 9: Deployment and Real-time Monitoring

Once the HNIDS achieves satisfactory performance, it can be deployed in a real network environment for continuous intrusion detection. Real-time monitoring ensures that network security is maintained, and threats are detected as they occur.

Step 10: Continuous Improvement

The intrusion detection system should be continuously monitored and improved to adapt to new threats and evolving network landscapes. Regular updates and retraining with fresh data are essential for long-term effectiveness.

**SOFTWARE DESCRIPTION:**

PYTHON 3.7:

Python is an interpreter, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace.

Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object- oriented programming. Python’s elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in manya reason most platforms and

may be freely distributed. The same site also contains distributions of and pointers to many free third party Python modules, programs and tools, and additional documentation. The Python interpreter is easily extended with new functions and data types implemented in C or C++ (or other languages callable from C). Python is also suitable as an extension language for customizable applications. This tutorial introduces the reader informally to the basic concepts and features of the Python language and system. It helps to have a Python interpreter handy for hands-on experience, but all examples are self-contained, so the tutorial can be read off- line as well. For a description of standard objects and modules, see library-index. Reference-index gives a more formal definition of the language. To write extensions in C or C++, read extending-index and c-api-index. There are also several books covering Python in depth. This tutorial does not attempt to be comprehensive and cover every single feature, or even every commonly used feature. Instead, it introduces many of Python’s most notes worthy features, and will give you a good idea of the language’s flavor and style. After reading it, you will be able to read and write Python modules and programs, and you will be ready to learn more about the various Python library modules described in library-index. If you do much work on computers, eventually you find that there’s some task you’d like

to automate. For example, you may wish to perform a search-and-replace over a large number of text files, or rename and rearrange a bunch of photo files in a complicated way. Perhaps you’d like to write a small custom database, or a specialized

GUI application or a simple game. If you’re a professional software developer, you may have to work with several C/C++/Java libraries but find the usual write/compile/test/re-compile cycle is too slow. Perhaps you’re writing a test suite for such a library and find writing the testing code a tedious task. Or maybe you’ve written a program that could use an extension language, and you don’t want to design and implement a whole new language for your application.

Typing an end-of-file character (Control-D on Unix, Control-Z on Windows) at the primary prompt causes the interpreter to exit with a zero exit status. If that doesn’t work, you can exit the interpreter by typing the following command: quit(). The interpreter’s line-editing features include interactive editing, history substitution and code completion on systems that support read line. Perhaps the quickest check to see whether command line editing is supported is typing Control-P to the first Python prompt you get. If it beeps, you have command line editing; see Appendix Interactive Input Editing and History Substitution for an introduction to the keys. Ifnothing appears to happen, or if ^P is echoed, command line editing isn’t available; you’ll only be able to use backspace to remove characters from the current line. The interpreter operates somewhat like the Unix shell: when called with standard input connected to a tty device, it reads and executes commands interactively; when called with a file name argument or with a file as standard input, it reads and executes a script from that file. A second way of starting the interpreter is python -c command [arg] ..., which executes the statement(s) in command, analogous to the shell’s -c option. Since Python statements often contain spaces or other characters that are special to the shell, it is usually advised to quote commands in its entirety with single quotes.Some Python modules are also useful as scripts. These can be invoked using python-m module [arg]...,which executes the source file for the module as if you had spelled out its full name on the command line. When a script file is used, it is sometimes useful to be able to run the script and enter interactive mode afterwards. This can be done by passing -i before the script.

There are tools which use doc strings to automatically produce online or printed documentation or to let the user interactively browse through code; it’s good practice to include doc strings in code that you write, so make a habit of it. The execution of a function introduces a new symbol table used for the local variables of the function. More precisely, all variable assignments in a functions to read the value in the local symbol table; whereas variable references first look in the local symbol table, then in the local symbol tables of enclosing functions, then in the global symbol table, and finally in the table of built-in names. Thus, global variables cannot be directly assigned a value within a function (unless named in a global statement), although they may be referenced. The actual parameters (arguments) to a function call are introduced in the local symbol table of the called function when it is called; thus, arguments are passed using call by value (where the value is always an object reference, not the value of the object).1 When a function calls another function, a new local symbol table is created for that call. A function definition introduces the function name in the current symbol table. The value of the function name has a type that is recognized by the interpreter as a user-defined function. This value can be assigned to another name which can then also be used as a function.

Annotations are stored in the annotations attribute of the function as a dictionary and haven o effect on any other part of the function. Parameter annotations are defined by a colon after the parameter name, followed by an expression evaluating to the value of the annotation. Return annotationsare defined by a literal ->, followed by an expression, between the parameter list and the colon denoting the end of the def statement.

The comparison operators in and not in check whether a value occurs (does not occur) in a sequence. The operator is and does not compare whether two objects are really the same object; this only matters for mutable objects like lists. All comparison operators have the same priority, which is lower than that of all numerical operators. Comparisons can be chained. For example,a<b==ctestswhetheraislessthanbandmoreoverbequalsc. Comparisons may be combined using the Boolean operators and the outcome of a comparison (or of any other Boolean expression) may be negated with not. These have lower priorities than comparison operators; between them, not has the highest priority and or the lowest, so that A and not B or C is equivalent to (A and (not B)) or C. As always, parentheses can be used to express the desired composition. The Boolean operators and are so-called short-circuit operators: their arguments are evaluated from left to right, and evaluation stops as soon as the outcome is determined. For example, if A and C are true but Bis false, A and B and C does not evaluate the expression C. When used as a general value and not as a Boolean, the return value of a short-circuit operator is the last evaluated argument.

Classes provide a means of bundling data and functionality together. Creating a new class creates a new type of object, allowing new instances of that type to be made. Each class instance can have attributes attached to it for maintaining its state. Class instances can also have methods (defined by its class) for modifying its state. Compared with other programming languages, Python’s class mechanism adds classes with a minimum of new syntax and semantics. It is a mixture of the class mechanisms found in C++ and Modula-3. Python classes provide all the standard features of Object Oriented Programming: the class inheritance mechanism allows multiple base classes, a derived class can override any methods of its base class or classes, and a method can call the method of a base class with the same name. Objects can contain arbitrary amounts and kinds of data. As is true for modules, classes partake of the dynamic nature of Python: they are created at runtime, and can be modified further after creation. In C++ terminology, normally class members (including the data members) are public (except see below Private Variables), and all member functions are virtual. A sin Modula-3, there are no short hands for referencing the object’s members from its methods: the method function is declared with an explicit first argument representing the object, which is provided implicitly by the call. A sin Small talk, classes themselves are objects. This providesSemantics for importing and renaming. Unlike C++ and Modula-3, built-in types can be used as base classes for extension by the user. Also, like in C++, most built-in operators with special syntax (arithmetic operators, sub scripting etc.) can be redefined for class instances.(Lacking universally accepted terminology to talk about classes, I will make occasional use of Smalltalk and C++ terms. I would use Modula-3 terms, since its object- oriented semantics are closer to those of Python than C++, but I expect that few readers have heard of it.)

Objects have individuality, and multiple names (in multiple scopes) can be bound to the same object. This is known as aliasing in other languages. This is usually not appreciated on a first glance at Python, and can be safely ignored when dealing with immutable basic types (numbers, strings, tuples).However, aliasing has a possibly surprising effect on these mantic of Python code involving mutable objects such as lists, dictionaries, and most other types. This is usually used to the benefit of the program, since aliases behave like pointers in some respects. For example, passing an object is cheap since only a pointer is passed by the implementation; and if a function modifies an object passed as an argument, the caller will see the change — this eliminates the need for two different argument passing mechanisms as in Pascal.

A namespace is a mapping from names to objects. Most name spaces are currently implemented as Python dictionaries, but that’s normally not noticeable in any way (except for performance), and it may change in the future. Examples of name spaces are: these to f built-in names (containing functions such as abs(), and built-in exception names); the global names in a module; and the local names in a function invocation. In a sense the set of attributes of an object also form a namespace. The important thing to know about namespaces is that there is absolutely no relation between names in different namespaces; for instance, two different modules may both define a function maximize without confusion — users of the modules must prefix it with the module name. By the way, I use the word attribute for any name following a dot — for example, in the expression z. real, real is an attribute of the object z. Strictly speaking, references to names in modules are attribute references: in the expression modname.funcname, modname is a module object and funcname is an attribute of it. In this case there happens to be a straight forward mapping between the module’s attributes and the global names defined in the module: they share the same namespace!1 Attributes may be read-only or writable. In the latter case, assignment to attributes is

possible. Module attributes are writable: you can

write modname.the\_answer = 42. Writable attributes may also be deleted with the del statement. For example, del mod name .the\_ answer will remove the attribute the\_answer from the object named by mod name. Namespaces are created at different moments and have different lifetimes. The namespace containing the built-in names is created when the Python interpreter starts up, and is never deleted. The global namespace for a module is created when the module definition is read in; normally, module namespaces also last until the interpreter quits.The statements executed by the top-level invocation of the interpreter, either read from a script file or interactively, are considered part of a module called main, so they have their own global namespace.(The built-in names actually also live in a module; this is called built ins.) The local namespace for a function is created when the function is called, and deleted when the function returns or raises an exception that is not handled within the function. (Actually, forgetting would be a better way to describe what actually happens.) Of course, recursive invocations each have their own local namespace.

To speed uploading modules, Python caches the compiled version

of each module in the pycache directory under the name

module.version.pyc, where the version encodes the format of the compiled

file; it generally contains the Python version number. For example, in CPython release 3.3 the compiled version of spam.py would be cached as

pycache/spam.cpython-33.pyc. This naming

convention allows compiled modules from different releases and different versions of Python to coexist. Python checks the modification date of the source against the compiled version to see if it’s out of date and needs to be recompiled. This is a completely automatic process. Also, the compiled modules are platform-independent, so the same library can be shared among systems with different architectures. Python does not check the cache in two circumstances. First, it always recompiles and does not store the result for the module that’s loaded directly from the command line. Second, it does not check the cache if there is no source module. To support anon-source (compiled only) distribution, the compiled module must be in the source directory, and there must not be a source module. Some tips for experts:

* You can use the -O or -OO switches on the Python command to reduce the size of a compiled module. The -O switch removes assert statements, the -OO switch removes both assert statements and doc

strings. Since some programs may rely on having these available, you should only use this option if you know what you’re doing. “Optimized”

modules have an opt- tag and are usually smaller. Future releases may change the effects of optimization.

* A program doesn’t run any faster when it is read from a .pyc file than when it is read from a .py file; the only thing that’s faster about .pyc files is the speed with which they are loaded.
* The module compile all can create .pyc files for all modules in a directory.
* There is more detail on this process, including a flow chart of the decisions

THONNY IDE:

Thonny is as mall and light weight Integrated Development Environment. It was developed to provide a small and fast IDE, which has only a few dependencies from other packages. Another goal was to be as independent as possible from a special Desktop Environment like KDE or GNOME, so Thonny only requires the GTK2 toolkit and therefore you only need the GTK2 runtime libraries installd to run it.

For compiling Thonny yourself, you will need the GTK (>= 2.6.0) libraries and header files. You will also need the Pango, Gliband ATK libraries and header files. All these files are available at [http://www.gtk.org.](http://www.gtk.org/) Furthermore you need, of course, a C compiler and the Make tool; a C++ compiler is also required for the included Scintilla library. The GNU versions of these tools are recommended.

Compiling Thonny is quite easy. The following should do it:

% ./configure

% make

% make install

The configure script supports several common options, for a detailed list, type

% ./configure --help

There are also some compile time options which can be found in src/Thonny .h. Please see Appendix C for more information. In the case that your system lacks dynamic linking loader support, you probably want to pass the option --disable-vte to the configure script. This prevents

compiling Thonny with dynamic linking loader support to automatically load libvte.so.4 if available. Thonny has been successfully compiled and tested under Debian 3.1 Sarge, Debian 4.0 Etch, Fedora Core 3/4/5, Linux From Scratch and FreeBSD 6.0. It also compiles under Microsoft Windows

At startup, Thonny loads all files from the last time Thonny was launched. You can disable this feature in the preferences dialog (see Figure 3-4). If you specify some files on the command line, only these files will be opened, but you can find the files from the last session in the file menu under the "Recent files" item. By default this contains the last 10 recently opened files. You can change the amount of recently opened files in the preferences dialog. You can start several instances of Thonny , but only the first will load files from the last session. To run a second instance of Thonny , do not specify any file names on the command-line, or disable opening files in a running instance using the appropriate command line option.

Thonny detects an already running instance of itself and opens files from the command-line in the already running instance. So, Thonny can be used to view and edit files by opening them from other programs such as a file

manager. If you do not like this for some reason, you can disable using the first instance by using the appropriate command line option

If you have installed libvte.so in your system, it is loaded automatically by Thonny , and you will have a terminal widget in the notebook at the bottom. If Thonny cannot find libvte.so at startup, the terminal widget will not be loaded. So there is no need to install the package containing this file in order to run Thonny . Additionally, you can disable the use of the terminal widget by command line option, for more information see Section3.2.You can use this terminal (from now on called VTE) nearly as an usual terminal program like xterm. There is basic clipboard support. You can paste the contents of the clipboard by pressing the right mouse button to open the popup menu and choosing Paste. To copy text from the VTE, just select the desired text and then press the right mouse button and choose Copy from the pop up menu. On systems running the X Window System you can paste the last selected text by pressing the middle mouse button in the VTE (on 2-button mice, the middle button can often be simulated by pressing both mouse buttons together).

As long as a project is open, the Make and Run commands will use the project’s settings, instead of the defaults. These will be used whichever document is currently displayed. The current project’s settings

are saved when it is closed, or when Thonny is shut down. When restarting Thonny , the previously opened project file that was in use at the end of the last session will be reopened.

Execute will run the corresponding executable file, shell script or interpreted script in a terminal window. Note that the Terminal tool path must be correctly set in the Tools tab of the Preferences dialog - you can use any terminal program that runs a Bourne compatible shell and accept the "-e" command line argument to start a command. After your program or script has finished executing, you will be prompted to press the return key. This allows you to review any text output from the program before the terminal window is closed.

By default the Compile and Build commands invoke the compiler and linker with only the basic arguments needed by all programs. Using Set Includes and Arguments you can add any include paths and compile flags for the compiler, any library names and paths for the linker, and any arguments you want to use when running Execute.

Thonny has basic printing support. This means you can print a file by passing the filename of the current file to a command which actually prints the file.

However, the printed document contains no syntax highlighting.

**CONCLUSION:**

In conclusion, the Hybrid Network Intrusion Detection System (HNIDS) offers a robust and adaptable solution for network intrusion detection in contemporary, big data-driven environments. By integrating data balancing techniques, advanced feature extraction through Convolutional Neural Networks (CNNs), and the powerful classification capabilities of Support Vector Machines (SVMs), HNIDS not only improves accuracy but also demonstrates the potential to effectively safeguard network infrastructures against a wide array of threats, providing a foundation for continuous monitoring and enhancement of network security.

**FUTURE SCOPE:**

The future scope of this research lies in the continual advancement and refinement of the Hybrid Network Intrusion Detection System (HNIDS), exploring avenues for integrating additional machine learning techniques, enhancing real-time monitoring capabilities, and adapting to emerging threats in an ever-evolving network security landscape. Additionally, HNIDS can serve as a model for further research in intrusion detection and cybersecurity, paving the way for innovative solutions to address future challenges.

**REFERENCES:**

1.Anish Halimaa A.,K. Sundarakantham,"Machine Learning Based Intrusion Detection System",2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI)

2.B. Mukherjee,L.T. Heberlein,K.N. Levitt,"Network intrusion detection",IEEE Network

3.Usman Shuaibu Musa,Megha Chhabra,Aniso Ali,Mandeep Kaur,"Intrusion Detection System using Machine Learning Techniques: A Review",2020 International Conference on Smart Electronics and Communication (ICOSEC)

4.F. Sabahi,A. Movaghar,"Intrusion Detection: A Survey",2008 Third International Conference on Systems and Networks Communications

5.Uzair Bashir,Manzoor Chachoo,"Intrusion detection and prevention system: Challenges & opportunities",2014 International Conference on Computing for Sustainable Global Development (INDIACom)

6.Usman Shuaibu Musa,Sudeshna Chakraborty,Muhammad M. Abdullahi,Tarun Maini,"A Review on Intrusion Detection System using Machine Learning Techniques",2021 International Conference on Computing Communication and Intelligent Systems (ICCCIS)

7.Muder Almi'ani,Alia Abu Ghazleh,Amer Al-Rahayfeh,Abdul Razaque,"Intelligent intrusion detection system using clustered self organized map",2018 Fifth International Conference on Software Defined Systems (SDS)

8.Ajmeera Kiran,S. Wilson Prakash,B Anand Kumar,Likhitha,Tammana Sameeratmaja,Ungarala Satya Surya Ram Charan,"Intrusion Detection System Using Machine Learning",2023 International Conference on Computer Communication and Informatics (ICCCI)

9.Sara Al-Emadi,Aisha Al-Mohannadi,Felwa Al-Senaid,"Using Deep Learning Techniques for Network Intrusion Detection",2020 IEEE International Conference on Informatics IoT and Enabling Technologies (ICIoT)

10.Loubna Dali,Ahmed Bentajer,Elmoutaoukkil Abdelmajid,Karim Abouelmehdi,Hoda Elsayed,Eladnani Fatiha,Benihssane Abderahim,"A survey of intrusion detection system",2015 2nd World Symposium on Web Applications and Networking (WSWAN)