## ABSTRACT

Benefiting from the advance of Deep Learning technology, IoT devices and systems are becoming more intelligent and multi-functional. They are expected to run various Deep Learning inference tasks with high efficiency and performance. This requirement is challenged by the mismatch between the limited computing capability of edge devices and large-scale Deep Neural Networks. Edge-cloud collaborative systems are then introduced to mitigate this conflict, enabling resource-constrained IoT devices to host arbitrary Deep Learning applications. However, the introduction of third- party clouds can bring potential privacy issues to edge computing. In this paper, we conduct a systematic study about the opportunities of attacking and protecting the privacy of edge-cloud collaborative systems. Our contributions are twofold we first devise a set of new attacks for an untrusted cloud to recover arbitrary inputs fed into the system, even if the attacker has no access to the edge device’s data or computations, or permissions to query this system. We empirically demonstrate that solutions that add noise fail to defeat our proposed attacks, and then propose two more effective defense methods. This provides insights and guidelines to develop more privacy-preserving collaborative systems and algorithms.

# INTRODUCTION

## 1.INTRODUCTION

Deep Learning (DL) and Internet of Things (IoT) technologies have advanced quickly in recent years. IoT gadgets become desirable targets for deep learning applications. They gather data and information from ambient contexts using a variety of sensors (such as cameras, microphones, and gyroscopes), run DL programs to analyze sensory input, and make control decisions. The era of Artificial Intelligence of Things (AIoT), which has profoundly altered our daily lives, is the result of the convergence of AI with IoT: Small-scale AIoT systems are being deployed to create smart homes and improve living standards; Large-scale AIoT systems can aid in the development of smart cities, while medium-sized AIoT systems are used in factories and warehouses to increase automation and efficiency.

Implementing applications for deep learning inference on commodity

edge gadgets face a number of difficulties. One way that an Internet of Things (IoT) device might gather data is through streaming, where it can do things like vehicle detection, remote monitoring, scene analysis, and application trace analysis. This necessitates rapid data analysis and DL model execution on the device. However, as sizes increase, state-of-the-art DL models get more intricate, making it Princeton University, Princeton, NJ, USA, 08540 is home to Z. He and R. Lee. Email address: rblee}zechengh}@princeton.edu.

Nanyang Technological University, Singapore, 639798 is where T. Zhang is employed.

[Tianwei.Zhang@ntu.edu.sg](mailto:Tianwei.Zhang@ntu.edu.sg) is her email.

IoT devices with limited resources are unable to meet the performance criteria. The device's constrained processing power can result in noticeable latency, and its small storage capacity makes it challenging to store a big DNN model; a critical energy consumption limitation is brought about by the limiting battery capacity.

One way to get around this problem is to use the cloud to handle the inference computation and the full DL model. The edge device receives the output from the cloud after sending the input data there. Although this can

overcome the aforementioned edge device restrictions, delivering a huge amount of raw data results in considerable connectivity expenses. In addition, there may be integrity breaches of the model if the cloud is unreliable and privacy breaches of the inference data , particularly if the input data are extremely sensitive like patient information.

#### Purpose

The purpose of this project is to analyze the vulnerabilities of Edge–Cloud collaborative inference systems concerning data privacy and security. By identifying potential attack vectors and evaluating existing protection mechanisms, this study aims to enhance the resilience of these systems against adversarial threats. Ultimately, the goal is to provide insights and guidelines for designing secure and privacy-preserving Edge–Cloud collaborative inference systems.

##### The purpose of studying and addressing data privacy in edge-cloud collaborative inference systems is multifaceted:

**Understanding Threats:** Identify potential vulnerabilities and threats to data privacy within edge-cloud collaborative systems. This includes examining possible attack vectors such as data interception, unauthorized access, and inference model extraction.

**Developing Countermeasures**: Develop robust mechanisms and protocols to protect sensitive data and ensure privacy in collaborative inference systems. This may involve encryption techniques, access control mechanisms, and secure communication protocols.

**Balancing Performance and Privacy:** Strike a balance between the performance requirements of collaborative inference systems and the need to preserve data privacy. Implement efficient privacy-preserving techniques that do not significantly degrade system performance.

**Compliance and Regulation:** Ensure that edge-cloud collaborative inference systems adhere to relevant data privacy regulations and standards. This includes compliance with laws such as GDPR (General Data

Protection Regulation) and industry-specific guidelines for handling sensitive data.

**User Trust and Confidence:** Enhance user trust and confidence in edge- cloud collaborative systems by demonstrating a strong commitment to protecting their privacy. This involves transparently communicating privacy measures and providing users with control over their data.

**Fostering Innovation:** Create an environment that fosters innovation in edge-cloud collaboration while safeguarding data privacy. Encourage the development of novel privacy-preserving techniques and frameworks that enable secure and efficient collaborative inference.

**Addressing Ethical Concerns:** Consider the ethical implications of data privacy in edge-cloud collaborative inference systems, particularly concerning the responsible use of personal data and the potential impact on individuals' privacy rights.

Overall, the purpose is to ensure that edge-cloud collaborative inference systems can effectively leverage the combined resources of edge devices and cloud infrastructure while mitigating risks to data privacy and maintaining user trust.

#### Scope

##### His research focuses on the following aspects:

Examination of the architecture and operation of Edge–Cloud collaborative inference systems.

Identification and classification of potential attack vectors targeting data privacy in such systems.

Evaluation of existing defense mechanisms and their effectiveness in mitigating privacy risks.

Proposal of novel techniques and strategies to enhance data privacy protection in Edge–Cloud collaborative inference systems.

Case studies and simulations to demonstrate the impact of attacks and the efficacy of proposed countermeasures.

##### In edge-cloud collaborative inference systems, data privacy concerns cover a number of important areas:

**Threat Landscape Analysis:** Determine and evaluate possible risks to edge-cloud collaborative inference systems' data privacy. Examining several attack vectors such as data interception, model extraction, inference privacy assaults, and unauthorized access are part of this task.

**Methods for Preserving Privacy:** Assess current methods and protocols for preserving privacy that are relevant to edge-cloud collaborative inference systems. This covers federated learning techniques, holomorphic encryption, secure multiparty computation, encryption, and differential privacy.

**System Architecture:** From a privacy standpoint, evaluate the architecture and design of edge-cloud collaborative inference systems. To find potential privacy risks, this entails looking at data flow, communication protocols, access control methods, and data storage practices.

**Security Measures:** Examine security measures in edge-cloud collaborative environments to guard against data privacy assaults.

systems of inference. This covers secure communication channels, intrusion detection/prevention systems, encryption of data while it's in transit and at rest, and authentication methods.

**Regulatory Compliance:** Take into account data privacy-related regulations and compliance frameworks, such as GDPR, HIPAA, and CCPA. Examine how edge-cloud collaborative inference systems can protect user privacy and comply with these rules.

Conduct privacy impact evaluations to determine whether implementing edge-cloud collaborative inference systems entails any potential privacy problems. This entails determining the degree of sensitivity of the data being processed, the possible influence on people's right to privacy, and mitigating measures.

Discuss data privacy issues in edge-cloud collaborative inference systems from an ethical perspective. Fairness, accountability, openness, and consent are among the values that guide data processing procedures.

Enhance user knowledge and control over data in edge-cloud collaborative inference systems by looking into possible ways to do so. This entails offering user-friendly privacy options, detailed consent methods, and transparent privacy rules.

**Performance Optimisation:** Without sacrificing data privacy, maximise the effectiveness of privacy-preserving methods in edge-cloud collaborative inference systems. Computational overhead, transmission latency, and model accuracy factors must all be balanced in this process.

Incident Response and Recovery: Create procedures for incident response and recovery in the event that edge-cloud collaborative inference systems experience possible data privacy violations. This covers protocols for identifying, addressing, and recovering from security events with the least amount of privacy harm possible.

Researchers and practitioners can effectively address the issues of exploiting and defending data privacy in edge-cloud collaborative inference by taking these aspects into consideration within the scope of their work.

**1.3 Objectives of the Proposed Work**

The objectives section highlights the key goals and outcomes you aim to achieve in your project. For "Attacking and Protecting Data Privacy in Edge-Cloud Collaborative Inference Systems," the objectives could include the following:

**Identify Privacy Vulnerabilities**:

* + The first objective is to explore and understand how sensitive data can be compromised when edge devices collaborate with cloud servers for deep learning tasks. This involves identifying the weak points in the system where data can be exposed or intercepted, even if the cloud service is not fully trusted.

**Develop New Attack Strategies**:

* + Another key objective is to design new methods for an untrusted cloud to recover data without having direct access to the edge device’s computations. These attack methods will aim to reconstruct the original data or infer sensitive information, providing insights into the real risks associated with such systems.

**Evaluate Existing Defense Mechanisms**:

* + A critical part of the project is assessing current privacy protection strategies, such as adding noise to data or using encryption. This objective will focus on testing whether these existing defenses are sufficient to protect against the new attack methods developed.

**Propose Enhanced Defense Techniques**:

* + Building on the identified weaknesses, the objective here is to propose stronger privacy protection methods that can effectively guard against the attacks. These solutions should offer better protection without compromising the performance or efficiency of the edge-cloud system.

**Empirical Validation**:

* + Finally, the objective is to validate the proposed attack and defense methods through empirical experiments. This involves testing the system with real-world data and scenarios to demonstrate the practical effectiveness of the privacy protection methods.

**1.4 Motivation**

The motivation section explains why this project is important and relevant in the current technological landscape. For "Attacking and Protecting Data Privacy in Edge-Cloud Collaborative Inference Systems," the motivation could be broken down into the following points:

**Rapid Growth of IoT and Edge Devices**:

* + With the increasing number of IoT (Internet of Things) devices, such as smart home gadgets, wearable tech, and industrial sensors, the demand for smarter, more efficient systems is rising. These devices often require deep learning capabilities to perform tasks like image recognition, predictive maintenance, or autonomous decision-making.
  + However, due to their limited computational power, many of these tasks are offloaded to cloud services. This collaboration between edge devices and cloud systems is necessary to enable complex operations while keeping the devices lightweight.

**Privacy Concerns in Edge-Cloud Systems**:

* + Although edge-cloud collaboration improves efficiency, it introduces significant privacy risks. Sensitive data, such as personal information or behavioral patterns, is often processed by third-party cloud servers, which may be untrusted or vulnerable to attacks.
  + Users are increasingly concerned about how their data is being handled and whether it can be accessed or reconstructed by unauthorized entities. Therefore, finding ways to protect data privacy in such systems is crucial.

**Lack of Effective Defense Mechanisms**:

* + Existing privacy-preserving techniques, such as adding noise or encrypting data, often fail to provide adequate protection, especially in scenarios where advanced attacks target the collaboration between edge and cloud.
  + There's a need for stronger, more robust solutions to ensure that sensitive data is protected without sacrificing the performance and efficiency of the edge-cloud system.

**Importance of Securing Emerging Technologies**:

* + As edge-cloud systems become more prevalent in sectors like healthcare, smart cities, and autonomous vehicles, securing these systems is vital to prevent potential data breaches or cyberattacks.

#### 1.5Need For System

**Real-time Data Processing:** Edge–Cloud collaborative inference systems enable efficient processing of large volumes of data in real-time, facilitating timely decision-making and action.

**Resource Optimization:** By distributing computational tasks between edge devices and cloud servers, these systems optimize resource utilization and reduce latency, thereby improving overall system performance.

**Scalability:** Edge–Cloud collaboration allows for seamless scalability, enabling the system to handle increasing workloads without compromising performance.

**Privacy Concerns:** Despite their benefits, these systems raise significant privacy concerns, as sensitive data may be exposed to unauthorized access or manipulation during transmission or processing.

**Regulatory Compliance:** With the enactment of stringent data privacy regulations such as GDPR and CCPA, ensuring compliance is crucial for organizations deploying Edge–Cloud collaborative inference systems.

**Trust and Reliability:** Addressing privacy and security concerns is essential to maintaining trust in these systems among users, stakeholders, and regulatory bodies.

In conclusion, this project aims to address the pressing need for enhancing data privacy protection in Edge–Cloud collaborative inference systems by investigating potential vulnerabilities and proposing effective countermeasures. By securing these systems against adversarial threats, organizations can harness the full potential of Edge–Cloud collaboration while safeguarding sensitive data and maintaining regulatory compliance.

#### 1.5.1Existing System

The existing Edge–Cloud collaborative inference systems typically involve the deployment of edge devices (such as sensors, smartphones, and IoT devices) that collect data, perform preliminary analysis, and then offload computationally intensive tasks to cloud servers for further processing. While this architecture offers several advantages in terms of real-time processing, resource optimization, and scalability, it also poses significant challenges to data privacy and security.

#### Disadvantages

**Data Exposure:** The transmission of raw or processed data between edge devices and cloud servers poses a risk of interception or unauthorized access, leading to potential data breaches and privacy violations.

**Centralized Processing**: Reliance on cloud servers for computation introduces a single point of failure and increases the vulnerability to attacks targeting the cloud infrastructure, compromising the privacy of the entire system.

**Latency:** Despite the optimization of resources, the latency introduced by data transmission between edge devices and cloud servers can impact the responsiveness of the system, particularly in applications requiring real- time decision-making.

**Regulatory Compliance:** Ensuring compliance with data privacy regulations such as GDPR and CCPA becomes challenging due to the distributed nature of data processing and the potential for cross-border data transfers, increasing the risk of legal liabilities.

#### Proposed System

The proposed system aims to address the aforementioned disadvantages by implementing advanced privacy-preserving techniques and decentralized data processing strategies. Key components of the proposed system include:

**Differential Privacy Mechanisms:** Integration of differential privacy techniques at the edge devices to add noise or perturbations to the data before transmission, ensuring individual privacy while preserving the overall statistical properties of the dataset.

**Homomorphic Encryption:** Utilization of homomorphic encryption schemes to perform computations on encrypted data without decrypting it, thereby protecting sensitive information even during processing on cloud servers.

**Federated Learning:** Adoption of federated learning frameworks, where machine learning models are trained collaboratively across edge devices without sharing raw data, preserving data privacy and reducing reliance on centralized servers.

**Blockchain Technology:** Implementation of blockchain-based solutions for secure and immutable data storage, ensuring data integrity and traceability while enhancing trust among participants in the Edge–Cloud collaborative inference system.

#### Advantages

**Enhanced Data Privacy:** By employing differential privacy and homomorphic encryption techniques, the proposed system ensures that sensitive information remains protected throughout the data lifecycle, mitigating the risk of data breaches and privacy violations.

**Decentralized Processing:** Leveraging federated learning and blockchain technology enables decentralized data processing, reducing dependency on centralized servers and minimizing the impact of potential attacks on the system's integrity and availability.

**Improved Regulatory Compliance:** The adoption of privacy-preserving mechanisms and transparent data management practices facilitates compliance with data privacy regulations, fostering trust among users and regulatory authorities.

**Reduced Latency:** With distributed processing capabilities at the edge, the proposed system minimizes latency by performing computations closer to the data source, enabling faster response times and enhanced real-time performance in latency-sensitive applications.

**1.6 Contribution**

The **Contribution** section outlines the key innovations and findings your project adds to the existing body of knowledge on data privacy in edge-cloud collaborative systems. For "Attacking and Protecting Data Privacy in Edge-Cloud Collaborative Inference Systems," the contributions can be summarized as follows:

1. **Novel Attack Methods**:
   * One major contribution is the development of new, sophisticated attack strategies that allow an untrusted cloud to infer sensitive data from edge devices, even without direct access to the data or computations. This helps demonstrate how vulnerable existing edge-cloud systems are, even when basic privacy measures are in place.
   * These attacks can serve as a benchmark for evaluating the privacy risks associated with different edge-cloud collaborative inference setups.
2. **Empirical Demonstration of Existing Defenses' Limitations**:
   * Another contribution is the empirical analysis that shows the inadequacy of some existing privacy protection techniques, such as noise addition. Through testing, the project highlights how these traditional methods fail to fully defend against the proposed attacks, especially when dealing with complex collaborative systems.
   * This insight provides a critical evaluation of current approaches and their shortcomings.
3. **Proposal of Enhanced Defense Mechanisms**:
   * The project introduces two new defense strategies specifically designed to combat the identified attack methods. These strategies aim to enhance privacy protection while ensuring the system maintains its performance and efficiency.
   * Unlike existing methods, these new defenses are more effective at preventing sensitive data leakage, providing a significant improvement in privacy protection for edge-cloud collaborative inference systems.
4. **Practical Guidelines for Future Systems**:
   * A key contribution is the development of guidelines and insights that can inform future designs of edge-cloud systems. These guidelines help system architects and developers implement more privacy-preserving architectures that balance security and efficiency.
   * The lessons learned from the proposed attacks and defenses offer a roadmap for improving the privacy of edge-cloud systems across different applications.

# SOFTWARE REQUIREMENT ANALYSIS AND SPECIFICATION

## SOFTWARE REQUIREMENTS ANALYSIS AND SPECIFICATION

The production of the requirements stage of the software development process is Software Requirements Specifications (SRS) (also called a requirements document). This report lays a foundation for software engineering activities and is constructing when entire requirements are elicited and analysed. SRS is a formal report, which acts as a representation of software that enables the customers to review whether it (SRS) is according to their requirements. Also, it comprises user requirements for a system as well as detailed specifications of the system requirements.

The SRS is a specification for a specific software product, program, or set of applications that perform particular functions in a specific environment. It serves several goals depending on who is writing it. First, the SRS could be written by the client of a system. Second, the SRS could be written by a developer of the system. The two methods create entirely various situations and establish different purposes for the document altogether. The first case, SRS, is used to define the needs and expectation of the users. The second case, SRS, is written for various purposes and serves as a contract document between customer and developer.

#### Characteristics of good SRS



**Fig 2.1:** Software Requirement

##### The following are the features of a good SRS document:

**Correctness:** User review is used to provide the accuracy of requirements stated in the SRS. SRS is said to be perfect if it covers all the needs that are truly expected from the system.

Definition of their responses of the software to all realizable classes of input data in all available categories of situations

**Completeness:** Full labels and references to all figures, tables, and diagrams in the SRS and definitions of all terms and units of measure.

**Consistency:** The SRS is consistent if, and only if, no subset of individual requirements is described in its conflict.

##### There are three types of possible conflict in the SRS:

The specified characteristics of real-world objects may conflict. For example,

1. The format of an output report may be described in one requirement as tabular but in another as textual.
2. One condition may state that all lights shall be green while another states that all lights shall be blue.
3. Two or more requirements may define the same real-world object but use different terms for that object. For example, a program's request for user input may be called a "prompt" in one requirement and a "cue" in another. The use of standard terminology and descriptions promotes consistency.

**Unambiguousness:** SRS is unambiguous when every fixed requirement has only one interpretation. This suggests that each element is uniquely interpreted. In case there is a method used with multiple definitions, the requirements report should determine the implications in the SRS so that it is clear and simple to understand.

**Ranking for importance and stability:** The SRS is ranked for importance and stability if each requirement in it has an identifier to indicate either the significance or stability of that particular requirement.

Typically, all requirements are not equally important. Some prerequisites may be essential, especially for life-critical applications, while others may be desirable. Each element should be identified to make these differences clear and explicit. Another way to rank requirements is to distinguish classes of items as essential, conditional, and optional.

**Modifiability:** SRS should be made as modifiable as likely and should be capable of quickly obtain changes to the system to some extent. Modifications should be perfectly indexed and cross-referenced.

**Verifiability:** SRS is correct when the specified requirements can be verified with a cost-effective system to check whether the final software meets those requirements. The requirements are verified with the help of reviews.

**Traceability:** The SRS is traceable if the origin of each of the requirements is clear and if it facilitates the referencing of each condition in future development or enhancement documentation.

#### Properties of a good SRS document

##### The essential properties of a good SRS document are the following:

**Concise:** The SRS report should be concise and at the same time, unambiguous, consistent, and complete. Verbose and irrelevant descriptions decrease readability and also increase error possibilities.

**Structured:** It should be well-structured. A well-structured document is simple to understand and modify. In practice, the SRS document undergoes several revisions to cope up with the user requirements. Often, user requirements evolve over a period of time. Therefore, to make the modifications to the SRS document easy, it is vital to make the report well- structured.

**Black-box view:** It should only define what the system should do and refrain from stating how to do these. This means that the SRS document should define the external behavior of the system and not discuss the implementation issues. The SRS report should view the system to be developed as a black box and should define the externally visible behavior of the system. For this reason, the SRS report is also known as the black- box specification of a system.

**Conceptual integrity:** It should show conceptual integrity so that the reader can merely understand it. Response to undesired events: It should characterize acceptable responses to unwanted events. These are called system response to exceptional conditions.

**Verifiable:** All requirements of the system, as documented in the SRS document, should be correct. This means that it should be possible to decide whether or not requirements have been met in an implementation.

### Related Work

1. Y. Tang, C. Zhang, R. Gu, P. Li, and B. Yang, “Vehicle detection and recognition for intelligent traffic surveillance system,” Multimedia tools and applications, vol. 76, no. 4, pp. 5817–5832, 2017.

As the main means of transportation for urban residents, the number of motor vehicles is increasing year by year. With the continuous development of society and the gradual improvement of people’s quality of life, automobiles have gradually become an indispensable means of transportation in people’s lives, resulting in increased traffic flow. However, the old traffic system is still unable to cope with the rapid growth of traffic pressure, and traffic congestion and various accidents occur frequently, which is a huge test for the contemporary intelligent traffic system. With the gradual development of society, more and more researchers are devoted to intelligent transportation systems, which make the development of target detection technology based on video image processing more and more rapid. The primary problem in embedding digital video in applications is that the complexity of video encoding and decoding far exceeds that of simple image and audio compression and decompression. Digital video can take various forms and formats. Developers need to support complex configurations and various aspects, such as different resolutions/display sizes, different bit rates, real-time issues, and even the reliability of the video source. Intelligent transportation

achieved a lot of results. However, there are still some deficiencies in precision and robustness. At the same time, the improvement of video image processing technology gives us a new idea. To further improve the intelligent traffic system, provide accurate data information for all departments, and improve the traffic situation, this study, based on video image processing technology, combined with the three-frame difference algorithm, calculates and studies the data of illegal parking at a certain intersection. The calculated false detection rates for Y2 are 1.1%, 0.9%, and 2.4%, and the leakage rates for Y1 are 2.4%, 1.9%, and 4.7%, respectively. This shows that the algorithm has high accuracy for vehicle parking detection data and can collect information quickly and effectively. Applying the algorithm to the detection of other vehicles can provide efficient services for relevant traffic departments and public security departments and relieve traffic pressure. The image processing technology is a process of analyzing and processing images through certain computer technology to achieve the desired results. The scheme in the article realizes background extraction, image filtering, image binarization, morphological transformation, vehicle detection and segmentation, shadow detection, etc.

1. G. Chen, T. X. Han, Z. He, R. Kays, and T. Forrester, “Deep convolutional neural network based species recognition for wild animal monitoring,” in 2014 IEEE International Conference on Image Processing (ICIP). IEEE, 2014, pp. 858–862.

We proposed a novel deep convolutional neural network based species recognition algorithm for wild animal classification on very challenging camera-trap imagery data. The imagery data were captured with motion triggered camera trap and were segmented automatically using the state of the art graph-cut algorithm. The moving foreground is selected as the region of interests and is fed to the proposed species recognition algorithm. For the comparison purpose, we use the traditional bag of visual words model as the baseline species recognition algorithm. It is clear that the proposed deep convolutional neural network based species recognition achieves superior performance. To our best knowledge, this is the first

attempt to the fully automatic computer vision based species recognition on the real camera-trap images. We also collected and annotated a standard camera-trap dataset of 20 species common in North America, which contains 14, 346 training images and 9, 530 testing images, and is available to public for evaluation and benchmark purpose.

1. C. Zhang, H. Li, X. Wang, and X. Yang, “Cross-scene crowd counting via deep convolutional neural networks,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2015, pp. 833– 841.

Cross-scene crowd counting is a challenging task where no laborious data annotation is required for counting people in new target surveillance crowd scenes unseen in the training set. The performance of most existing crowd counting methods drops significantly when they are applied to an unseen scene. To address this problem, we propose a deep convolutional neural network (CNN) for crowd counting, and it is trained alternatively with two related learning objectives, crowd density and crowd count. This proposed switchable learning approach is able to obtain better local optimum for both objectives. To handle an unseen target crowd scene, we present a data- driven method to finetune the trained CNN model for the target scene. A new dataset including 108 crowd scenes with nearly 200,000 head annotations is introduced to better evaluate the accuracy of cross-scene crowd counting methods. Extensive experiments on the proposed and another two existing datasets demonstrate the effectiveness and reliability of our approach.

1. L. Xiao, Y. Li, X. Huang, and X. Du, “Cloud-based malware detection game for mobile devices with offloading,” IEEE Transactions on Mobile Computing, vol. 16, no. 10, pp. 2742–2750, 2017.

As accurate malware detection on mobile devices requires fast process of a large number of application traces, cloud-based malware detection can utilize the data sharing and powerful computational resources of security servers to improve the detection performance. In this paper, we investigate the cloud-based malware detection game, in which mobile devices offload

their application traces to security servers via base stations or access points in dynamic networks. We derive the Nash equilibrium (NE) of the static malware detection game and present the existence condition of the NE, showing how mobile devices share their application traces at the security server to improve the detection accuracy, and compete for the limited radio bandwidth, the computational and communication resources of the server. We design a malware detection scheme with Q-learning for a mobile device to derive the optimal offloading rate without knowing the trace generation and the radio bandwidth model of other mobile devices. The detection performance is further improved with the Dyna architecture, in which a mobile device learns from the hypothetical experience to increase its convergence rate. We also design a post-decision state learning-based scheme that utilizes the known radio channel model to accelerate the reinforcement learning process in the malware detection. Simulation results show that the proposed schemes improve the detection accuracy, reduce the detection delay and increase the utility of a mobile device in the dynamic malware detection game, compared with the benchmark strategy.

1. F. Mireshghallah, M. Taram, P. Ramrakhyani, A. Jalali, D. Tullsen, and

H. Esmaeilzadeh, “Shredder: Learning noise distributions to protect inference privacy,” in Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems, 2020, pp. 3–18.

A wide variety of deep neural applications increasingly rely on the cloud to perform their compute-heavy inference. This common practice requires sending private and privileged data over the network to remote servers, exposing it to the service provider and potentially compromising its privacy. Even if the provider is trusted, the data can still be vulnerable over communication channels or via side-channel attacks in the cloud. To that end, this paper aims to reduce the information content of the communicated data with as little as possible compromise on the inference accuracy by making the sent data noisy. An undisciplined addition of noise can significantly reduce the accuracy of inference, rendering the service

unusable. To address this challenge, this project devises Shredder, an end- to-end framework, that, without altering the topology or the weights of a pre-trained network, learns additive noise distributions that significantly reduce the information content of communicated data while maintaining the inference accuracy. The key idea is finding the additive noise distributions by casting it as a disjoint offline learning process with a loss function that strikes a balance between accuracy and information degradation. The loss function also exposes a knob for a disciplined and controlled asymmetric trade-off between privacy and accuracy. While keeping the DNN intact, Shredder divides inference between the cloud and the edge device, striking a balance between computation and communication. In the separate phase of inference, the edge device takes samples from the Laplace distributions that were collected during the proposed offline learning phase and populates a noise tensor with these sampled elements. Then, the edge device merely adds this populated noise tensor to the intermediate results to be sent to the cloud.

**2.2 Existing Algorithms/Techniques**

This section discusses the current algorithms and techniques that are commonly used to protect data privacy in edge-cloud collaborative inference systems. It provides an overview of the methods that have been developed to secure the data and mitigate privacy risks. Here's how this could be explained:

1. **Homomorphic Encryption**:
   * Homomorphic encryption allows computations to be performed on encrypted data without needing to decrypt it first. This technique is widely used in privacy-preserving systems to ensure that sensitive data remains encrypted while being processed in the cloud.
   * **Limitation**: While this method provides strong privacy guarantees, it is computationally expensive and can significantly slow down the performance of deep learning inference tasks. This makes it less practical for real-time applications in resource-constrained edge devices.
2. **Differential Privacy**:
   * Differential privacy is a technique that adds controlled noise to data, ensuring that individual data points cannot be distinguished, even if an attacker has access to the aggregate data. It has been used to protect the privacy of data when it is shared with cloud servers.
   * **Limitation**: While differential privacy provides a good balance between data utility and privacy, the added noise can reduce the accuracy of deep learning models, especially when the noise is too large. In edge-cloud systems, this trade-off can be critical, as performance is a key requirement.
3. **Federated Learning**:
   * Federated learning allows edge devices to collaboratively train a machine learning model without sharing raw data with the cloud. Each device trains the model locally and only shares model updates (gradients or parameters) with the cloud, which aggregates the updates to improve the global model.
   * **Limitation**: Although federated learning reduces the need to send raw data to the cloud, it is still vulnerable to attacks on the shared model updates, such as model inversion attacks, where an attacker can reconstruct original data from the updates.
4. **Secure Multi-Party Computation (SMPC)**:
   * SMPC allows multiple parties (e.g., edge devices and cloud servers) to jointly compute a function over their inputs without revealing their private data to each other. It provides strong privacy guarantees in collaborative settings.
   * **Limitation**: SMPC is highly secure, but like homomorphic encryption, it is computationally intensive and can introduce significant delays in the inference process, making it challenging for real-time edge-cloud collaboration.
5. **Trusted Execution Environments (TEEs)**:
   * TEEs are hardware-based solutions that isolate sensitive computations in a secure environment, such as Intel SGX. This ensures that even if the cloud infrastructure is compromised, the data processed within the TEE remains protected.
   * **Limitation**: TEEs rely on specific hardware and have limited scalability, making them less suitable for large-scale or diverse edge-cloud systems where various hardware platforms are involved.
6. **Obfuscation Techniques**:
   * Obfuscation involves transforming data in a way that conceals its true meaning while still allowing some level of processing. This can be used in edge-cloud systems to hide sensitive information from the cloud.
   * **Limitation**: Obfuscation can degrade the quality of the data and the results of the computations, particularly in deep learning tasks where precise information is often critical for accuracy.

**2.3 Proposed Algorithms/Techniques**

In this section, you will introduce the new algorithms or techniques you are proposing to address the privacy vulnerabilities in edge-cloud collaborative inference systems. The goal is to develop methods that enhance privacy protection while maintaining system efficiency. Here’s an explanation of how this section could be structured:

1. **New Attack Strategies**:
   * **Data Reconstruction Attacks**:
     + One of the proposed techniques involves a novel attack where an untrusted cloud can attempt to reconstruct the original input data sent from the edge device by analyzing intermediate results or partial computations. This type of attack can bypass existing protections by using machine learning models trained on patterns derived from the collaborative inference process.
     + **Purpose**: To demonstrate how sensitive data can be leaked even when the cloud doesn’t have direct access to the raw input data or edge computations. This provides insight into the potential risks in current edge-cloud systems.
2. **Privacy-Leaking Gradient Inference Attack**:
   * This attack targets the gradients (model updates) shared during collaborative learning processes (such as in federated learning). By reverse-engineering these gradients, an untrusted cloud server can infer sensitive information about the original data without needing to access it directly.
   * **Purpose**: To reveal the vulnerability of sharing model updates and propose more secure methods to handle collaborative learning in edge-cloud environments.
3. **Defense Strategy 1: Split Neural Network Approach**:
   * **Description**: This defense involves splitting a deep neural network (DNN) between the edge device and the cloud. The first few layers of the DNN are executed on the edge device, while the rest of the model is processed in the cloud. The key idea is to split the network in a way that the intermediate representations (generated on the edge) do not reveal any sensitive data.
   * **Purpose**: By keeping sensitive feature extraction on the edge device and only sending abstract representations to the cloud, the risk of exposing raw data or personal information is significantly reduced. The cloud processes only non-sensitive information, which minimizes privacy risks.
   * **Advantages**: This technique provides a good balance between privacy and computational efficiency, allowing real-time processing while maintaining security.
4. **Defense Strategy 2: Hybrid Encryption with Noise Addition**:
   * **Description**: This method combines encryption techniques with noise addition to secure the data being sent to the cloud. Unlike traditional noise addition (which can reduce accuracy), this approach dynamically adjusts the level of noise based on the sensitivity of the data and the computational task. Only highly sensitive parts of the data are heavily obfuscated, while less sensitive parts are processed more freely.
   * **Purpose**: This hybrid technique addresses the limitations of basic noise addition by ensuring that privacy is maintained without excessively degrading model performance or accuracy.
   * **Advantages**: It provides strong protection against data reconstruction attacks while maintaining the performance of the deep learning inference system, making it suitable for real-time applications.
5. **Dynamic Partitioning Based on Data Sensitivity**:
   * **Description**: This algorithm dynamically adjusts which parts of the computation are performed on the edge device versus the cloud, based on the sensitivity of the data being processed. Sensitive data is kept on the edge, while non-sensitive tasks are offloaded to the cloud.
   * **Purpose**: The dynamic nature of this method ensures that only the minimal necessary amount of sensitive information is processed in the cloud, reducing the risk of data leakage.
   * **Advantages**: By dynamically partitioning tasks, this approach offers flexibility, allowing the system to adjust to different types of data and varying privacy needs, ensuring both security and efficiency.
6. **Federated Learning with Enhanced Privacy**:
   * **Description**: In this technique, the proposed enhancement to federated learning involves adding secure aggregation protocols to the model updates. This ensures that even if the cloud has access to the aggregated updates from multiple devices, it cannot infer sensitive data from any individual device.
   * **Purpose**: To make federated learning more secure by preventing any potential privacy leakage from model updates.
   * **Advantages**: This technique maintains the collaborative nature of federated learning while improving the privacy of individual devices.

**Summary of Advantages of Proposed Techniques:**

* **Higher Security**: The proposed attack and defense mechanisms provide a deeper understanding of privacy risks and offer stronger protections than existing methods.
* **Efficiency**: These techniques are designed to maintain or minimally impact the performance of the system, making them suitable for real-time applications where both speed and security are essential.
* **Scalability**: The proposed solutions can be scaled to accommodate large edge-cloud systems without significantly increasing the computational burden on the edge devices or the cloud

### 2.4 Software Requirements

Operating system : Windows10 Ultimate Coding Language : Java

Back-End : Django-ORM

Designing : Html, css, javascript

Data Base : MySQL (WAMP Server)

### Hard Ware Requirements

Processor : I3 or higher

Speed : 2.9 GH

RAM : 4 GB (min)

Hard Disk : 160 GB

### Functional Requirements

An integrated land management information system is a comprehensive software solution designed to facilitate efficient and effective management of land-related activities.

Here are some functional requirements that are typically expected from such a system:

**Land Parcel Management:** Capture, store, and manage information about land parcels, including boundaries, ownership details, and legal descriptions. Enable the creation and maintenance of updated land parcel data. Support spatial data management, allowing the integration of maps and geographic information system (GIS) data.

**Land Registration and Title Management:** Provide functionality for registering and managing land titles, including registration of new titles, transfers, and encumbrances. Generate unique identifiers for land parcels and titles to ensure accurate and reliable tracking. Facilitate searches and inquiries related to land titles and ownership.

**Planning and Zoning:** Support land use planning and zoning processes, allowing the definition and enforcement of zoning regulations. Enable the management of development permits and applications. Facilitate the review and approval process for zoning changes, subdivisions, and land development projects.

**Property Tax Assessment and Billing:** Calculate and manage property tax assessments based on assessed property values, exemptions, and applicable tax rates. Generate and distribute property tax bills to property owners. Track tax payments and provide reporting capabilities for tax revenue management.

**Land Use Compliance and Enforcement:** Monitor compliance with land use regulations and enforce zoning codes. Track violations and enable the management of enforcement actions, such as citations, fines, and hearings. Provide reporting and analytical tools to monitor compliance trends and identify areas of non-compliance.

**Document Management:** Store and manage a variety of land-related documents, such as deeds, surveys, contracts, and permits. Enable efficient search, retrieval, and sharing of documents. Ensure document version control and security to maintain data integrity and confidentiality.

**Reporting and Analytics:** Generate standard and customizable reports, such as land ownership reports, tax assessment summaries, and compliance status reports. Provide analytical tools for data analysis, trend identification, and decision support. Support integration with business intelligence tools for advanced reporting and visualization.

**Integration and Interoperability:** Integrate with external systems and data sources, such as GIS systems, financial systems, and public records databases. Support data exchange standards to enable interoperability with other government agencies and stakeholders. Provide APIs or web services for seamless integration with third-party applications.

**User Management and Security:** Implement user roles and permissions to control access to system functionalities and data. Ensure data security through encryption, authentication, and authorization mechanisms. Maintain audit trails and logs for system activities and changes.

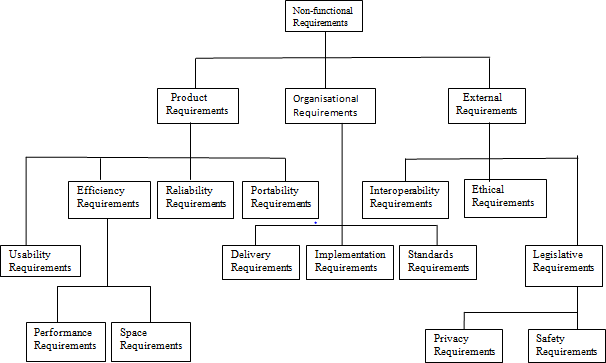
**Mobile and Web Access:** Provide mobile and web-based interfaces to enable remote access and field operations. Support data collection in the field, such as property inspections and survey data capture. Ensure responsive design and user-friendly interfaces for ease of use on different

devices. These functional requirements serve as a starting point for an integrated land management information system. The specific requirements may vary based on the organization's needs and local regulations. It is important to conduct a detailed analysis and consultation with stakeholders to refine and customize the requirements for the intended implementation.

### Non-Functional Requirements

Non-functional requirements describe user-visible aspects of the system that are not directly related to the functionality of the system. Non-functional requirements are Constraints on the services or functions offered by the system.

* + - Accuracy
    - Performance
    - Scalability
    - Reliability
    - Security
    - Usability
    - Availability
    - Data integrity



**Fig 2.8.1:** Non-Functional Requirements

Non-functional requirements are often called qualities of a system. Other terms for non-functional requirements are "constraints", "quality attributes", "quality goals", "quality of service requirements" and "non- behavioral requirements". Qualities that are non-functional requirements. It can be divided into two main categories

Execution qualities, such as security and usability, which are observable at run time.

Evolution qualities, such as testability, maintainability, extensibility, and scalability, which are embodied in the static structure of the software system.

**Accuracy:** The prediction system should strive to provide accurate forecasts of stock market trends. It should minimize errors and consistently generate reliable predictions.

**Performance:** The system should be able to handle large volumes of data and perform complex calculations efficiently. It should provide timely predictions, allowing users to make informed decisions in a timely manner.

**Scalability:** As the amount of data and user demand increases, the system should be capable of scaling up to handle the additional load without sacrificing performance or accuracy.

**Reliability:** The prediction system should be highly reliable and available. It should be able to handle unexpected failures and recover gracefully to minimize downtime or disruption in generating predictions.

**Security:** Since stock market predictions involve sensitive financial data, the system should implement robust security measures to protect the confidentiality, integrity, and availability of the data. It should comply with relevant security standards and regulations.

**Usability :**As it is an Internet Application, must have some usability Features. End users of this System are Unlimited and from Various Skilled groups so we can’t restrict them. By providing some facilities we have to make them comfortable.

* + - The colors that we use in this Web Portal design must be attractive.
    - Easy Navigations are Preferable to do any task.
    - The home page Should be a Centralized System (Screen/Window) to go to any
    - feature and to get any result.
    - The facility to return to the Home page from any page Should be available.
    - Labels of all Objects in the entire system Must be Understandable.

**Availability:** Due to the time-sensitive nature of financial markets, the availability needs for predicting stock market trends utilizing machine learning and deep learning algorithms is crucial. The system should attempt

to meet availability needs such as real-time updates, Load Balancing, Monitoring, and Alerts to ensure continuous availability and timely predictions.

**Data Integrity:** When predicting stock market movements with machine learning and deep learning algorithms, data integrity is critical. Inaccurate or distorted data might result in untrustworthy projections and perhaps disastrous financial decisions. To ensure data integrity, the system must meet requirements such as data validation, data cleaning, and timestamp accuracy. The system can ensure that the data utilised to predict stock market movements is reliable, accurate, and reliable by following to certain data integrity requirements. In the dynamic and competitive stock market environment, maintaining data integrity is critical for producing valuable insights and making informed financial decisions.

#### SDLC Methodologies

SDLC stands for Software Development Life Cycle. A Software Development Life Cycle is essentially a series of steps, or phases, that provide a model for the development and lifecycle management of an application or piece of software. The intent of an SDLC process is to help produce a product that is cost-efficient, effective, and of high quality.

The SDLC methodology usually contains the following stages:

1. Requirement Gathering Stage
2. Analysis
3. Designing
4. Coding
5. Testing
6. Maintenance

**Requirements Gathering Stage**

The requirements gathering process takes as its input the goals identified in the high-level requirements section of the project plan. Each goal will be relined into a set of one or more requirements. These requirements define

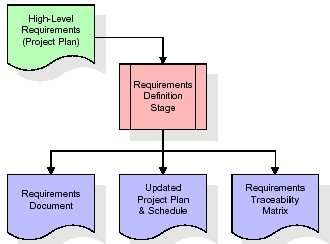
the major functions of the intended application, define operational data areas and reference data areas, and define the initial data entities. Major functions include critical processes to be managed, as well as mission critical inputs, outputs and reports. A user class hierarchy is developed and associated with these major functions, data areas, and data entities. Each of these definitions is termed a Requirement. Requirements are identified by unique requirement identifiers and, at minimum, contain a requirement title and textual description.

These requirements are fully described in the primary deliverables for this stage, the Requirements Document and the Requirements Traceability Matrix (RTM). The requirements document contains complete descriptions of each requirement, including diagrams and references to external documents as necessary. Note that detailed listings of database tables and fields are not in the Requirements document.

Requirement Gathering or commonly known as the Discovery Phase is basically a process in which we understand and identify a business’s project technical requirements and proceed with a well-defined plan. Although the discovery phase is an essential phase in any critical project plan, it is quite often overlooked with the absence of sufficient ground work.

Requirements gathering are the process of identifying your project’s exact requirements from start to finish. This process occurs during the [project](https://asana.com/resources/project-initiation) [initiation](https://asana.com/resources/project-initiation) phase but you’ll continue to manage your project requirements throughout the entire [project timeline](https://asana.com/resources/create-project-management-timeline-template).

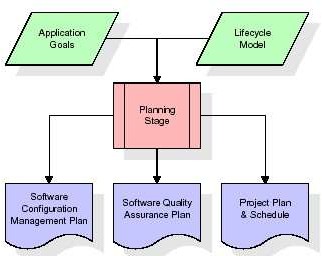
The title of each requirement is also placed into the first version of the RTM, along with the title of each goal from the project plan. The purpose of the RTM is to show that the product components developed during each stage of the software development lifecycle are formally connected to the components developed in prior stages. The outputs of the requirements definition stage include the requirements document, the RTM, and an updated project plan.



**Fig 2.8.2:** Requirements Gathering Stage

#### Analysis Stage

The planning stage establishes a bird's eye view of the intended software product, and uses this to establish the basic project structure, evaluate feasibility and risks associated with the project, and describe appropriate management and technical approaches

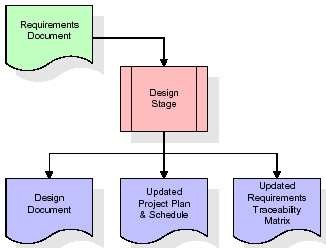


**Fig 2.8.3:** Analysis Stage

The most critical section of the project plan is a listing of high-level product requirements, also referred to as goals. All of the software product requirements to be developed during the requirements definition stage flow from one or more of these goals. The minimum information for each goal consists of a title and textual description, although additional information and references to external documents may be included. The outputs of the project planning stage are the configuration management plan, the quality assurance plan, and the project plan and schedule, with a detailed listing of scheduled activities for the upcoming Requirements stage, and high-level estimates of effort for the out stages.

#### Designing Stage

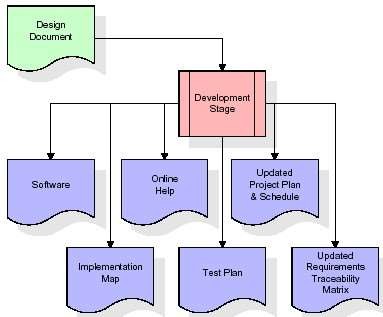
The design stage takes as its initial input the requirements identified in the approved requirements document. For each requirement, a set of one or more design elements will be produced as a result of interviews, workshops, and/or prototype efforts. Design elements describe the desired software features in detail, and generally include functional hierarchy diagrams, screen layout diagrams, tables of business rules, business process diagrams, pseudo code, and a complete entity-relationship diagram with a full data dictionary. These design elements are intended to describe the software in sufficient detail that skilled programmers may develop the software with minimal additional input.



**Fig 2.8.4:** Designing Stage

##### Development Stage

The development stage takes as its primary input the design elements described in the approved design document. For each design element, a set of one or more software artifacts will be produced. Software artifacts include but are not limited to menus, dialogs, and data management forms, data reporting formats, and specialized procedures and functions. Appropriate test cases will be developed for each set of functionally related software artifacts, and an online help system will be developed to guide users in their interactions with the software.



**Fig 2.8.5:** Development Stage

#### Testing

During the integration and test stage, the software artifacts, online help, and test data are migrated from the development environment to a separate test environment. At this point, all test cases are run to verify the correctness and completeness of the software. Successful execution of the test suite confirms a robust and complete migration capability. During this stage, reference data is finalized for production use and production users are identified and linked to their appropriate roles. The final reference data and production user list are compiled into the Production Initiation Plan.

The spiral model was defined by Barry Boehm in his 1988 article, “A spiral Model of Software Development and Enhancement. This model was not the first model to discuss iterative development, but it was the first model to explain why the iteration models.

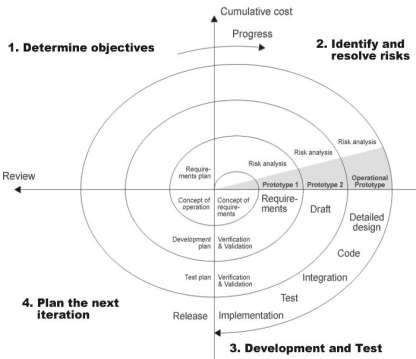
As originally envisioned, the iterations were typically 6 months to 2 years long. Each phase starts with a design goal and ends with a client reviewing

the progress thus far. Analysis and engineering efforts are applied at each phase of the project, with an eye toward the end goal of the project.

The steps for Spiral Model can be generalized as follows

* The new system requirements are defined in as much details as possible. A preliminary design is created for the new system.
* A first prototype of the new system is constructed from the preliminary design. This is usually a scaled-down system, and represents an approximation of the characteristics of the final product.
* A second prototype is evolved by a fourfold procedure:
* Evaluating the first prototype in terms of its strengths, weakness, and risks.
* Defining the requirements of the second prototype.
* Planning and designing the second prototype.
* Constructing and testing the second prototype.
* At the customer option, the entire project can be aborted if the risk is deemed too great. Risk factors might involved development cost overruns, operating-cost miscalculation, or any other factor that could, in the customer’s judgment, result in a less-than-satisfactory final product.
* The existing prototype is evaluated in the same manner as was the previous prototype, and if necessary, another prototype is developed from it according to the fourfold procedure outlined above.
* The preceding steps are iterated until the customer is satisfied that the refined prototype represents the final product desired.
* The final system is constructed, based on the refined prototype.
* The final system is thoroughly evaluated and tested. Routine maintenance is carried on a continuing basis to prevent large scale failures and to minimize down time.

The following diagram shows how a spiral model acts like



**Fig 2.8.5:** Spiral Model

#### Advantages

* + High amount of risk analysis
  + Good for large and mission-critical projects.
  + Software is produced early in the software life cycle.

# SYSTEM DESIGN

## SYSTEM DESIGN

Software design sits at the technical kernel of the software engineering process and is applied regardless of the development paradigm and area of application. Design is the first step in the development phase for any engineered product or system. The designer’s goal is to produce a model or representation of an entity that will later be built. Beginning, once system requirements have been specified and analyzed, system design is the first of the three technical activities design, code, and test that is required to build and verify software.

The importance can be stated with a single word “Quality”. Design is the place where quality is fostered in software development. The design providesus with representations of software that can assess quality. Design is the only way that we can accurately translate a customer’s view into a finished software product or system. Software design is a foundation for all the following software engineering steps. Without a strong design, we risk building an unstable system one that will be difficult to test, one whose quality cannot be assessed until the last stage.

During design, progressive refinement of data structure, program structure, and procedural details are developed reviewed, and documented. System design can be viewed from either a technical or project management perspective. From the technical point of view, design is comprised of four activities like architectural design, data structure design, interface design, and procedural design.

#### Data Design (Use ER-Model)

Database Designing is a part of the development process. In the linear development cycle, it is used during the system requirements phase to construct the data components of the analysis model. This model represents the major data objects and the relationship between them. It should not be confused with data analysis, which takes place in the system design phase. As in a DFD, a model of data consists of a number of symbols joined up

according to certain conventions. System designers describe these conceptual modeling using symbols from a modeling method known as entity relationship analysis.

The data pertaining to proposed system is voluminous that careful design of the database must proceed storing the data in the database. A database management system provides flexibility in storing and retrieving of data and production of information. The RDBMS is a bridge between the application programs which determine what data is needed and how they are processed and the operating system of the computer, which is responsible for placing database that a specific program will use.

#### Entity Relationship Diagram

Entity relationship analysis uses three major abstractions to describe data. The Entities (which are distinct things in the enterprise), Relationships (which are meaningful interactions between the objects) and Attributes (which are the properties of the entities and relationship). The entity Relationship Diagram (ERD) depicts the relationship between the data objects. The ERD is the notation that is used to conduct the date modeling activity the attributes of each data object noted is the ERD can be described resign a data object descriptions.

The primary purpose of the ERD is to represent data objects and their relationships. The relative simplicity and pictorial clarity of this diagramming technique may well account in large part for the widespread use of ER model. The relation upon the system is structure through a conceptual ER-Diagram, which not only specifics the existential entities but also the standard relations through which the system exists and the cardinalities that are necessary for the system state to continue.

The entity Relationship Diagram (ERD) depicts the relationship between the data objects. The ERD is the notation that is used to conduct the date modeling activity the attributes of each data object noted is the ERD can be

described resign a data object descriptions. The primary purpose of the ERD is to represent data objects and their relationships. The set of primary components that are identified by the ERD are

* Data object
* Relationships
* Attributes
* Various types of indicators.

#### ER Diagram Components

Rectangle, it represents the entity set.

Ellipse, it represents attributes. Diamond, it represents relationship sets.

Lines, which link attributes to entity sets and. entity sets to relationships.

.

Double Ellipse, represents multivalve attributes.

#### Mapping Cardinalities

It expresses the number of entities to which another entity can be associated via a relationship. For binary relationship sets between entity sets A and B, **The mapping cardinality must be one of the following:**

**One-to-One:** An entity in A is associated with at most one entity in B, and an entity in B is associated with at most one entity in A.

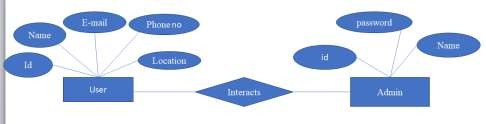
**One-to-Many:** An entity in A is associated with any number in B. An entity in B is associated with any number in A.

**Many-to-Many:** Entities in A and B are associated with any number from each other.

**Cardinality:** It indicates that which type of relationship the business rule follows is called cardinality.

**Connectivity:** It specifies which type of notation the entities are connected in both sides one side or many side.

### ER Diagrm



**Fig 3.1.1:** ER Diagram for Overall System

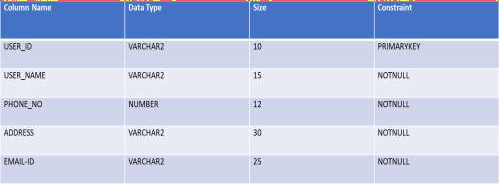
#### Data Dictionary

The logical characteristics of current systems data stores, including name, description, aliases, contents, and organization, identifies processes where the data are used and where immediate access to information is required, Serves as the basis for identifying database requirements during system design.

#### Uses of Data Dictionary

* + - To manage the details in large systems.
    - To communicate a common meaning for all system elements.
    - To Document the features of the system.
    - To facilitate analysis of the details in order to evaluate characteristics and
    - determine where system changes should be made
    - To locate errors and omissions in the system.

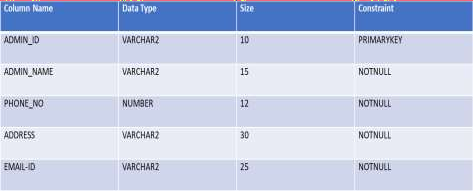
#### Table name: User



**Table 3.2.1:** User

**Description:** This table shows the user data

#### Table name: Admin



**Table 3.2.2:** System

**Description:** This table shows the System

### UML Design

UML is an abbreviation for Unified Modelling Language. The UML has established itself as the software blueprint language of choice for analysts, designers, and programmers alike. The UML provides a consistent vocabulary for everyone involved in software design, from business analysts to designers to programmers. The unified modelling language enables a software engineer to represent an analysis model using modelling notation, which is governed by a set of syntactic, semantic, and paradigm rules. A UML system is represented by five separate perspectives, each of

which describes the system from a distinct point of view. Each view is specified by the following set of diagrams.

#### User Model View

This view represents the system as seen by the user. The analytical depiction depicts a usage scenario as seen by the end user.

#### Structural Model View

The data and functionality in this model are obtained from within the system. The static structures are represented by this model view.

#### Behavioural Model View

It represents the interactions of collection between various structural elements described in the user model and structural model view, representing the dynamic of behavioural as system parts.

#### Implementation Model View

The structural and behavioural components of the system are depicted as they will be built in this model.

#### Environmental Model View

This represents the structural and behavioural aspects of the environment in Which the system will be deployed.

#### Basic Building Blocks of UML

The UML vocabulary includes three types of building blocks.

* Things
* Relationships
* Diagrams

Objects are first-class citizens in a model; relationships connect them; and diagrams group interesting collections of objects.

#### Things in UML

In UML, there are four types of things..

* Structural Things.
* Behavioral Things.
* Grouping Things.
* Annotational Things.

These are the fundamental object-oriented building pieces of UML..

#### Structural Things

The nouns of UML models are structural objects. These are the primarily static aspects of a model that represent either conceptual or physical elements. There are seven different types of structural things.

#### Class

A class is a description of a group of objects with similar features, operations, relationships, and semantics. A class implements one or more interfaces. A class is represented graphically as a rectangle, which often includes its name, attributes, and operations.

|  |
| --- |
| Windows |
| Origin size |
| Open() Close() Move()  Display() |

**Sample Class**

#### Interface

An interface is a set of operations that define a class or component's service. As a result, an interface explains the element's externally observable behaviour. An interface may represent the entire behaviour of a class or component, or merely a portion of it. An interface is represented graphically as a circle with its name.



**Sample Interface**

#### Collaboration

Collaboration is a society of roles and other aspects that work together to generate some cooperative behaviour that is more than the sum of the elements. As a result, collaborations have both structural and behavioural dimensions. Collaboration is represented visually as an ellipse with dashed lines, usually including merely its name.

Chain of responsibility

#### Use Case

**Sample Collaboration**

A use case is a description of a series of actions that a system takes that results in an observable result of value to a specific actor. A use case is used to organize the behavioural aspects of a model.

Place order

**Sample Use Case**

#### Active Class

Active classes, components, and nodes are all class-like, which means they define a set of objects that have the same properties, operations, relationships, and semantics. However, these three are sufficiently distinct

and required for modelling specific parts of an object-oriented system to deserve special consideration.

An active class is one whose objects possess one or more processes or threads and so have the ability to initiate control activities. An active class is similar to a class in that its objects represent elements whose behaviour is concurrent with the behaviour of other components. An active class is rendered graphically in the same way as a class, but with heavier lines, usually featuring its name, properties, and operations.

|  |
| --- |
| Event Manager |
|  |
| Suspend() Flush() |

**Sample Active class**

#### Component

A component is a physical and replaceable portion of a system that conforms to and implements a set of interfaces. There are various types of deployment components in a system, such as Java Beans, as well as components that are artifacts of the development process, such as source code files. A component is usually the physical packaging of logical pieces such as classes, interfaces, and collaborations. A component is represented graphically as a rectangle with tabs, usually containing only its name.

Order for m.java

**Sample Component**

#### Node

A node is a physical element that exists at run time and represents a computing resource, typically with some memory and, more often than not, processing capabilities. A collection of components may reside on a node and may migrate from node to node. A node is represented graphically as a cube, with simply its name displayed.

Server

##### Node

Classes, interfaces, collaborations, use cases, active classes, components, and nodes are the seven main structural elements that can be included in a UML model. Variations on these seven include actors, signals, and utilities (class types), processes and threads (active class types), and applications, documents, files, libraries, pages, and tables.

#### Behavioral Things

Behaviours are dynamic components of UML models. These are the verbs of a model that depicts behaviour across time and space. In general, there are two types of behavioural things.

To begin, an interaction is a behaviour that consists of a set of messages sent among a set of objects in a certain context to accomplish a specific goal. An interaction might specify the behaviour of an object society or the behaviour of an individual operation. Other components of an interaction include messages and action sequences. Messages are graphically represented as directed lines that almost always include the name of the operation.

##### Display Message

Second, a state machine is a behaviour that specifies the sequence of states an object or an interaction goes through during its lifetime in response to events, as well as its responses to those events. A state machine also includes states, transitions (the flow from one state to another), events, and activities (the response to a transition). A state is typically represented graphically as a rounded rectangle that includes its name and substates.

Waiting

##### State

Interactions and state machines are the basic behaviour aspects that can be included in a UML model. These elements are typically semantically linked to various structural elements, including classes, collaborations, and objects.

#### Grouping Things

The original components of UML models are grouping things. These are the boxes that can be used to breakdown a model. Overall, there is just one type of grouping thing, and that is packages.

A package is a generic approach for grouping things into k8groups. A package can contain structural items, behavioural items, and even other grouping items. A package, unlike components (which exist at run time), is solely conceptual. A package is represented graphically as a tabbed folder, generally containing merely its name and, occasionally, its contents.

Business rules

**Package**

#### Annotational Things

Notational elements are the explanatory components of a UML model. These are the comments that you can use to describe, enlighten, and comment on any element in a model. A note is the most basic type of annotational object. A note is merely a symbol for attaching rendering limitations and comments to an element or group of elements. A graphic note is shown as a rectangle with a dog-eared corner and a textual or graphical statement.

Return of self

##### Note

This is the only fundamental annotational element that can be included in a UML model. Notes are often used to embellish diagrams with constraints or comments that are best stated in text, whether casual or formal.

#### Relationships in UML

In UML, there are four types of relationships.

* + Dependency
  + Association
  + Generalization
  + Realization

These are the fundamental relational building pieces of the UML. They are used to create well-formed models.

#### Dependency

To begin, dependence is a semantic relationship between two objects in which a change to one of them (the independent thing) might impact the semantics of the other (the dependent thing). Dependency is shown graphically as a dashed line, possibly directed, and occasionally with a label.



**Dependency**

#### Association

Second, an association is a structural relationship that describes a collection of links, each of which is a connection between things. Aggregation is a type of association that represents a structural relationship between a whole and its constituent pieces. An association is shown graphically as a solid line, possibly directed, occasionally with a label, and frequently with extra adornments such as multiplicity and role names.

**Employer Employee Association**

#### Generalization

Finally, a generalization is a specialization/generalization connection in which objects of the specialized element (the child) are interchangeable with objects of the generalized element (the parent). In this way, the youngster adopts the parent's structure and conduct. A generalization relationship is represented graphically as a solid line with a hollow arrowhead pointing to the parent.

**Generalization**

#### Realization

Fourth, a realization is a semantic link between two classifiers in which one classifier specifies a contract that the other classifier guarantees to fulfill. Realization relationships can be found between interfaces and the classes or components that realize them, as well as between use cases and the collaborations that realize them. A realization relationship is shown graphically as a cross between a generalization and a dependency relationship.

##### Realization

These four elements are the fundamental relational elements that can be included in a UML model. Variations of these four include refinement, trace, include, and extended (for dependencies).

UML diagrams are graphical representations of a collection of items that are often rendered as a connected graph of vertices (objects) and arcs (relationships). Diagrams are used to visualize a system from various perspectives, so a diagram provides an omitted view of the elements that comprise a system. The same element may occur in all diagrams, a subset of diagrams, or none at all. A diagram can theoretically contain any combination of things and relationships.

#### Diagrams in UML

A diagram is a graphical representation of a collection of items, which is typically represented as a connected network of vertices (objects) and arcs (relationships). Diagrams are used to visualize a system from various perspectives, so a diagram provides an omitted view of the elements that comprise a system. The same element may occur in all diagrams, a subset of diagrams, or none at all. A diagram can theoretically contain any combination of things and relationships.

However, in practice, a small number of common combinations emerge that are consistent with the five most relevant viewpoints that compose the architecture of a software intensive system. As a result, the UML includes nine such diagrams.

* Use Case Diagram
* Class Diagram
* Object Diagram
* Sequence Diagram
* Collaboration Diagram
* Component Diagram
* Deployment Diagram
* State chart Diagram
* Activity Diagram

#### Use Case Diagram

Use Case diagrams are one of five diagrams in the UML for modelling the dynamic features of systems. The other four diagrams in the UML for modelling the dynamic aspects of systems are activity diagrams, sequence diagrams, status chart diagrams, and collaboration diagrams. Use Case diagrams are essential for modelling the behaviour of a system, subsystem, or class.

Each one depicts a certain collection of use cases, actors, and interactions.

#### Class Diagram

Class diagrams are the most commonly used diagrams in the modelling of object-oriented systems. A class diagram depicts the links between classes, interfaces, and collaborations. A class diagram is a graphical representation of vertices and arcs.

#### Object Diagram

An object diagram depicts a group of things as well as their relationships. Object diagrams are used to depict data structures, which are static

snapshots of instances of the items seen in class diagrams. Object diagrams, like class diagrams, address the static design or static process view of a system, but from the standpoint of real or archetypal cases.

#### Collaboration Diagram

An interaction diagram depicts an interaction, which consists of a set of objects and their relationships, as well as the messages that can be sent among them. Interaction diagrams are used to model the system's dynamic features.

#### Sequence Diagram

A sequence diagram is an interaction diagram that emphasizes message time ordering. A sequence diagram is a graphical representation of a table with items grouped along the X-axis and messages sorted in increasing time along the Y-axis.

#### Activity Diagram

An Activity Diagram is essentially a flow chart that depicts the flow of control from one activity to the next. They are used to simulate the system's dynamic features. They can also be used to simulate the flow of an object as it transitions from one state to another at various points in the control flow.

#### State Chart Diagram

A state machine appears in a state chart diagram. State chart diagrams are used to model the system's dynamic features. The majority of the time, this entails simulating the behaviour of the reactive objects. A reactive object is one whose behaviour is best defined by its reaction to events that occur outside of its context.

#### Component Diagram

The component diagrams for the selected component package, as well as new entries in this list, always include an entry for the component package's main component diagram.

#### Deployment Diagram

Processors, devices, and connections are depicted in a deployment diagram. Each model has a single deployment diagram that depicts the relationships between its processors and devices, as well as the process allocation to processors.

#### Use Case Diagram

Use Case diagrams are one of five diagrams in the UML for modelling the dynamic features of systems. The other four diagrams in the UML for modelling the dynamic aspects of systems are activity diagrams, sequence diagrams, status chart diagrams, and collaboration diagrams. Use Case diagrams are essential for modelling the behaviour of a system, subsystem, or class.

Each one depicts a set of use cases, actors, and relationships.



**USER**

**ADMIN**

**pre process the data**

**Login**

**view model performance**

**upload dataset**

**Take dataset**

**Registration**

**Testing the data**

**Predict the result**

**Traing the model**

**View Result**

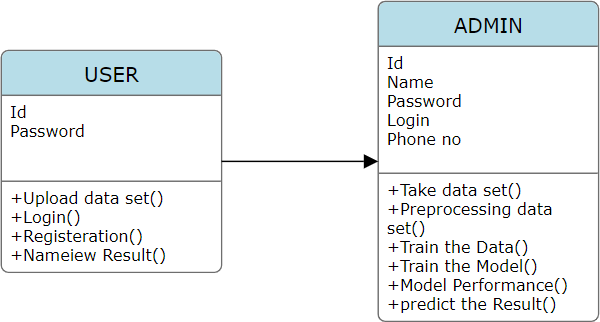
**Fig 3.3.1:** Use Case Diagram for Overall System

#### Class Diagram

The UML class diagram also referred to as object modeling, is the main static analysis diagram. Object modeling is the process by which the logical objects in the problem space are represented by the actual objects in the program. These diagrams show a set of classes, interfaces, and collaborations and their relationships. These diagrams address the static design view of a system. This view primarily supports the functional requirements of a system – the services the system should provide to its end users. A class diagram is a collection of static modeling elements, such as classes and their relationships, connected as a graph to each other and to their contents.

##### Class diagrams commonly contain the following things:

* Classes.
* Interfaces.
* Collaborations.
* Dependency, Generalization, and association relationships.



**Fig 3.3.2**: Class Diagram for Overall System

#### Sequence Diagram

It is an interaction diagram that emphasizes the time ordering of messages. A sequence diagram shows objects participating in the interaction by their lifetime and the messages that they exchange/arranged in the time sequence.

#### Description

It contains the following elements:

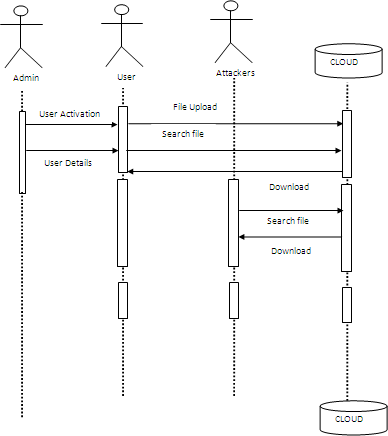
* **Object:** It is represented as a horizontal rectangle.
* **Object Lifeline:** It represents the existence of an object at a particularinstance of time and is represented as
* **The focus of control:** It is a tall, thin rectangle that shows the period oftime during which an object is performing an action.



##### Focus of control

* **Messages:** It is communication between objects, shown as a horizontalsolid ,arrow from one object to another object.In the below diagram

##### Messages

It is a type of interaction diagram that shows the interaction between a set of objects. The way they are linked to each other and also the message.

**Fig 3.3.3:** Sequence Diagram for Overall System

#### Activity Diagram

An Activity diagram illustrates the dynamic nature of a system by modeling the flow of control from activity to activity. An activity represents an operation on some class in the system that results in a change in the state of the system. Typically, activity diagrams are used to model workflow or business processes and internal operations. Because an activity diagram is a special kind of state chart diagram, it uses some of the same modeling conventions.

##### Basic Activity Diagram Symbols and NotationsAction states

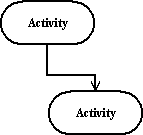
Action states represent the non-interruptible actions of objects. You can draw an action state in Smart Draw using a rectangle with rounded corners.



**Notations**

#### Action Flow

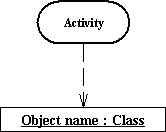
Action flow arrows illustrate the relationships among action states.



Action Flow

#### Object Flow

Object flow refers to the creation and modification of objects by activities. An object flow arrow from an action to an object means that the action creates or influences the object. An object flow arrow from an object to an action indicates that the action state uses the object.



**Object Flow**

#### Initial State

A filled circle followed by an arrow represents the initial action state.



**Initial State**

#### Final State

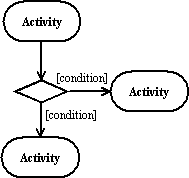
An arrow pointing to a filled circle nested inside another circle represents the final action state.



**Final State**

#### Branching

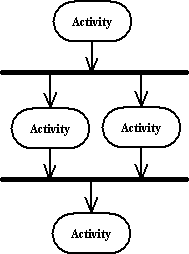
The outgoing alternates should be labeled with a condition or guard expression. You can also label one of the paths "else."



**Branching**

#### Synchronization

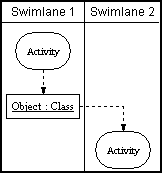
A synchronization bar helps illustrate parallel transitions. Synchronization is also called for king and joining.



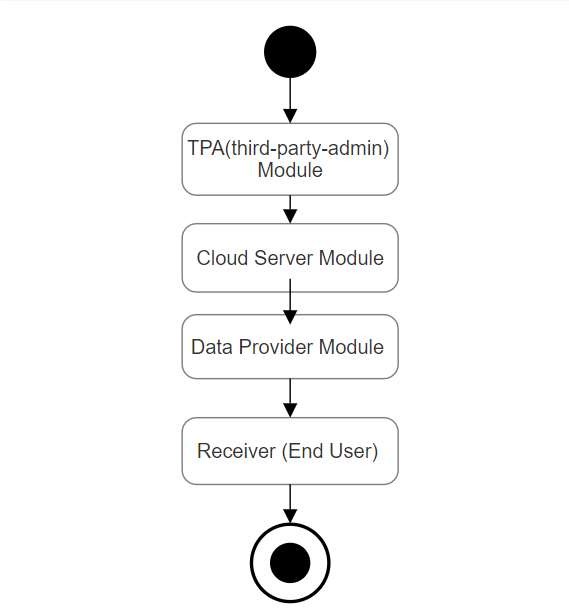
**Synchronization**

#### Swim lanes

Swim lanes group related activities into one column.



##### Swim lanes



**Fig 3.3.4:** Activity Diagram for Overall System

#### Collaboration Diagram:

The collaboration diagram is used to depict the relationship of objects in a system. The sequence and cooperation diagrams both display the same information, but in different ways. It illustrates the architecture of the object living in the system rather than the flow of messages because it is based on object-oriented programming. An object has various characteristics. Multiple things in the system are linked to one another. The collaboration diagram, also known as a communication diagram, is used to depict the architecture of an object in the system.

The collaboration diagram is used to show the relationship between the objects in a system. Both the sequence and the collaboration diagrams represent the same information but differently. Instead of showing the flow of messages, it depicts the architecture of the object residing in the system as it is based on object-oriented programming.

An object consists of several features. Multiple objects present in the system are connected to each other. The collaboration diagram, which is also known as a communication diagram, is used to portray the object's architecture in the system.

Operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.

#### Deployment Diagram:

Deployment diagrams are used to visualize the topology of the physical components of a system, where the software components are deployed. Deployment diagrams are used to describe the static deployment view of a system. Deployment diagrams consist of nodes and their relationships. The term Deployment itself describes the purpose of the diagram. Deployment diagrams are used for describing the hardware components, where software components are deployed. Component diagrams and deployment diagrams are closely related.

The deployment diagram visualizes the physical hardware on which the software will be deployed. It portrays the static deployment view of a system. It involves the nodes and their relationships.

It ascertains how software is deployed on the hardware. It maps the software architecture created in design to the physical system architecture, where the software will be executed as a node. Since it involves many nodes, the relationship is shown by utilizing communication paths.



**Fig 3.3.6:** Deployment Diagram for Overall System

# TESTING

## TESTING

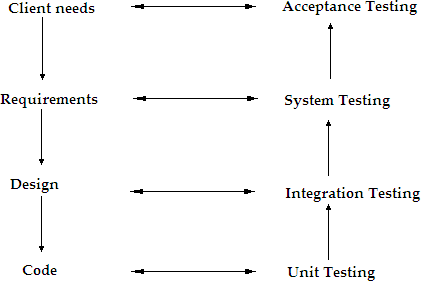
The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product. It is the process of exercising software 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 tests. Each test type addresses a specific testing requirement Testing is one of the most important phases in the software development activity.

In the software development life cycle (SDLC), the main aim of the testing process is the quality; the developed software is tested against attaining the required functionality and performance. During the testing process, the software is worked with some particular test cases and the output of the test cases is analysed whether the software is working according to the expectations or not.

The success of the testing process in determining the errors mostly depends upon the test case criteria, for testing any software we need to have a description of the expected behaviour of the system and a method of determining whether the observed behaviour confirmed the expected behaviour.

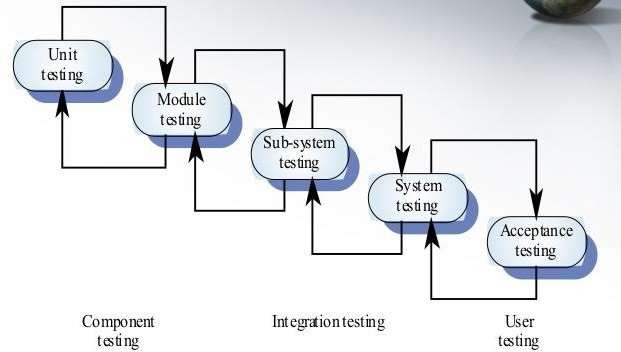
#### Levels of Testing

Since the errors in the software can be injured at any stage. So, we have to carry out the testing process at different levels during the development. The basic levels of testing are Unit, Integration, System, and Acceptance Testing



**Fig 4.0.1:** Levels of Testing

Unit Testing is carried out on coding. Here different modules are tested against the specifications produced during the design for the modules. In the case of integration testing different tested modules are combined into sub-systems and tested in case of the system testing the full software is tested and in the next level of testing the system is tested with user requirements.



**Fig 4.0.2:** Testing Approach

#### Unit Testing

Unit testing mainly focused first in the smallest and low-level modules, proceeding one at a time. Bottom-up testing was performed on each module. As developing a driver program, that tests modules by developed or used. But for the purpose of testing, modules themselves were used as stubs, to print verification of the actions performed. After the lower level modules were tested, the modules that in the next higher level those make use of the lower modules were tested. Each module was tested against required functionally and test cases were developed to test the boundary values.

#### Integration Testing

Is a systematic technique for constructing the program structure, while at the same time conducting tests to uncover errors associated with interfacing as the system consists of the number of modules the interface to be tested was between the edges of the two modules. The software tested under this was incremental bottom-up approach. Bottom-up approach integration strategy was implemented with the following steps.

* Low-level modules were combined into clusters that perform
* specificsoftware sub functions.
* The clusters were then tested.

#### System Testing

Is a series of different tests whose primary purpose is to fully exercise the computer-based system? It also tests to find discrepancies between the system and its original objective, current specifications.

#### Acceptance Testing

Acceptance Testing has performed by the customer to whether or not of accepts the delivery of the system.

#### Test Plan

Test plan is a general document for entire project, which defines the scope, approach to be taken and the personal responsible for different activities of testing. The inputs for forming test plan are following.

* Project plan.
* Requirements document.
* System design.

#### Test Case Specification

Although there is one test plan for entire project test cases have to be specified separately for each test case. Test case specification gives for each item to be tested. All test cases and outputs expected for those test cases.

#### Test Case Execution and Analysis

The steps to be performed for executing the test cases are specified in separate document called test procedure specification. This document specifies any specify requirements that exist for setting the test environment and describes the methods and formats for reporting the results of testing.

#### Test Case Approach

Testing can be done in two ways.

* + - * + Bottom up Approach.
        + Top down Approach.

#### Bottom up Approach

Testing can be performed starting from smallest and lowest level modules and proceeding one at a time. For each module in bottom up testing a short k2program executes the module and provides the needed data so that the module is asked to perform the way it will when embedded within the larger system.

#### Top down Approach

This type of testing starts from upper level modules. Since the detailed activities usually performed in the lower level routines are not provided stubs are written. A stub is a module shell called by upper level module and that when reached properly will return a message to the calling module indicating that proper interaction occurred.

#### Testing

The technique of testing without having any knowledge of the interior workings of the application is Black Box testing. The tester is oblivious to the system architecture and does not have access to the source code. Typically, when performing a black box test, a tester will interact with the system's user interface by providing inputs and examining outputs without knowing how and where the inputs are worked upon. An approach to testing where the program is considered as a ‘black-box’ The program test cases are based on the system specification is the testing process in which a tester can perform testing on an application without having any internal structural knowledge of the application.

|  |  |
| --- | --- |
| Advantages | Disadvantages |
| Well-suited and efficient for large code segments. | Limited Coverage since only a selected number of test scenarios are actually  performed. |
| Code Access is not required. | Inefficient testing is due to the fact that the tester only has limited knowledge about an application. |

**Table 4.0.1:** Black Box Testing

#### White Box Testing

Is the testing process in which tester can perform testing on an application with having internal structural knowledge? All independent paths have been exercised at least once. All logical decisions have been exercised on their true and false sides. All loops are executed at their boundaries and within their operational. To follow the concept of white box testing, tested each form have created independently to verify that data flow is correct, all conditions are exercised to check their validity, all loops are executed on their boundaries. White box testing is the detailed investigation of internal logic and structure of the code.

|  |  |
| --- | --- |
| **Advantages** | **Disadvantages** |
| As the tester has knowledge of the source code, it becomes very easy to find out which type of data can help in testing the  application effectively. | Due to the fact that a skilled tester is needed to perform white box testing, the costs are increased. |
| It helps in optimizing the code. | Sometimes it is impossible to look into every nook and corner to find out hidden errors that may create problems as many paths will tested. |
| Extra lines of code can be removed which can bring in hidden defects. | It is difficult to maintain white box testing as the use of specialized tools like code analysers, Debugging tools are required. |

**Table 4.0.2:** White Box Testing

#### Gary Box Testing

Gary Box testing is a technique to test the application with limited knowledge of the internal workings of an application. In software testing, the term the more you know the better carries a lot of weight when testing an application. Mastering the domain of a system always gives the tester an edge over someone with limited domain knowledge. Unlike black box testing, where the tester only tests the application's user interface, in grey box testing, the tester has access to design documents and the database. Having this knowledge, the tester is able to better prepare test data and test scenarios.

#### Functional Testing

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

#### 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.

#### 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.

#### Integration Testing

Testing is done for each module. After testing all the modules, the modules are integrated and testing of the final system is done with the test data, specially designed to show that the system will operate successfully in all its aspects conditions. Thus, the system testing is a confirmation that all is correct and an opportunity to show the user that the system works. The purpose of integration testing is to verify functional, performance and

reliability requirements placed on major design items. Simulated usage of shared data areas and inter-process communication is tested and individual subsystems are exercised through their input interface. Test cases are constructed to test that all components within assemblages interact correctly, for example across procedure calls or process activations, and this is done after testing individual modules, i.e. unit testing. The overall idea is a "building block" approach, in which verified assemblages are added to a verified base which is then used to support the integration testing of further assemblages.

#### Top-down Integration

Top down integrations is an incremental approach for construction of program structure. Modules are integrated by moving downward through the control hierarchy, beginning with the main control program. Modules subordinate to the main program are incorporated in the structure either in the breath-first or depth-first manner.

#### Bottom-up Integration

This method as the name suggests, begins construction and testing with atomic modules i.e., modules at the lowest level. Because the modules are integrated in the bottom up manner the processing required for the modules subordinate to a given level is always available and the need for stubs is eliminated.

#### Validation Testing

At the end of integration testing software is completely assembled as a package. Validation testing is the next stage, which can be defined as successful when the software functions in the manner reasonably expected by the customer. Reasonable expectations are those defined in the software requirements specifications. Information contained in those sections form a basis for validation testing approach.

#### System Testing

System testing is actually a series of different tests whose primary purpose is to fully exercise the computer-based system. Although each test has a different purpose, all work to verify that all system elements have been properly integrated to perform allocated functions.

#### Security Testing

Attempts to verify the protection mechanisms built into the system.

#### Performance Testing

This method is designed to test runtime performance of software within the context of an integrated system.

#### Basis Path Testing

Established technique of flow graph with Cyclamates complexity was used to derive test cases for all the functions. The main steps in deriving test cases were. Use the design of the code and draw correspondent flow graph the main it as follows. Determine the Cyclamates complexity of resultant flow graph, using formula. V (G) =E-N+2 or V (G) =P+1 or V (G)=Number of Regions

Where V (G) is Cyclometer complexity E is the number of edges

N is the number of flow graph nodes P is the number of predicate nodes

This type of testing ensures that

* All independent paths have been exercised at least once.
* All logical decisions have been exercised on their true and false sides.
* All loops are executed at their boundaries and within their operational bounds.
* All internal data structures have been exercised to assure their validity.
* To follow the concept of white box testing we have tested each form. we have created independently to verify that Data flow is correct, all conditions are exercised to check their validity, all loops are executed on their boundaries.

#### Conditional Testing

In this part of the testing each of the conditions were tested to both true and false aspects. And all the resulting paths were tested. So that each path that may be generate on particular condition is traced to uncover any possible errors.

#### Data Flow Testing

This type of testing selects the path of the program according to the location of definition and use of variables. This kind of testing was used only when some local variable was declared. Type of testing.

#### Loop Testing

In this type of testing all the loops are tested to all the limits possible. The following exercise was adopted for all loops.

* All the loops were tested at their limits, just above them and just below them.
* All the loops were skipped at least once.
* For nested loops test the inner most loop first and then work outwards.
* Unstructured loops were resolved into nested loops or concatenated loops and tested as above.

#### Alpha Testing

For this project alpha testing is carried out by the customer within the organization along with the developer. The Alpha tests are conducted in a controlled manner.

#### Beta Testing

Beta testing has performed by selecting group customers. The developer is not present at the site and the user will inform the problems that are encountered. When in future problems are reported they are rectified by software developer.

#### Functional Testing

In Functional Testing test cases are decided slowly on the basis of requirements of the program or module and the internals of the program or modules are not considered for selection of test cases.

#### Structural Testing

In Structural Testing test cases are generated on actual code of the program or module to be tested. This is called White Box Testing.

#### Testing Process

A number of activities must be performed for testing software. Testing starts with test plan. Test plan identifies all testing related activities that need to be performed along with the schedule and guide lines for testing.

### 4.1Test Cases

Creating a test case table for smart traffic light control using artificial intelligence involves defining various scenarios and expected outcomes to verify the system's functionality. Here's an example of a test case table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  | **Status** |
| **Test** | **Input** | **Expected Behavior** | **Observed** | **P =** |
| **S.No.** |  |  | **behavior** | **Passed** |
|  |  |  |  | **F =** |
|  |  |  |  | **Failed** |
| 1 | Login as user or admin with correct login details | Administrator or user Home page for manager should be displayed | -do- | P |
| 2 | Login as user or admin with wrong login details | Error message should be displayed | -do- | F |
| 3 | Upload file | Uploaded will to be encrypted using IBE | -do- | P |
| 4 | Download File | Download file using IBE | -do- | P |

**Table 4.1.1:** Test Cases for the overall system

In this table, each test case has a unique ID, a description of the scenario, inputs (sensor data, signals, etc.), expected outcomes, and a status column to mark if the test case passed or not. This table can be expanded and customized based on the specific functionalities and features of the smart traffic light control system being tested.

# IMPLEMENTATION

## IMPLEMENTATION

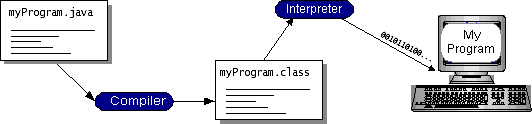
### Java Technology

Java technology is both a programming language and a platform. The Java Programming Language:

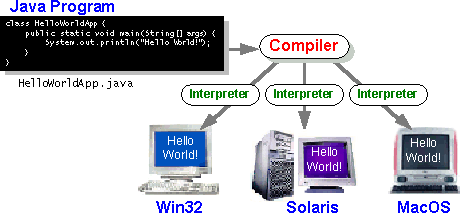
The Java programming language is a high-level language that can be characterized by all of the following buzzwords:

* Simple
* Architecture neutral
* Object oriented
* Portable
* Distributed
* High performance
* Interpreted
* Multithreaded
* Robust
* Dynamic
* Secure

With most programming languages, you either compile or interpret a program so that you can run it on your computer. The Java programming language is unusual in that a program is both compiled and interpreted. With the compiler, first you translate a program into an intermediate language called Java byte codes —the platform-independent codes interpreted by the interpreter on the Java platform. The interpreter parses and runs each Java byte code instruction on the computer. Compilation happens just once; interpretation occurs each time the program is executed. The following figure illustrates how this works.



**Fig 5.0.1:** Working of Java Program

If we think of Java byte codes as the machine code instructions for the Java Virtual Machine (Java VM). Every Java interpreter, whether it’s a development tool or a Web browser that can run applets, is an implementation of the Java VM. Java byte codes help make “write once, run anywhere” possible. You can compile your program into byte codes on any platform that has a Java compiler. The byte codes can then be run on any implementation of the Java VM. That means that as long as a computer has a Java VM, the same program written in the Java programming language can run on Windows 2000, a Solaris workstation, or on an iMac.

**Fig 5.0.2:** Implementation of Java Virtual Machine

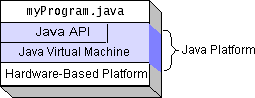
##### The Java Platform

A *platform* is the hardware or software environment in which a program runs. We’ve already mentioned some of the most popular platforms like Windows 2000, Linux, Solaris, and MacOS. Most platforms can be described as a combination of the operating system and hardware. The Java platform differs from most other platforms in that it’s a software-only platform that runs on top of other hardware-based platforms.

##### The Java platform has two components:

* The Java Virtual Machine (Java VM)
* The Java Application Programming Interface (Java API)

You’ve already been introduced to the Java VM. It’s the base for the Java platform and is ported onto various hardware-based platforms.



**Fig 5.0.3**: Program Running on the Java Platform

Native code is code that after you compile it, the compiled code runs on a specific hardware platform. As a platform-independent environment, the Java platform can be a bit slower than native code. However, smart compilers, well-tuned interpreters, and just-in-time byte code compilers can bring performance close to that of native code without threatening portability.

##### Feasibility Study Technical Feasibility

GUI is developed using HTML to capture the information from the customer. HTML is used to display the content on the browser. It uses TCP/IP protocol. It is an interpreted language. It is very easy to develop a page/document using HTML some RAD (Rapid Application Development) tools are provided to quickly design/develop our application. So many objects such as button, text fields, and text area etc are provided to capture the information from the customer.

##### Economical Feasibility

The economical issues usually arise during the economical feasibility stage are whether the system will be used if it is developed and implemented, whether the financial benefits are equal are exceeds the costs. The cost for developing the project will include cost conducts full system investigation, cost of hardware and software for the class of being considered, the benefits in the form of reduced costs or fewer costly errors. The project is economically feasible if it is developed and installed. It reduces the work load. Keep the class of application in the view, the cost of hardware and software is considered to be economically feasible.

##### Operational Feasibility

In our application front end is developed using GUI. So it is very easy to the customer to enter the necessary information. But customer must have some knowledge on using web applications before going to use our application.

### Installation of java:

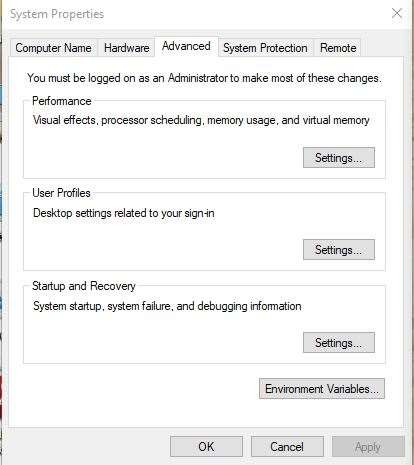
* + Go to [http://www.oracle.com/technetwork/java/javase/downloads](http://www.oracle.com/technetwork/java/javase/downloads/index.html)

[/index.html](http://www.oracle.com/technetwork/java/javase/downloads/index.html).

* + click on JDK DOWNLOAD button. run the exe file and then follow the instruction given in wizard.

##### To set up the path:-

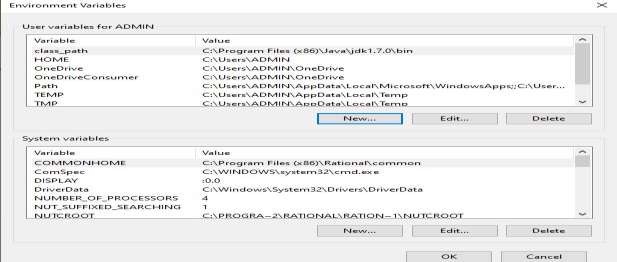
* + Right click on my pc and then go to my properties



**Fig 5.0.4:** properties wizard

Go to advanced settings and then click on environment variables

create a class path and copy the path of the java folder where it is located in program files.



**Fig 5.0.5:** path setting for java

### Installation and setup of Apache Tomcat:

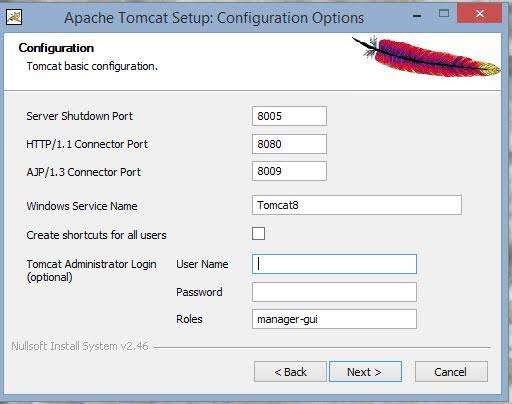
* + Go to <http://tomcat.apache.org/index.html> and click on download latest versions.
  + Run the exe file and click on next and follow the wizard instructions.



**Fig 5.0.6:** Welcome Page of Tomcat

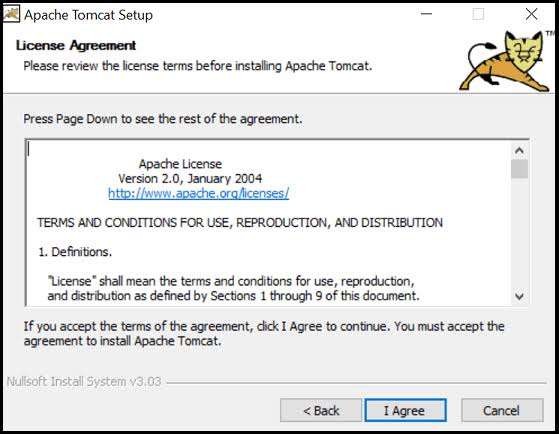
* + Click on install with port number 8090 with username and password as **aits** and **aits.**

Mention the connection port as 8090 and then click on next and finally click on finish.



**Fig 5.0.7:** Tomcat Configuration Options Page

Click on I agree button in. license agreement in order to accept the terms and condition.



**Fig 5.0.8:** Tomcat License Agreement

### Installation and setup of SQL:

* + Go to <http://dev.mywql.com/downloads/> . and click on install button.
  + After completion of installation, click on exe file and then click on next.
  + Run the MySQL setup and click on next and follow the instruction in wizard.



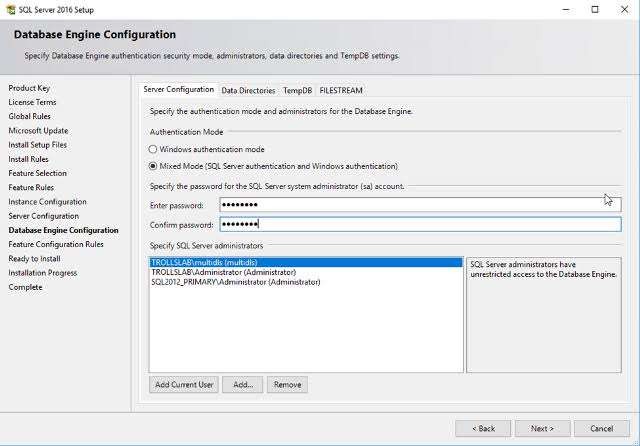
**Fig 5.0.9:** Welcome wizard of MySQL

Conform the type as typical and then click on next and follow the instructions.



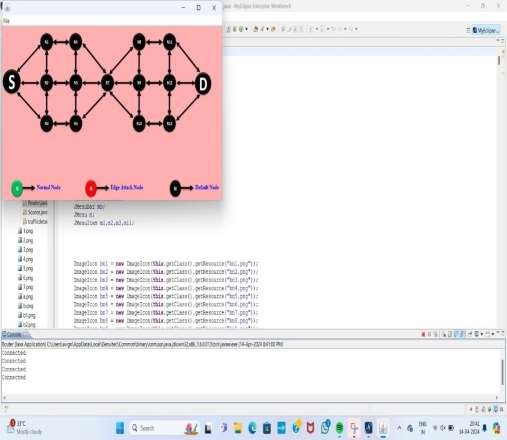
**Fig 5.0.10:** SQL setup Wizard

Now confirm the password as root in system settings field and then click on finish.



**Fig 5.0.11:** Database Configuration Engine

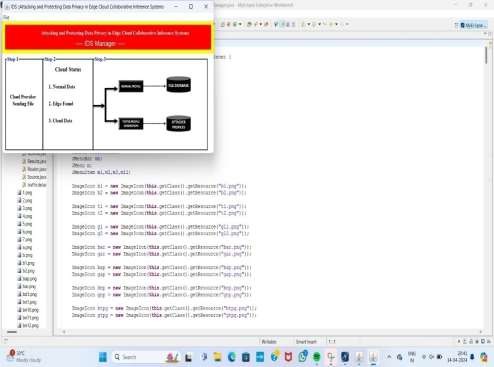
### Sample Screens



`**Screen 5.1.1:** Home Page

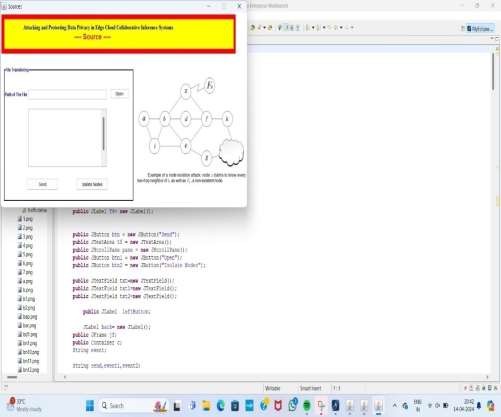
**Description:** In above screen we will open my ecplise software and then click on package explorer and select attacking and protecting data privacy and then click on src and then click default packager it will display the files click the router program and then run it will display the above result

\



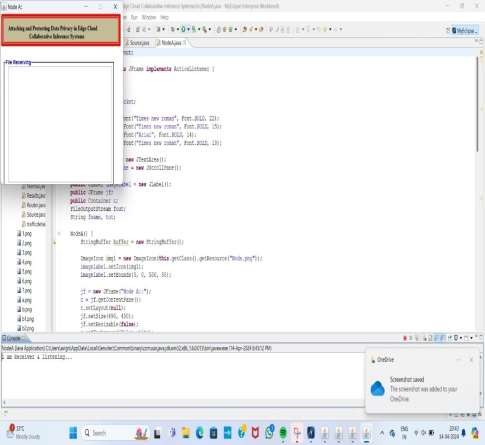
**Screen 5.1.2:** IDS Manager

**Description:** In the above screen click the IDS manager and then run the program it will display the above result



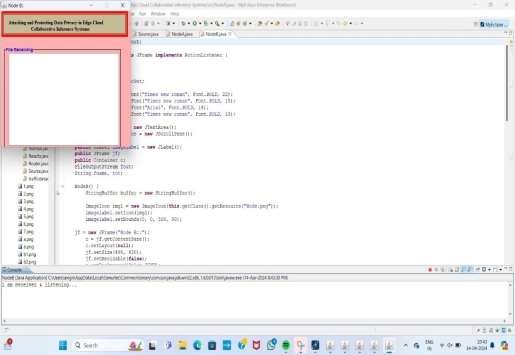
**Screen 5.1.3:** File Transferring page

**Description:** In the above screen click on source program and then run it will open the file transfering page



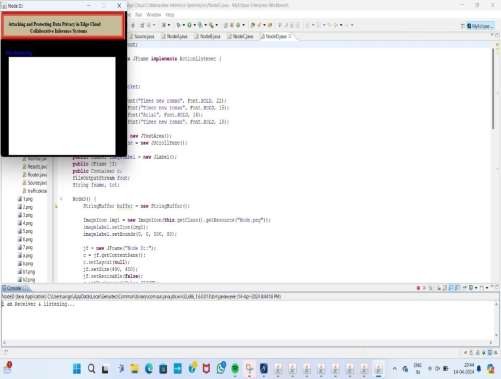
**Screen 5.1.4:** File Sent for Receiver

**Description:** Click on node A program and then run it will display the result that is file transfer to receiver



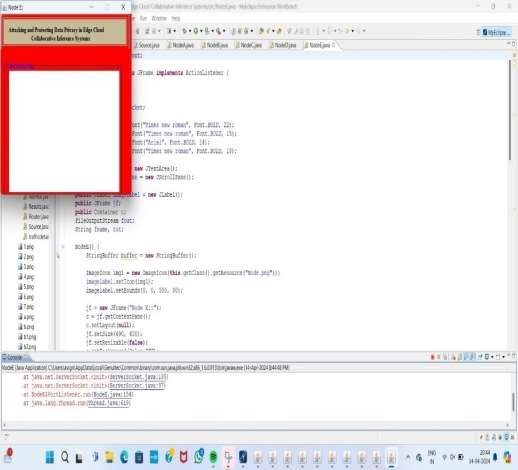
**Screen 5.1.5:** File Receiving Page

**Description:** Click on node B program and then run it will display the result that is file transfer to receiver



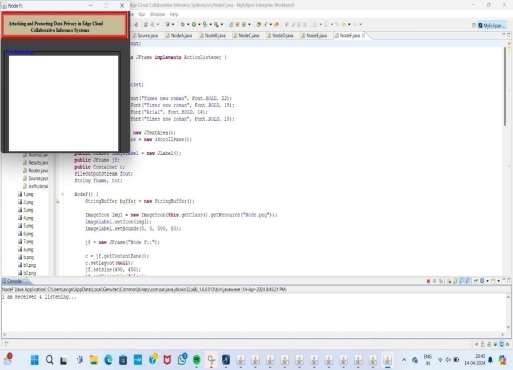
**Screen 5.1.6:** File Receiving with Default Node

**Description:** Click on node D program and then run it will display the result that is file transfer to receiver upto node A to nodeF



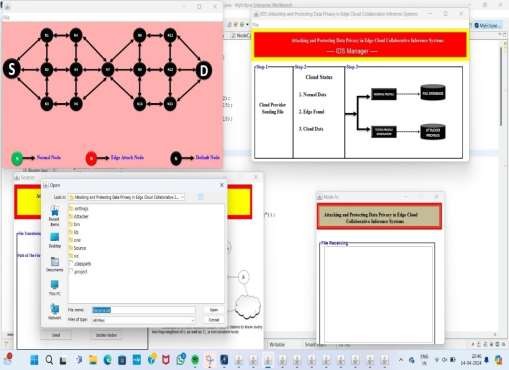
**Screen 5.1.7:** File Receive with Edge Attack Node

**Description:** Click on node E program and then run it will display the result that is file receive with edge attack node



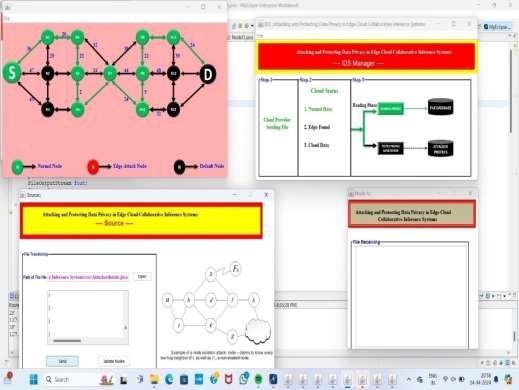
**Screen 5.1. 8:** After Receive file Listening

**Description:** Click on node F program and then run it will display the result that is After receive the file listening



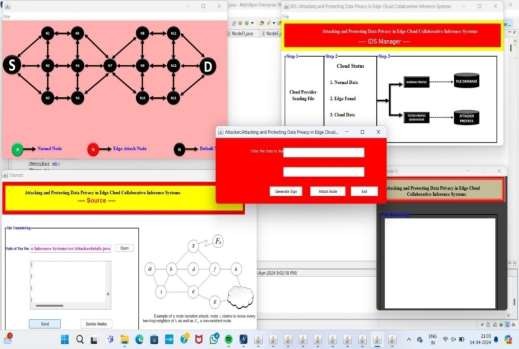
**Screen 5.1.9:** Selection of File to send from IDS to Receiver

**Description:** It will display all the screens and selection of file to send from IDS to receiver



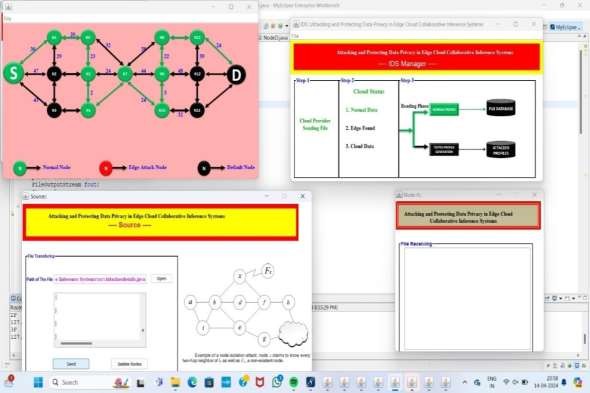
**Screen 5.1.10:** File Transferring from Source to Destination Using Normal Node

**Description:** It will display the output screen, file transfering from source to destination using normal node



**Screen 5.1.11:** Enter data for attack details

**Description:** In the above screen I will enter the data for attack details



**Screen 5.1.12:** Page Showing File Transferred from source to Destination

**Description:** Finally attacker will attacking the node. File is transferred from source to destination

# CONCLUSION

## CONCLUSION

Finally, I draw the conclusion In this project we explore the inference data privacy threats in edge-cloud collaborative systems. We discover that, an untrusted cloud can easily recover the inference samples from intermediate values. We propose a set of new attack techniques to compromise the inference data privacy under different attack settings. We demonstrate that the adversary can successfully and reliably recover the inputs with very few prerequisites. We also propose several methods to protect the inference data privacy for edge computing. Previous work all focus on the performance, efficiency and functionalities of Artificial Intelligence of Things, while ignoring privacy. We hope that this study can raise awareness about the importance of inference data privacy protection in edge-cloud systems, and encourage the balancing of privacy protection with usability when designing or implementing such systems.

**6.1 Future Enhancements**

The **Future Enhancements** section outlines potential areas where the project could be expanded or improved in the future. For "Attacking and Protecting Data Privacy in Edge-Cloud Collaborative Inference Systems," the future enhancements could include the following:

1. **Advanced Federated Learning Techniques**:
   * **Improvement**: While federated learning is already used to enhance privacy, future work could explore more advanced versions, such as **federated learning with differential privacy** or **secure multi-party computation** to further protect model updates.
   * **Impact**: This would strengthen the defense mechanisms by ensuring that even if the cloud has access to aggregated data from multiple devices, it cannot infer sensitive information about individual devices.
2. **Enhancing Scalability for Large Networks**:
   * **Improvement**: Future work could focus on optimizing the system for larger networks of edge devices. This could involve developing more scalable partitioning algorithms, which would allow for efficient processing and privacy protection across many devices and more complex collaborative tasks.
   * **Impact**: This would allow the system to handle massive numbers of IoT devices in smart cities, healthcare, and other large-scale applications without compromising on privacy or performance.
3. **Integration with Trusted Execution Environments (TEEs)**:
   * **Improvement**: Integrating the system with **trusted execution environments (TEEs)**, such as Intel SGX, would provide an additional layer of hardware-based security. TEEs could be used to secure sensitive computations on the cloud side without needing complex encryption.
   * **Impact**: This could offer more robust privacy protection by isolating sensitive computations from the rest of the cloud infrastructure, enhancing trust in edge-cloud collaboration.
4. **Real-Time Adaptive Privacy Mechanisms**:
   * **Improvement**: Implement real-time, adaptive privacy mechanisms that dynamically adjust the level of privacy protection based on the type of data being processed and the current system conditions. This could include **context-aware privacy protection**, where the system assesses the sensitivity of data in real-time and adjusts the processing accordingly.
   * **Impact**: This enhancement would provide more flexibility, allowing the system to offer stronger protection for highly sensitive data while maintaining speed and efficiency for less sensitive tasks.
5. **Privacy in Decentralized Edge Systems**:
   * **Improvement**: Explore privacy techniques for **decentralized edge-cloud systems**, where multiple clouds or fog nodes participate in the computation, and data is distributed across a network of collaborating entities. This could involve developing more secure data-sharing protocols in environments where trust levels vary between participants.
   * **Impact**: Such a system would be more resilient to attacks and could work in environments where full trust in a single cloud provider is not possible, improving both security and collaboration.
6. **Energy-Efficient Privacy Preservation**:
   * **Improvement**: Focus on **energy-efficient privacy-preserving techniques** for edge devices, which typically have limited battery life and computational power. Techniques like lightweight encryption, optimized noise addition, or more efficient model partitioning could be developed.
   * **Impact**: This would extend the battery life of IoT devices while maintaining robust privacy protection, making the system more practical for real-world deployments in areas such as smart homes or remote monitoring.

#### 7.References

* + Java Crash Course 2nd Edition - this is a basic level book for beginners.
  + Learning java 5th Edition - this book is a practical learning book for basic to advanced level.
  + Java Cookbook - this book for advanced programmer interested in learning about modern java development tools.
  + Automating Boring Stuff With Java - In this book you will learn to write programs in java.
  + Head First Java - this book covered the fundamental of java.
  + Think Java - the basics of programming concepts and cover advanced topics like data structure and object-oriented design.

### 7.1 Publications

The **Publications** section provides a list of academic papers, journals, and other scholarly sources that have been referenced or contributed to the foundation of your project. These publications provide background information, support for the proposed methods, and insights into the current state of research in the field. For a project like "Attacking and Protecting Data Privacy in Edge-Cloud Collaborative Inference Systems," the publications listed here would focus on topics like edge computing, cloud privacy, collaborative inference, and privacy-preserving techniques.

**URL listing**

* + [**www.google.co.in**](http://www.google.co.in/)
  + [**www.java.org**](http://www.java.org/)
  + [**www.w3schools.com**](http://www.w3schools.com/)
  + [**www.javatutorial.com**](http://www.javatutorial.com/)