



**National Forensic Sciences  
University  
Delhi Campus**  
(An Institute of National Importance)

## **MINOR PROJECT REPORT**

**ON**

# **“AI-Powered Real-Time DNS Threat Detection Using Large Language Models”**

**Submitted To**

**School of Cyber Security & Digital Forensics  
National Forensic Sciences University**  
For partial fulfillment for the award of degree

**B. Tech - M. Tech.**

**In**

**Computer Science & Engineering (Cyber Security)**

**Submitted By**

**Aviral Kaintura  
(102CTBMCS2122016)**

**Under the Supervision of**

**Dr. Abinash Mishra  
School of Cyber Security & Digital  
Forensics**

**National Forensic Sciences University,  
Delhi Campus, New Delhi – 110085, India**

**December 2024**



## DECLARATION

I hereby declare that the thesis entitled “**AI-Powered Real-Time DNS Threat Detection Using Large Language Models**” is a research work done by me and no part of the thesis has been presented earlier and will be presented for any degree, diploma, or similar title at any other institute/university.

**Aviral Kaintura**  
**102CTBMCS2122016**  
**B. Tech - M. Tech CSE**  
**2026**

**Date: December 16, 2024**  
**Place: New Delhi**



## ORIGINALITY REPORT CERTIFICATE

I certify that:

1. The work contained in the dissertation is original and has been done by myself under the supervision of my supervisor.
2. The work has not been submitted to any other institute for any degree or diploma.
3. I have complied with the standards and guidelines given in the Ethics Code of Conduct of the Institute.
4. Whenever I have used materials (data, theoretical analysis, and text) from other sources, I have given due credit to them by citing them in the text of the dissertation and giving their details in the references.
5. Whenever I have quoted written materials from other sources, due credit is given to the sources by citing them.
6. From the plagiarism test, it is found that the similarity index of the entire dissertation is within 10%, and for a single article it is less than 10%, according to university guidelines.

**Aviral Kaintura**

**Enroll. No.: 102CTBMCS2122016**

**Date: December 16, 2024**

**Place: New Delhi**

**Forwarded by**

**Dr. Abinash Mishra**

**Date: December 16, 2024**



## CERTIFICATE

This is to certify the work contained in the dissertation entitled “**AI-Powered Real-Time DNS Threat Detection Using Large Language Models**”, submitted by **Aviral Kaintura**(enroll. No.: 102CTBMCS2122016) for the award of the degree of **B. Tech - M. Tech in Computer Science and Engineering** to the **National Forensic Sciences University, Delhi Campus**, is a record of bonafide research work carried out by him under my supervision and guidance.

**Dr. Abinash Mishra**

Assistant Professor

School of Cyber Security & Digital Forensics

National Forensic Sciences University

Delhi Campus, Delhi, India

Date: December 16, 2024

Place: New Delhi

विद्यया अमृतं अश्नुते

# ACKNOWLEDGMENT

I would like to express my deepest gratitude to all those who have supported me throughout the journey of this project. First and foremost, I am sincerely thankful to my supervisor, Dr. Abinash Mishra, whose invaluable guidance, insightful feedback, and unwavering encouragement have been instrumental in the successful completion of this project. His expertise and dedication have inspired me to delve deeper into the field of cybersecurity and strive for excellence.

I am also grateful to the faculty members of the School of Cyber Security & Digital Forensics at the National Forensic Sciences University, Delhi Campus, for providing a stimulating academic environment. Their comprehensive teaching and insightful discussions have significantly enhanced my understanding of complex concepts and have been a constant source of motivation.

A special note of thanks to my peers and colleagues who have been supportive companions throughout this journey. Our collaborative discussions, brainstorming sessions, and mutual encouragement have not only enriched my knowledge but also made this journey enjoyable and memorable.

Lastly, I am indebted to my family and friends for their unwavering support and belief in me. Their love, patience, and encouragement have been the foundation upon which I have built my academic pursuits. Without their constant support, this project would not have been possible.

# ABSTRACT

In today's interconnected digital landscape, the Domain Name System (DNS) serves as a critical component of internet infrastructure, translating human-readable domain names into IP addresses that machines use to communicate. However, DNS is increasingly being exploited by cyber threats, posing significant risks to network security. Traditional DNS security measures, which often rely on static blacklists and signature-based detection, are insufficient against sophisticated and evolving threats that can bypass conventional defenses.

In this project, I have developed an AI-powered real-time DNS threat detection system that leverages Large Language Models (LLMs) such as GPT-4Liu et al. 2024 and LLaMA 3.1Touvron et al. 2023. By integrating these advanced AI models, the system processes and analyzes DNS queries in real-time, identifying malicious patterns and automating threat detection and response mechanisms. The backend system intercepts DNS queries, utilizes LLMs for in-depth analysis, and determines the threat levels associated with each query. A user-friendly frontend dashboard provides real-time monitoring, allowing administrators to visualize network activity, manage threats, and make informed decisions.

The implementation of this system demonstrates significant improvements in detecting potential threats that traditional methods might miss. The AI integration provides nuanced understanding and contextual analysis of DNS queries, enhancing the overall security posture of the network. This project signifies a step forward in utilizing artificial intelligence for cybersecurity, showcasing the potential of LLMs in real-world applications, and lays the groundwork for future advancements in the field.

**Keywords:** DNS Security, Large Language Models, Real-Time Analysis, Threat Detection, Artificial Intelligence

# Contents

<b>ACKNOWLEDGMENT</b>	<b>i</b>
<b>ABSTRACT</b>	<b>ii</b>
<b>List of Abbreviations</b>	<b>vi</b>
<b>List Of Tables</b>	<b>viii</b>
<b>List of Figures</b>	<b>ix</b>
<b>List of Screenshots</b>	<b>x</b>
<b>Introduction</b>	<b>1</b>
0.1 Introduction and Problem Summary . . . . .	1
0.2 Aim and Objectives of the Project . . . . .	3
0.3 Scope of the Project . . . . .	3
<b>Literature Survey</b>	<b>4</b>
0.1 Current State of DNS Security . . . . .	4
0.2 DNS Attack Methods and Their Weaknesses . . . . .	5
0.2.1 DNS Spoofing: DNS Cache Poisoning . . . . .	5
0.2.2 DNS-Based Malware: DNS Tunneling . . . . .	6
0.2.3 DDoS Using DNS: DNS Amplification . . . . .	7
0.2.4 Domain Name System Security Extensions (DNSSEC) . . . . .	8
0.3 Problems and Weaknesses of Current Systems . . . . .	9
0.4 Conclusion of Literature Survey . . . . .	10
<b>Design: Analysis, Design Methodology, and Implementation Strategy</b>	<b>11</b>
0.1 Function of the System . . . . .	11
0.1.1 Use Case Description . . . . .	12
0.2 Data Modelling . . . . .	13
0.2.1 Entity-Relationship Modelling . . . . .	13
0.3 Functional and Behavioral Modelling . . . . .	15
0.3.1 Process Flow . . . . .	15
0.3.2 Data Flow Diagram . . . . .	16
0.3.3 Sequence Diagram . . . . .	18
0.4 Implementation Strategy . . . . .	18
0.4.1 Development Model . . . . .	19
0.4.2 Prototyping . . . . .	19

0.4.3	Version Control and Collaboration	21
0.5	System Components	21
0.5.1	DNS Monitoring Module	22
0.5.2	AI Analysis Module	22
0.5.3	Decision Engine	25
0.5.4	Cache Management	26
0.5.5	Frontend Dashboard	28
0.5.6	Database	28
0.5.7	Integration and Workflow	31
0.5.8	Security and Compliance	33
0.5.9	Scalability and Performance	33
0.5.10	Reliability and Redundancy	33
0.5.11	User Experience and Accessibility	34
0.6	Conclusion of Design	34
<b>Implementation</b>		<b>35</b>
0.1	Implementation Environment	35
0.1.1	Hardware Environment	35
0.1.2	Software Environment	35
0.2	Coding Standards	36
0.3	Key Implementation Details	36
0.3.1	DNS Query Interception	36
0.3.2	Integration with Large Language Models (LLMs)	37
0.3.3	Caching Mechanism	37
0.3.4	Frontend Dashboard Development	37
0.3.5	Security Features	38
0.3.6	Backend Server Development	38
0.4	Challenges Faced	39
0.5	Testing and Validation	40
0.6	Screenshots/Snapshots	40
0.6.1	Frontend Dashboard Display	41
0.6.2	Security Alerts	42
0.6.3	IP Management Interface	42
0.6.4	Domain Categories Chart	43
0.6.5	Risk Score Distribution Histogram	43
0.6.6	Recent Queries Table	44
0.7	Conclusion of Implementation	44
<b>Summary of Results and Future Scope</b>		<b>45</b>
0.1	Advantages and Unique Features	45
0.2	Results and Discussions	46
0.3	Future Scope of Work	46
<b>Conclusion</b>		<b>48</b>
<b>Bibliography</b>		<b>49</b>



<b>Appendices</b>	<b>50</b>
I. Code Samples . . . . .	50
A. DNS Attack Testing Script . . . . .	50
B. Frontend Dashboard . . . . .	54
<b>List of Publications/Online References</b>	<b>63</b>



# List of Abbreviations

Abbreviation	Description
<b>AI</b>	Artificial Intelligence: Simulation of human intelligence in machines to perform tasks like learning and problem-solving.
<b>API</b>	Application Programming Interface: A set of tools and protocols that allow software components to communicate.
<b>Asyncio</b>	Asynchronous I/O: A Python library for managing non-blocking, concurrent operations using <code>async/await</code> syntax.
<b>CI</b>	Continuous Integration: A practice where code updates are frequently integrated and tested in a shared repository.
<b>CI/CD</b>	Continuous Integration/Continuous Deployment: Methods for automating code integration, testing, and release processes.
<b>DGA</b>	Domain Generation Algorithm: A technique used to generate domain names programmatically, often for malicious purposes.
<b>DoH</b>	DNS over HTTPS: Encrypting DNS queries using HTTPS to enhance security and privacy.
<b>DoT</b>	DNS over TLS: Securing DNS queries through encryption with the TLS protocol.
<b>DNS</b>	Domain Name System: A service that maps domain names to numerical IP addresses for communication over the internet.
<b>Git</b>	Version Control System: A tool that tracks changes to source code, enabling collaboration and history management.
<b>GPT</b>	Generative Pre-trained TransformerVaswani 2017: A type of AI model used for tasks involving natural language understanding and generation.
<b>HTTP</b>	HyperText Transfer Protocol: A protocol for exchanging hypertext-based information on the web.
<b>HTTPS</b>	HyperText Transfer Protocol Secure: A secure version of HTTP that encrypts data to prevent eavesdropping.
<b>ICMP</b>	Internet Control Message Protocol: A protocol for sending error messages and network diagnostics.
<b>IP</b>	Internet Protocol: The system for addressing and routing data packets across networks.
<b>IPv4</b>	Internet Protocol Version 4: A version of IP that uses 32-bit addressing.
<b>IPv6</b>	Internet Protocol Version 6: The latest version of IP, featuring 128-bit addresses for improved scalability.
<b>JWT</b>	JSON Web Token: A compact format for securely exchanging information between parties.

<b>LLM</b>	Large Language Model: AI models trained on extensive datasets to process and generate human-like text.
<b>ML</b>	Machine Learning: A branch of AI where systems learn from data and improve their performance without explicit programming.
<b>NAT</b>	Network Address Translation: A technique for modifying network address information in data packets.
<b>PEP 8</b>	Python Enhancement Proposal 8: The official style guide for formatting Python code.
<b>Pytest</b>	Python Testing Framework: A tool for writing simple yet scalable tests for Python codebases.
<b>Redis</b>	Remote Dictionary Server: A fast, in-memory key-value database often used for caching and messaging.
<b>REST</b>	Representational State Transfer: A lightweight architecture for web APIs that uses HTTP methods for data communication.
<b>Scapy</b>	Python Packet Manipulation Library: A Python tool for creating, sending, and capturing network packets.
<b>SIEM</b>	Security Information and Event Management: Systems that collect and analyze security data from multiple sources.
<b>SSL</b>	Secure Sockets Layer: An outdated protocol for encrypting internet communications.
<b>TCP</b>	Transmission Control Protocol: A reliable, connection-oriented protocol for sending data over a network.
<b>TLS</b>	Transport Layer Security: A modern protocol for encrypting and securing data transmission.
<b>UI</b>	User Interface: The visual elements through which users interact with a system.
<b>UDP</b>	User Datagram Protocol: A connectionless protocol for transmitting data quickly without reliability guarantees.
<b>UX</b>	User Experience: The overall satisfaction and ease experienced by users while interacting with a system.
<b>WS</b>	WebSocket: A protocol that allows two-way, real-time communication between a client and a server.

---

# List of Tables

- Table 1: Pie Chart Categorizing Domains Based on Risk Levels
- Table 2: Histogram Showing Distribution of Risk Scores Assigned to DNS Queries



# List of Figures

1	DNS Resolution Process: A Hierarchical Approach . . . . .	1
2	DNS Spoofing via DNS Cache Poisoning . . . . .	5
3	DNS-Based Malware via DNS Tunneling . . . . .	6
4	DDoS Using DNS Amplification . . . . .	7
5	DNSSEC Chain of Trust: Validating DNS Records Using Public-Private Keys . . . . .	8
6	Use Case Diagram Illustrating Primary System Interactions . . . . .	12
7	Entity-Relationship Diagram for Database Schema . . . . .	14
8	Activity Diagram Illustrating DNS Query Processing . . . . .	15
9	Data Flow Diagram Showing Process Flow . . . . .	17
10	Sequence Diagram Illustrating Component Interactions . . . . .	18
11	Prototyping Diagram Depicting Development Stages and Iterative Refinement . . . . .	20
12	Class Diagram Illustrating System Components and Their Relationships . . . . .	22
13	AI Analysis Module Architecture . . . . .	24
14	Decision Engine Workflow . . . . .	26
15	Cache Management Process Flow . . . . .	27
16	Frontend Dashboard Interface . . . . .	28
17	Database Schema and Relationships . . . . .	30
18	Integration Workflow of System Components . . . . .	32
19	Frontend Dashboard Displaying DNS Queries and Threat Levels . . . . .	41
20	Security Alerts for Detected Threats . . . . .	42
21	IP Management Interface for Blocking and Unblocking IP Addresses . . . . .	42
22	Pie Chart Categorizing Domains Based on Risk Levels . . . . .	43
23	Histogram Showing Distribution of Risk Scores Assigned to DNS Queries . . . . .	43
24	Table Listing Recent DNS Queries Along with Their Risk Assessments . . . . .	44

# List of Screenshots

- Screenshot 1: Frontend Dashboard Displaying DNS Queries and Threat Levels
- Screenshot 2: IP Management Interface for Blocking and Unblocking IP Addresses
- Screenshot 3: Security Alerts for Detected Threats
- Screenshot 4: Listing Recent DNS Queries Along with Their Risk Assessments



# Introduction

## 0.1 Introduction and Problem Summary

In today's interconnected world, the rise of internet-enabled devices and the increasing reliance on digital services have made network security a critical concern. One of the most fundamental technologies enabling the functioning of the internet is the Domain Name System (DNS) Mockapetris and Dunlap 1988 Acting as a translator between human-readable domain names (e.g., `www.example.com`) and machine-understandable IP addresses, DNS serves as the internet's backbone for navigation and communication.

Despite its importance, DNS is frequently targeted by cyber attackers. My exploration of cybersecurity, both academically and practically, has highlighted significant vulnerabilities in traditional DNS security systems. Attackers are continuously innovating ways to exploit DNS through techniques like DNS tunneling, cache poisoning, domain generation algorithms (DGAs), and fast-flux hosting. These methods allow attackers to bypass traditional security defenses, making detection and mitigation complex.

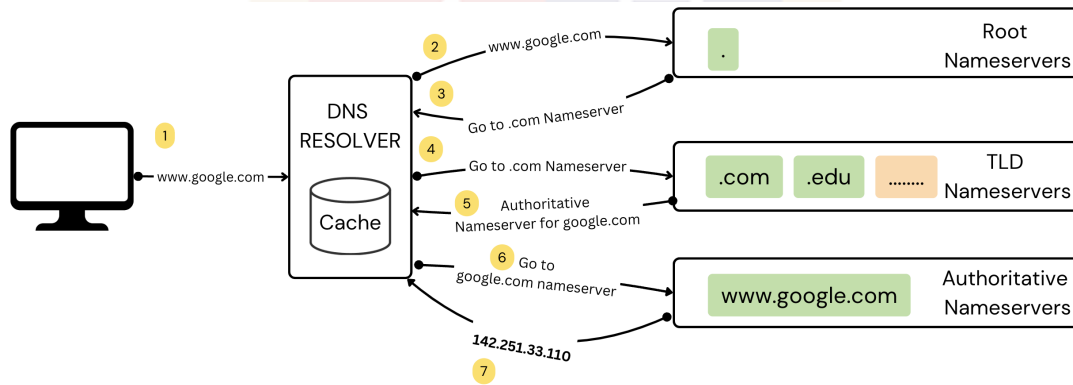


Figure 1: DNS Resolution Process: A Hierarchical Approach

The DNS resolution process, shown in Figure 1, involves multiple stages:

1. The DNS resolver (usually managed by the user's ISP) receives the query.
2. If a cached result exists, the resolver returns the IP address directly.
3. If no cache entry is found, the resolver contacts the **Root Nameservers**, which point it to the appropriate Top-Level Domain (TLD) nameservers.
4. The resolver queries the **TLD Nameservers** (e.g., `.com` domains) for further direction.

5. The **Authoritative Nameservers** provide the IP address for the requested domain.
6. The resolver returns the IP address to the user, enabling access to the target website.

While this hierarchical process is efficient, it is also a target for exploitation. For instance, attackers can manipulate DNS responses to redirect users to malicious websites, intercept queries, or use DNS channels to bypass network security systems. Techniques like DGAs allow malware to generate numerous domain names dynamically, making it difficult for blacklist-based systems to detect malicious activities. Likewise, DNS tunneling enables data exfiltration through DNS traffic, avoiding detection by firewalls.

To address these challenges, a more advanced and intelligent DNS security mechanism is required—one capable of adapting to the dynamic and evolving nature of cyber threats. Leveraging Large Language Models (LLMs) La O, Catania, and Parlanti 2024 such as GPT-4 and LLaMA 3.1 presents a significant opportunity to enhance detection capabilities. LLMs, with their ability to analyze patterns and contexts in human-like ways, can identify subtle anomalies in DNS queries that traditional methods may overlook.

This project focuses on designing and implementing an AI-powered DNS security system that uses real-time analysis and LLM-driven insights to detect and mitigate emerging threats. The ultimate goal is to contribute toward smarter, adaptive cybersecurity solutions that can evolve alongside the ever-changing threat landscape.



## 0.2 Aim and Objectives of the Project

**Aim:** Develop an AI-driven DNS security system that performs real-time network analysis and threat detection using Large Language Models (LLMs).

**Objectives:**

- **Real-Time Processing:** Build a system capable of analyzing DNS traffic in real-time for immediate identification of suspicious activities.
- **AI Model Integration:** Leverage pre-trained Large Language Models like GPT-4 to detect unusual patterns and behaviors in DNS queries.
- **Automated Threat Mitigation:** Implement mechanisms to automatically detect, respond to, and block malicious domains while alerting administrators.
- **User-Friendly Interface:** Design a comprehensive dashboard for real-time monitoring, threat visualization, and system management.
- **Performance Validation:** Measure system effectiveness against traditional security methods, ensuring it operates efficiently without significant network delays.

## 0.3 Scope of the Project

This project focuses on the design, development, and deployment of an AI-based DNS security solution. Key areas of work include:

- **Backend Development:** Creating a scalable backend system to intercept and process DNS queries while integrating AI models for analysis.
- **Frontend Development:** Designing an intuitive, web-based interface for administrators to monitor DNS traffic, receive alerts, and manage security configurations.
- **AI Model Integration:** Fine-tuning pre-trained LLMs to analyze DNS patterns and improve accuracy in identifying anomalies.
- **System Testing:** Validating the system through simulations, real-world datasets, and comparison with existing security tools.
- **Documentation and Deployment:** Providing comprehensive installation guidelines and operational documentation to support real-world deployment.

By addressing these areas, this project aims to bridge the gap between advanced AI techniques and practical cybersecurity needs. The developed system will serve as a scalable and adaptive solution to combat modern DNS-based threats effectively.

# Literature Survey

## 0.1 Current State of DNS Security

In today's interconnected world, the Domain Name System (DNS) is an essential component of internet infrastructure, responsible for translating human-readable domain names into IP addresses that computers use to communicate with each other. Securing DNS has been a critical focus within the cybersecurity community, leading to extensive research and development efforts over the years. Traditional approaches to DNS security have predominantly relied on fixed rules, blacklists, and signature-based detection methods. Tools such as Intrusion Detection Systems (IDS) and Intrusion Prevention Systems (IPS) have been used to monitor network traffic for suspicious activities.

One significant advancement in DNS security has been the introduction of the Domain Name System Security Extensions (DNSSEC) [Eastlake 3rd 1999](#), which authenticate DNS data to prevent certain types of attacks by ensuring data integrity and origin authentication. While DNSSEC adds a layer of security by preventing cache poisoning and other tampering attacks, its adoption has been slow, and it does not address all DNS security concerns.

Throughout my studies and research, I have observed that these conventional methods exhibit significant limitations, especially when confronted with the rapidly evolving landscape of cyber threats. Attackers continuously adapt their techniques, rendering static security measures less effective over time. Despite the implementation of DNSSEC and other security protocols, the dynamic nature of modern threats poses substantial challenges that necessitate more advanced solutions.

Furthermore, the rise of sophisticated attack methods, such as DNS tunneling and DNS-based malware, has highlighted the inadequacy of traditional defenses. These methods exploit the inherent trust in DNS communications and the fact that DNS traffic is often allowed to pass freely through network firewalls without thorough inspection.

## 0.2 DNS Attack Methods and Their Weaknesses

To gain a deeper understanding of the challenges in securing DNS, it is essential to examine specific attack methods that exploit vulnerabilities in the system. The following subsections detail prevalent DNS attack techniques, their methodologies, and the weaknesses they exploit.

### 0.2.1 DNS Spoofing: DNS Cache Poisoning

DNS Spoofing, specifically through **DNS Cache Poisoning**, involves attackers corrupting DNS resolver caches with malicious IP addresses. This attack redirects users to fraudulent websites without their knowledge, enabling attackers to carry out phishing attacks, distribute malware, or steal sensitive information.

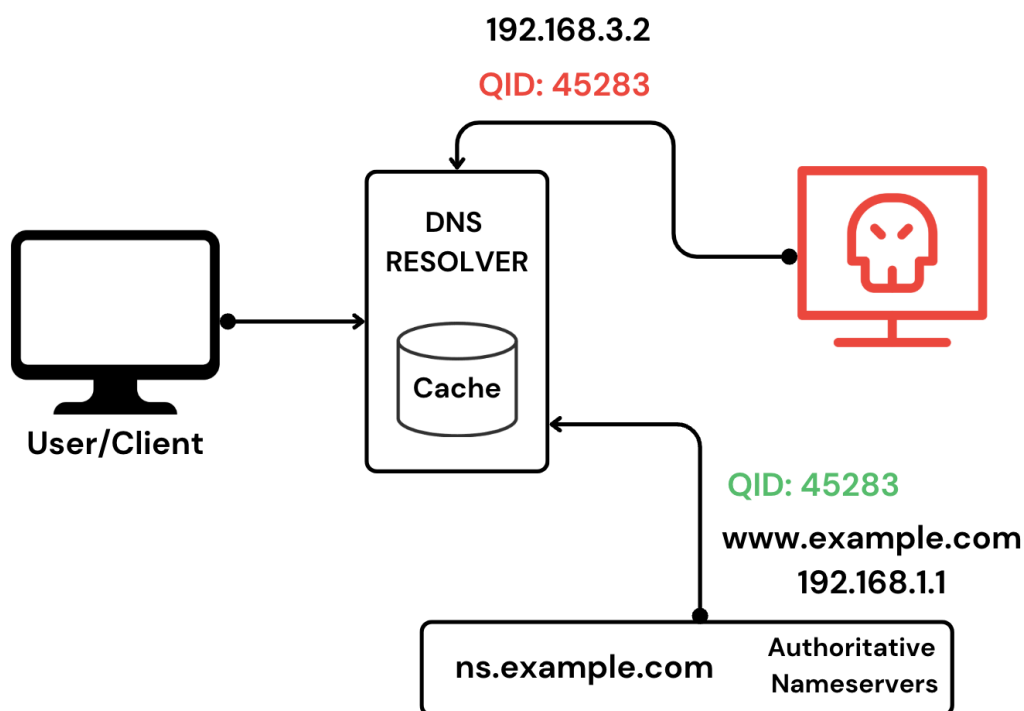


Figure 2: DNS Spoofing via DNS Cache Poisoning

#### Attack Steps:

1. **Injection of False DNS Records:** The attacker sends forged DNS responses to a resolver, attempting to insert malicious IP addresses into the resolver's cache.
2. **Redirection of Users:** When users query a legitimate domain (e.g., `www.example.com`), the resolver returns the incorrect IP address from its poisoned cache, leading users to the malicious site.
3. **Exploitation:** Users are unaware of the redirection and may provide sensitive information or download malware from the fraudulent site.

This type of attack highlights the vulnerabilities in DNS resolvers and underscores the critical need for validation mechanisms to prevent tampering. In my research, I found

that despite measures like source port randomization and transaction ID randomization, resolvers can still be susceptible to sophisticated cache poisoning attacks Kaminsky 2008.

### 0.2.2 DNS-Based Malware: DNS Tunneling

DNS Tunneling is an advanced technique where DNS queries are used as covert communication channels to bypass firewalls and exfiltrate data. Attackers exploit the fact that DNS traffic is typically allowed through firewalls and may not be closely monitored.

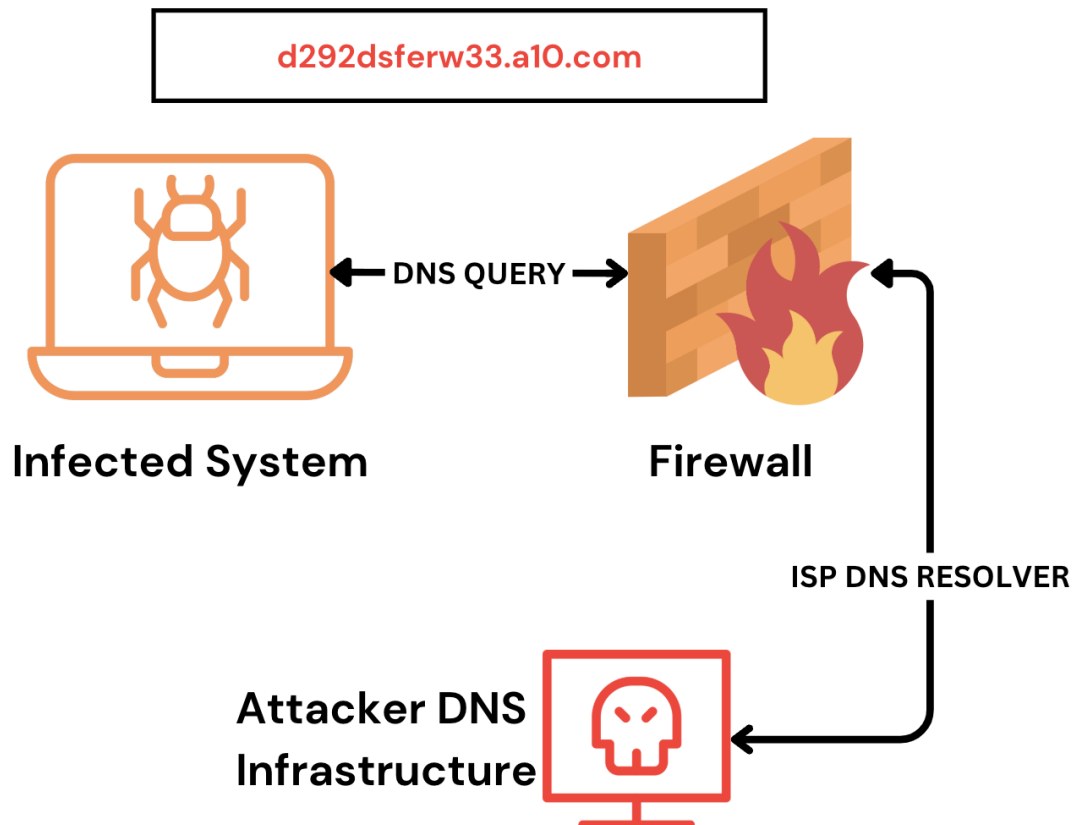


Figure 3: DNS-Based Malware via DNS Tunneling

#### How It Works:

1. **Data Encoding:** Malware-infected systems encode data or commands into DNS queries directed at attacker-controlled domains.
2. **Firewall Bypass:** Because DNS traffic is generally trusted, these queries bypass security mechanisms undetected.
3. **Data Exfiltration and Command Control:** The attacker's DNS server decodes the data from the queries and can send responses back, establishing a bidirectional communication channel.

This technique demonstrates how DNS, a legitimate protocol, can be exploited to evade detection and facilitate persistent threats. Cases like the *Dnscat2* tool have shown how attackers can use DNS tunneling for command and control communication Bilge et al. 2011.

### 0.2.3 DDoS Using DNS: DNS Amplification

DNS Amplification attacks are a form of Distributed Denial-of-Service (DDoS) that leverage the disparity in DNS request and response sizes to overwhelm a target. By exploiting open DNS resolvers, attackers can amplify the amount of traffic directed at a victim.

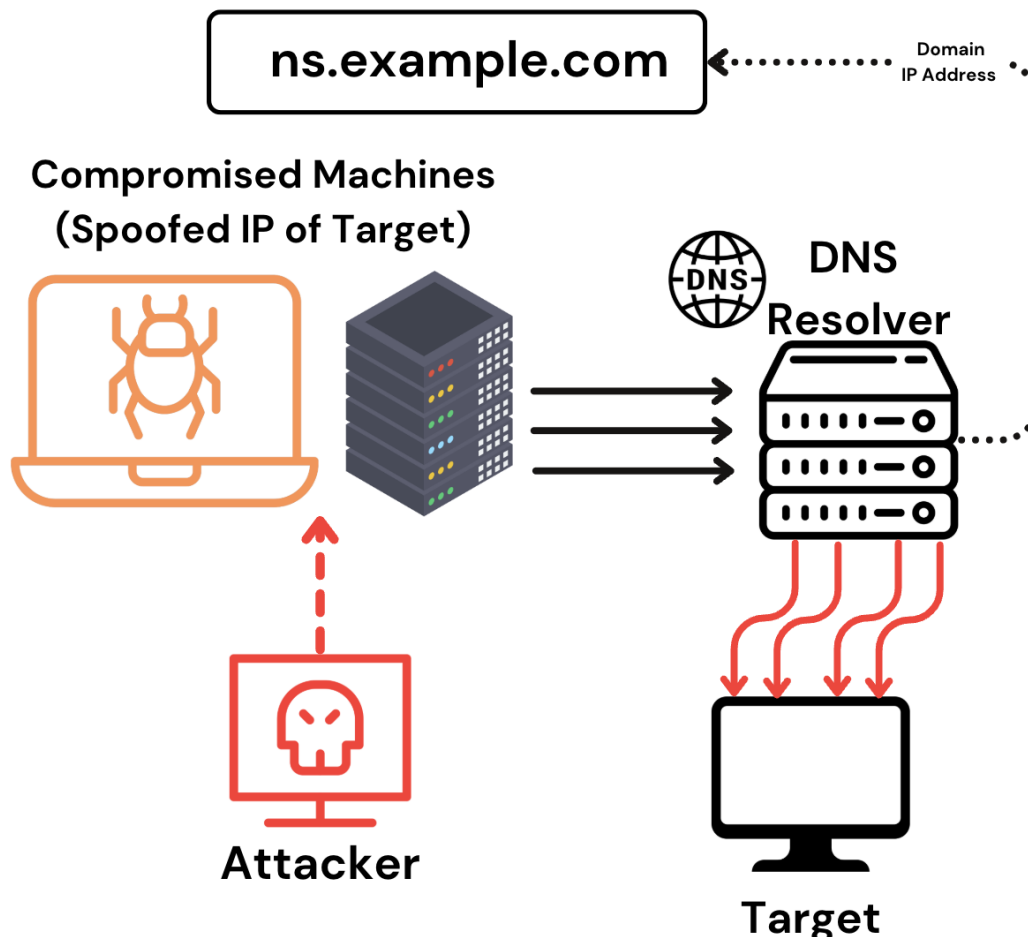


Figure 4: DDoS Using DNS Amplification

#### Attack Process:

1. **Spoofed Queries:** The attacker sends numerous small DNS queries with a spoofed source IP address (the target's IP) to open DNS resolvers.
2. **Amplification:** The resolvers, believing the queries come from the target, send large DNS responses to the spoofed IP address.
3. **Resource Exhaustion:** The overwhelming traffic consumes the target's bandwidth and resources, causing disruption and downtime.

DNS Amplification attacks highlight the misuse of publicly accessible DNS resolvers as amplification tools for large-scale attacks. Notable incidents include the 2013 Spamhaus DDoS attack, which peaked at 300 Gbps, one of the largest at the time **CloudflareSpamhaus**.

## 0.2.4 Domain Name System Security Extensions (DNSSEC)

DNSSEC is a critical protocol introduced to enhance DNS security by addressing vulnerabilities like DNS spoofing and cache poisoning. DNSSEC adds cryptographic signatures to DNS records to ensure data integrity and validate the authenticity of responses.

*"DNSSEC is like adding a digital seal of authenticity to DNS data, ensuring that the chain of trust remains intact throughout the DNS resolution process."*

**How DNSSEC Works:** DNSSEC establishes a hierarchical **Chain of Trust**, where each level of the DNS hierarchy (Root, TLD, and Authoritative Nameserver) validates the next using public-private key cryptography.

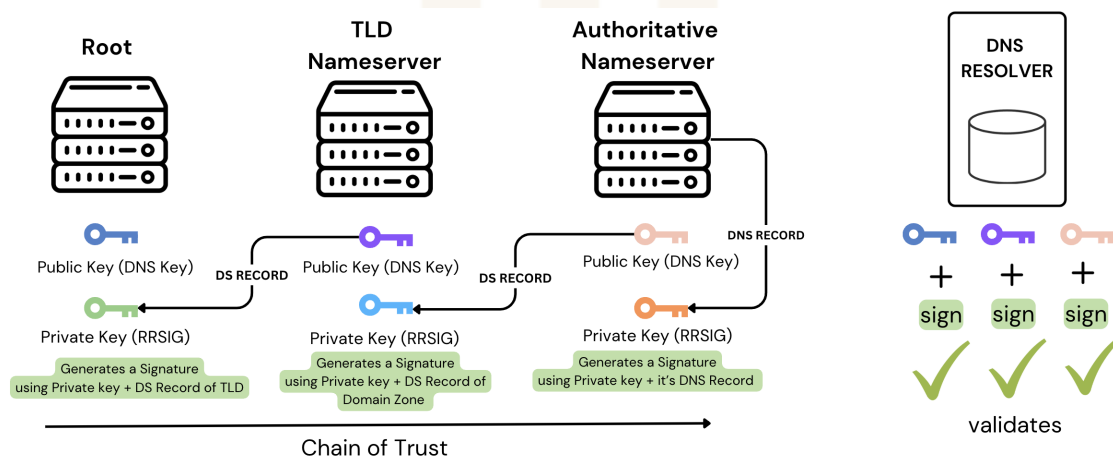


Figure 5: DNSSEC Chain of Trust: Validating DNS Records Using Public-Private Keys

### Steps in DNSSEC Validation:

1. **Signature Generation:** Each zone in the DNS hierarchy signs its DNS records with a private key, generating a *Resource Record Signature (RRSIG)*.
2. **Public Key Publication:** The public key is published in a *DNSKEY* record, which can be used by resolvers to verify the signature.
3. **Chain of Trust Establishment:** A chain of trust is established from the root zone downward, where each level validates the keys of the level below.
4. **Validation by Resolvers:** DNS resolvers validate each signature at every level of the chain using the corresponding public keys, ensuring the authenticity and integrity of the data.

### Benefits of DNSSEC:

- **Data Integrity and Authenticity:** Ensures that DNS responses have not been tampered with and are authentic.
- **Chain of Trust:** Establishes a trusted hierarchy from the root servers to the authoritative servers.
- **Protection Against Spoofing:** Prevents attackers from successfully conducting cache poisoning and man-in-the-middle attacks.



### Challenges of DNSSEC:

- **Increased Response Size:** Cryptographic signatures increase the size of DNS responses, potentially leading to fragmentation and affecting performance.
- **Key Management Complexity:** Requires careful management of cryptographic keys, including regular key rotation to maintain security.
- **Partial Adoption:** Limited deployment across all DNS resolvers and authoritative servers reduces its overall effectiveness.
- **Not a Comprehensive Solution:** Does not protect against all types of DNS attacks, such as DNS tunneling or DDoS attacks.

Despite its challenges, DNSSEC is an essential measure for improving DNS security and protecting against specific types of attacks. However, it's clear that DNSSEC alone is insufficient to address the full spectrum of DNS-related threats.

## 0.3 Problems and Weaknesses of Current Systems

From my analysis of DNS attack methods and security protocols, I have identified several critical weaknesses in current DNS security systems:

- **Lack of Real-Time Detection:** Delays in analyzing DNS traffic can allow attacks to succeed before mitigation measures are triggered. Traditional systems often rely on reactive rather than proactive measures.
- **Static Detection Mechanisms:** Blacklists and signature-based methods fail to detect zero-day threats or evolving attack patterns. Attackers can easily circumvent static defenses by changing their tactics.
- **Limited Contextual Understanding:** Traditional security solutions lack the ability to understand the context or semantics of DNS queries, missing subtle indicators of malicious intent.
- **High Rates of False Positives/Negatives:** Over-reliance on static rules can result in legitimate queries being blocked (false positives) or malicious queries being allowed (false negatives), undermining network efficiency and security.
- **Scalability Challenges:** High volumes of DNS queries in modern networks strain existing systems, leading to performance issues and potential gaps in security coverage.
- **Encrypted Traffic Limitations:** The adoption of DNS over HTTPS (DoH) and DNS over TLS (DoT) complicates traffic analysis for traditional security solutions, as the contents of DNS queries are encrypted.

These challenges emphasize the need for an intelligent, adaptive DNS security system capable of real-time detection, contextual analysis, and automated mitigation. In my exploration of current literature, I found that integrating artificial intelligence and machine learning into security solutions offers promising avenues for addressing these weaknesses Sommer and Paxson 2010.

## 0.4 Conclusion of Literature Survey

The literature survey highlights the limitations of current DNS security systems, the prevalence of DNS-based attack methods, and the role of protocols like DNSSEC in protecting DNS infrastructure. While DNSSEC addresses specific vulnerabilities by adding cryptographic validation, its complexity and limited adoption leave significant gaps in security.

To overcome these limitations, a new approach is required—one that leverages AI and machine learning for real-time DNS query analysis, threat detection, and automated responses. By incorporating advanced techniques capable of understanding context and adapting to evolving threats, the proposed system aims to enhance DNS security effectively.

Through this project, I intend to bridge the gap between existing security measures and the emerging needs of modern networks. By integrating Large Language Models into DNS analysis, we can harness their capabilities to detect anomalies, understand patterns, and respond to threats promptly. This innovative approach holds the potential to significantly improve the security posture of networks and protect users from sophisticated DNS-based attacks.





# Design: Analysis, Design Methodology, and Implementation Strategy

## 0.1 Function of the System

Designing an effective DNS security system is a multifaceted endeavor that demands a thoughtful and comprehensive approach. The primary objective is to create a system that not only intercepts and analyzes DNS queries but also makes informed, real-time decisions to safeguard the network from potential threats. At its core, the system functions as an intelligent intermediary between clients and DNS servers, bridging the gap between user requests and domain resolution.

When a client initiates a DNS query, the system seamlessly intercepts this request, positioning itself strategically between the client and the DNS resolver. This interception is crucial as it allows the system to scrutinize each query meticulously before it reaches the DNS server. Utilizing advanced Artificial Intelligence (AI) models, specifically Large Language Models (LLMs) like GPT-4 and LLaMA 3.1, the system processes the intercepted queries to assess their legitimacy and potential risk.

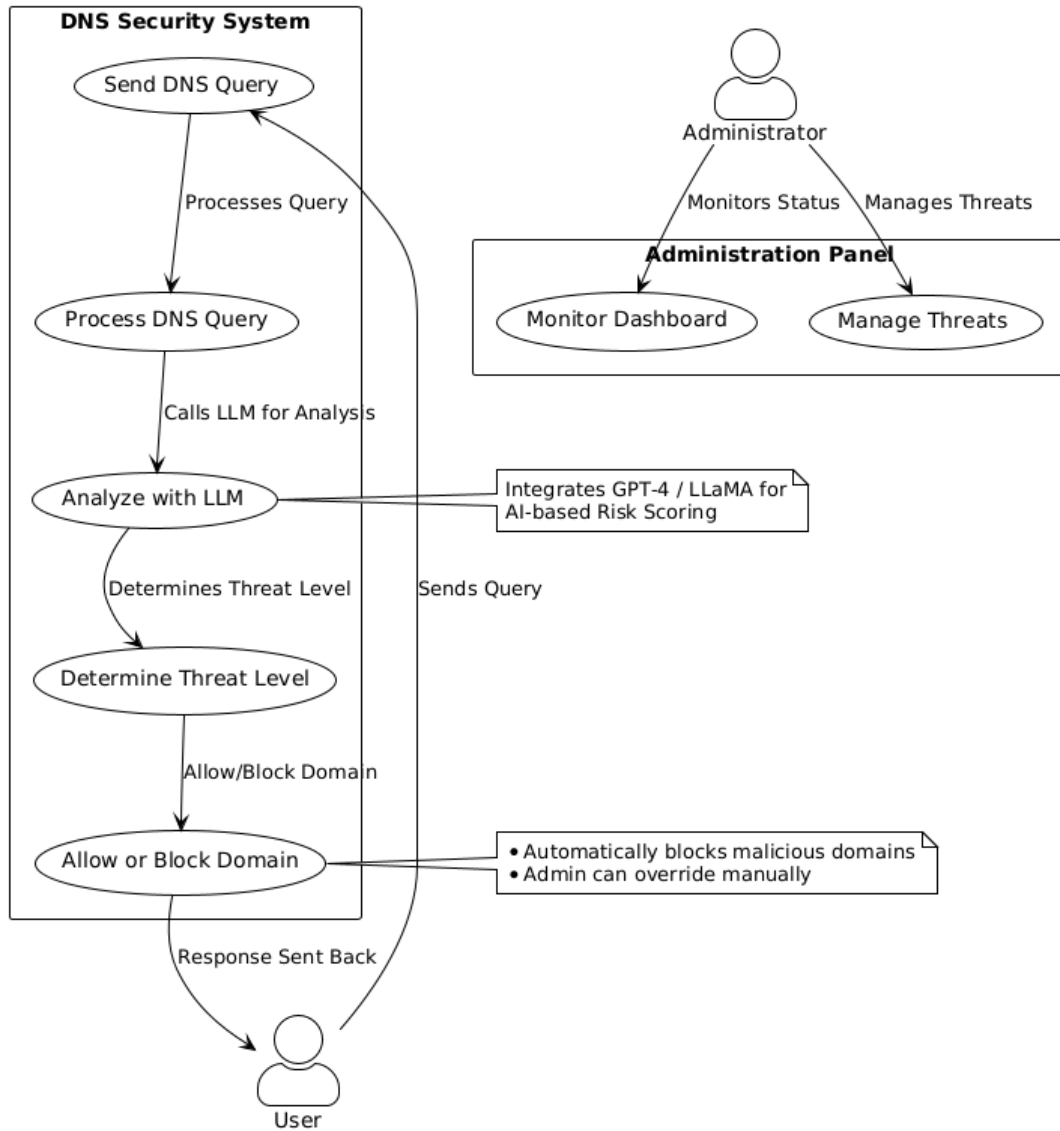


Figure 6: Use Case Diagram Illustrating Primary System Interactions

### 0.1.1 Use Case Description

In developing the system, I identified several primary use cases that encapsulate the essential interactions between users, the system, and external entities. Understanding these use cases was pivotal in shaping the system's functionality and ensuring it meets the intended security objectives.

- **Process DNS Query:** The system receives DNS queries from users and initiates the processing pipeline.
- **Analyze Query with LLM:** Leveraging Large Language Models (LLMs) like GPT-4 and LLaMA 3.1, the system analyzes each query to assess its risk level.
- **Determine Threat Level:** Based on the analysis, the system categorizes the query as either safe or potentially malicious.

- **Allow or Block Domain:** The system decides to permit the DNS resolution or block the domain to prevent any malicious activity.
- **Monitor Dashboard:** Administrators can view real-time data, threat alerts, and system status through a user-friendly dashboard.
- **Manage Threats:** Administrators have the capability to manually manage threats, such as blocking or unblocking specific IPs and adjusting system settings.

These use cases are further detailed in the accompanying Use Case Diagram (Figure 6), which visually represents the interactions and workflows within the system.

## 0.2 Data Modelling

Effective data management is crucial for the system's performance and reliability. To achieve this, I employed Entity-Relationship (ER) modelling to design a robust database schema that caters to the system's data storage and retrieval needs.

### 0.2.1 Entity-Relationship Modelling

Understanding the relationships between different data entities ensures that the system can efficiently store, access, and manage information. Below are the key entities identified and their interactions:

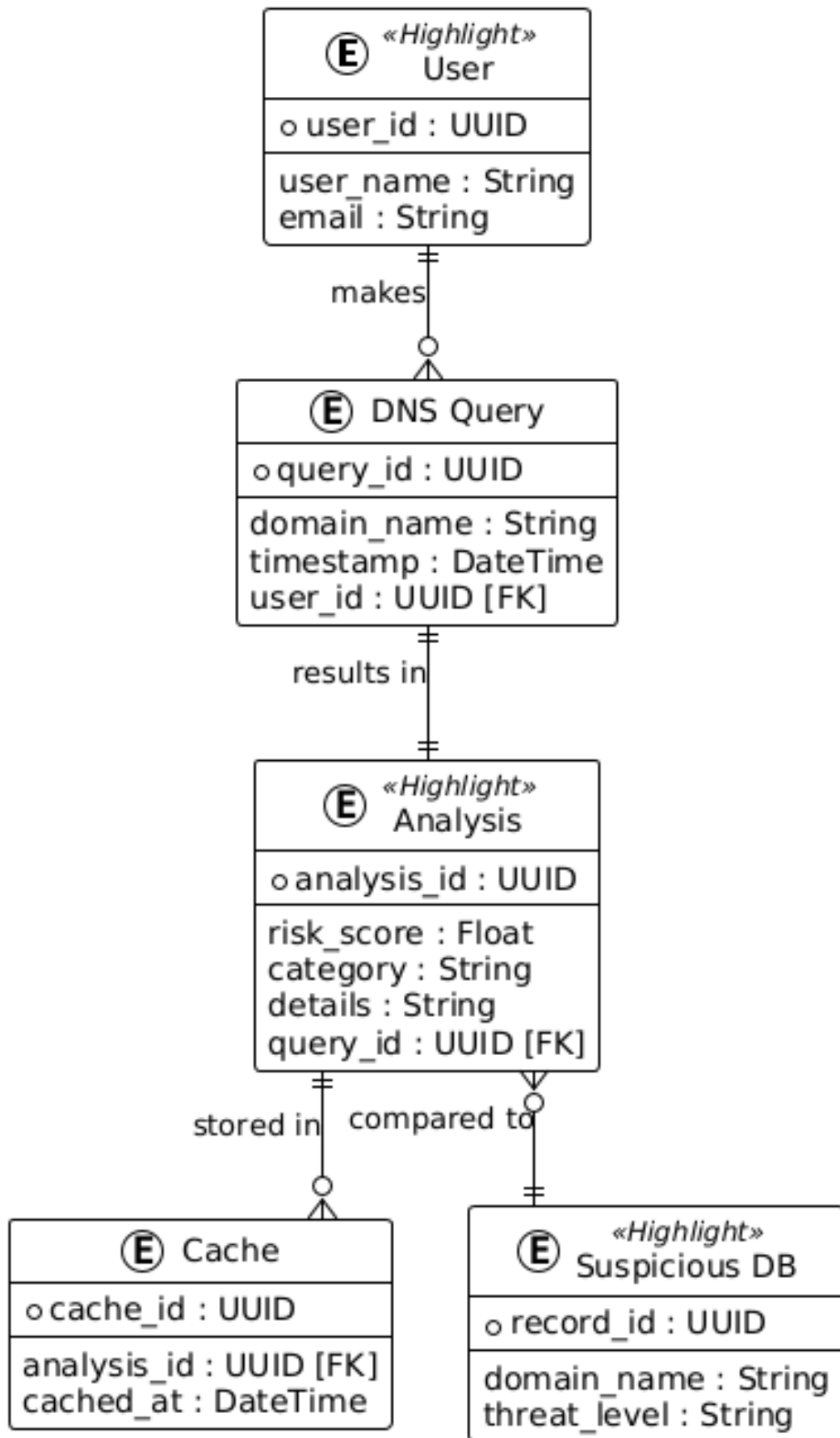


Figure 7: Entity-Relationship Diagram for Database Schema

### Key Entities:

- **User:** Represents system users who send DNS queries.
- **DNS Query:** Captures the domain name, timestamp, and user reference.
- **Analysis:** Stores the results of threat analysis, including risk scores and categories.
- **Cache:** Contains analysis results for previously queried domains to improve performance.
- **Suspicious DB:** Maintains a list of known malicious domains and their threat levels.

The ER Diagram (Figure 7) illustrates the relationships between these entities, ensuring data integrity and facilitating efficient data operations.

## 0.3 Functional and Behavioral Modelling

To capture the dynamic interactions and workflows within the system, I utilized various modelling techniques, including Data Flow Diagrams (DFD), Activity Diagrams, and Sequence Diagrams. These models provide a clear visualization of the system's processes and component interactions.

### 0.3.1 Process Flow

The system's process flow is designed to handle DNS queries seamlessly from interception to resolution or blocking. The following steps outline the workflow:

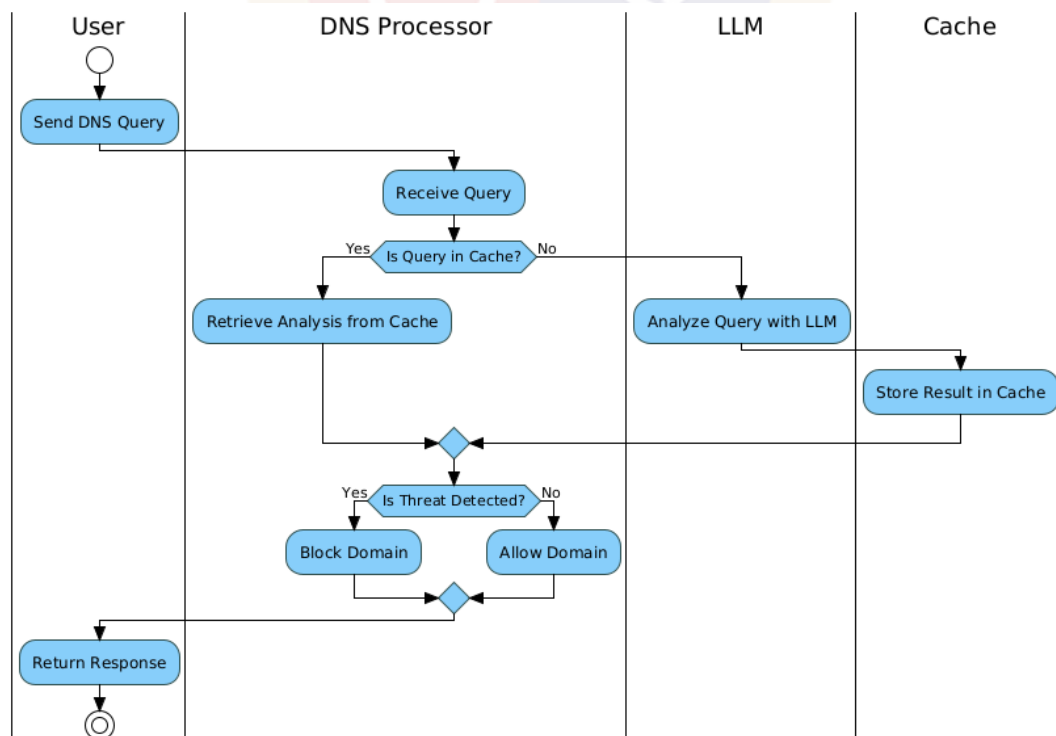


Figure 8: Activity Diagram Illustrating DNS Query Processing

1. **Query Interception:** The system captures DNS queries initiated by clients.
2. **Cache Check:** It verifies if the domain has been recently analyzed and if the result is cached to optimize performance.
3. **Analysis:** If the query is not cached, it is sent to the LLM for detailed analysis.
4. **Result Evaluation:** The system evaluates the analysis results to determine the threat level.
5. **Decision Making:** Based on the threat assessment, the system decides to allow or block the domain.
6. **Response to Client:** The decision is communicated back to the client, either forwarding the query to the DNS server or terminating it.
7. **Logging and Reporting:** All actions are logged for auditing, and relevant data is updated on the dashboard for real-time monitoring.

The Activity Diagram (Figure 8) visually represents these steps, highlighting the flow of processes and decision points within the system.

### 0.3.2 Data Flow Diagram

To provide a comprehensive view of how data moves through the system, I developed a Data Flow Diagram (DFD) that outlines the interactions between different modules and data stores.

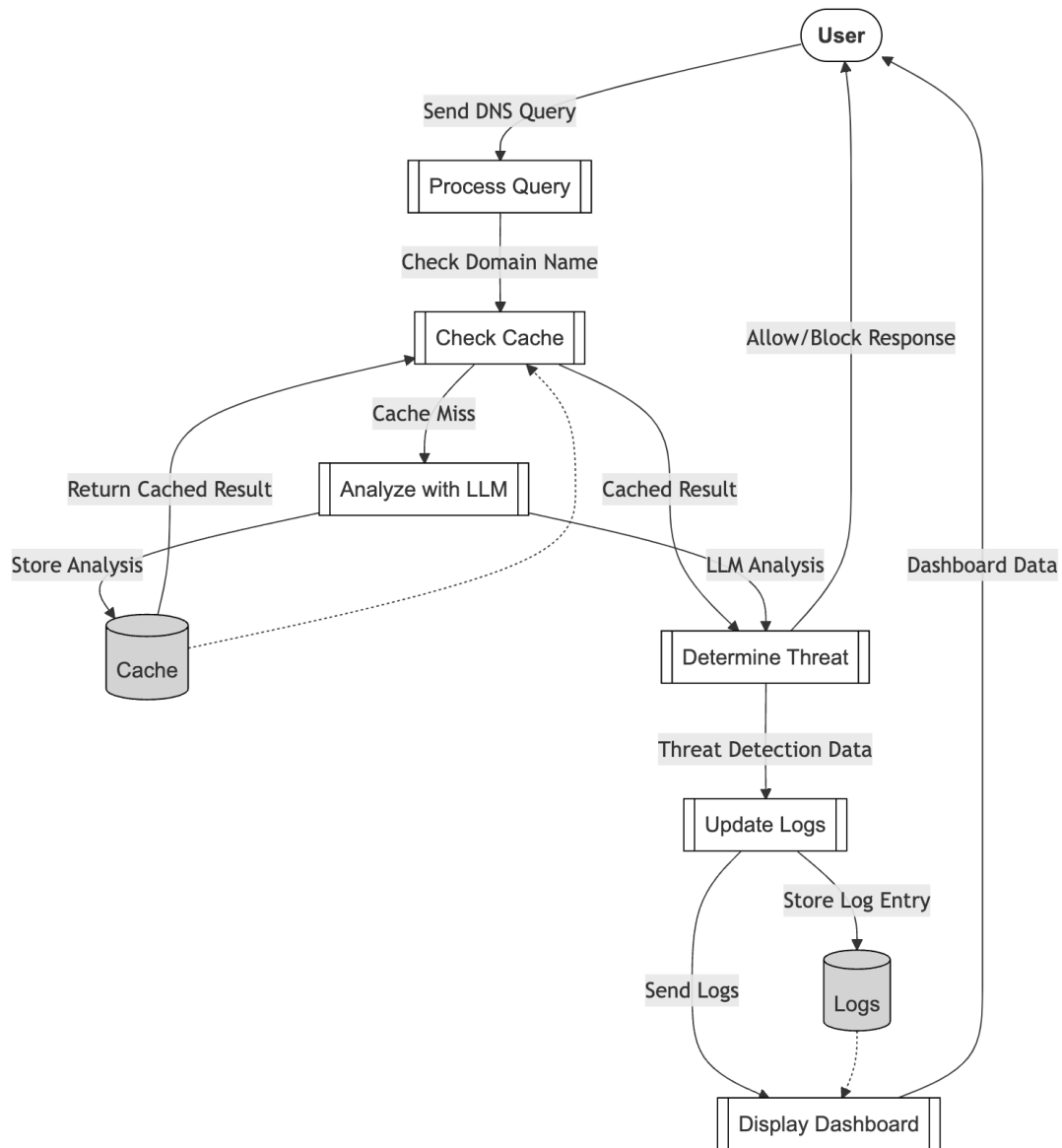


Figure 9: Data Flow Diagram Showing Process Flow

**Key Components:**

- **User/Client:** Initiates DNS queries.
- **DNS Monitoring Module:** Intercepts and processes DNS queries.
- **AI Analysis Module:** Interfaces with LLMs to analyze queries.
- **Decision Engine:** Determines the appropriate action based on analysis.
- **Cache Management:** Stores recent analysis results for quick retrieval.
- **Frontend Dashboard:** Displays real-time data and alerts to administrators.
- **Database:** Stores logs, settings, and historical data.

The DFD (Figure 9) elucidates the flow of data between these components, ensuring clarity in how information is processed and managed within the system.

### 0.3.3 Sequence Diagram

To detail the interactions between various system components during the DNS query processing, I created a Sequence Diagram that maps out the sequence of messages exchanged.

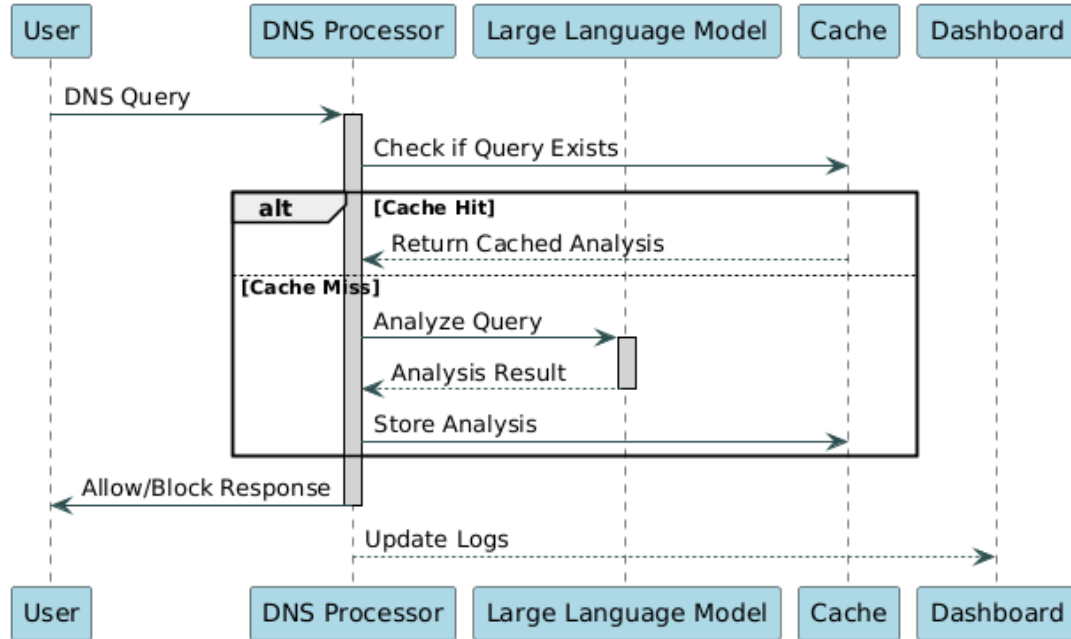


Figure 10: Sequence Diagram Illustrating Component Interactions

#### Interaction Steps:

1. The client sends a DNS query to the DNS Monitoring Module.
2. The DNS Monitoring Module checks the Cache Management for recent analysis results.
3. If the query is not cached, it forwards the query to the AI Analysis Module.
4. The AI Analysis Module processes the query using the integrated LLM and returns the analysis result.
5. The Decision Engine evaluates the result and decides to allow or block the domain.
6. The DNS Monitoring Module responds to the client based on the decision.
7. All actions are logged into the Database, and relevant data is updated on the Frontend Dashboard.

This Sequence Diagram (Figure 10) provides a clear depiction of the temporal order of interactions, ensuring that each component functions cohesively within the system.

## 0.4 Implementation Strategy

Developing a sophisticated DNS security system required a strategic implementation approach to ensure scalability, reliability, and effectiveness. I adopted a combination of iterative development and modular design principles to facilitate this process.



### 0.4.1 Development Model

I embraced an **Iterative and Incremental Development** model, which allowed me to build and refine the system in manageable segments. This approach provided the flexibility to incorporate feedback, address challenges promptly, and ensure that each component met its functional requirements before proceeding to the next phase.

### 0.4.2 Prototyping

Prototyping was an indispensable component of the development strategy, enabling a hands-on approach to validate critical components and processes early in the project lifecycle. By constructing proof-of-concept models, I was able to test the feasibility of integrating Large Language Models (LLMs) with the DNS monitoring infrastructure. These prototypes served multiple purposes: they allowed for the identification of potential technical challenges, provided opportunities to refine system functionalities, and facilitated the optimization of performance metrics.

For instance, the development of the **LLM Integration Prototype** was crucial in assessing the effectiveness of AI-driven analysis in real-time threat detection. This prototype ensured that the system could accurately interpret and evaluate DNS queries, leveraging the advanced capabilities of models like GPT-4 and LLaMA 3.1. Additionally, the **dashboard prototype** provided a preliminary interface for administrators, offering insights into data visualization and user experience design. This early version of the dashboard was instrumental in gathering user feedback and making iterative improvements to enhance usability and functionality.

Another significant prototype was the **Packet Capture Prototype**, which was essential in testing the system's ability to intercept and preprocess DNS queries efficiently. This prototype validated the real-time interception mechanism, ensuring minimal latency and seamless integration with existing network infrastructure. Throughout the prototyping phase, iterative testing and feedback loops were crucial in uncovering and addressing issues, ultimately leading to a more resilient and efficient DNS security system.

Furthermore, the **Prototyping Diagram** (Figure 11) illustrates the various stages and components involved in the prototyping process. Each prototype acted as a building block, contributing to the overall robustness and reliability of the final implementation. This proactive approach not only mitigated risks early on but also enhanced the system's scalability and adaptability to evolving threats.

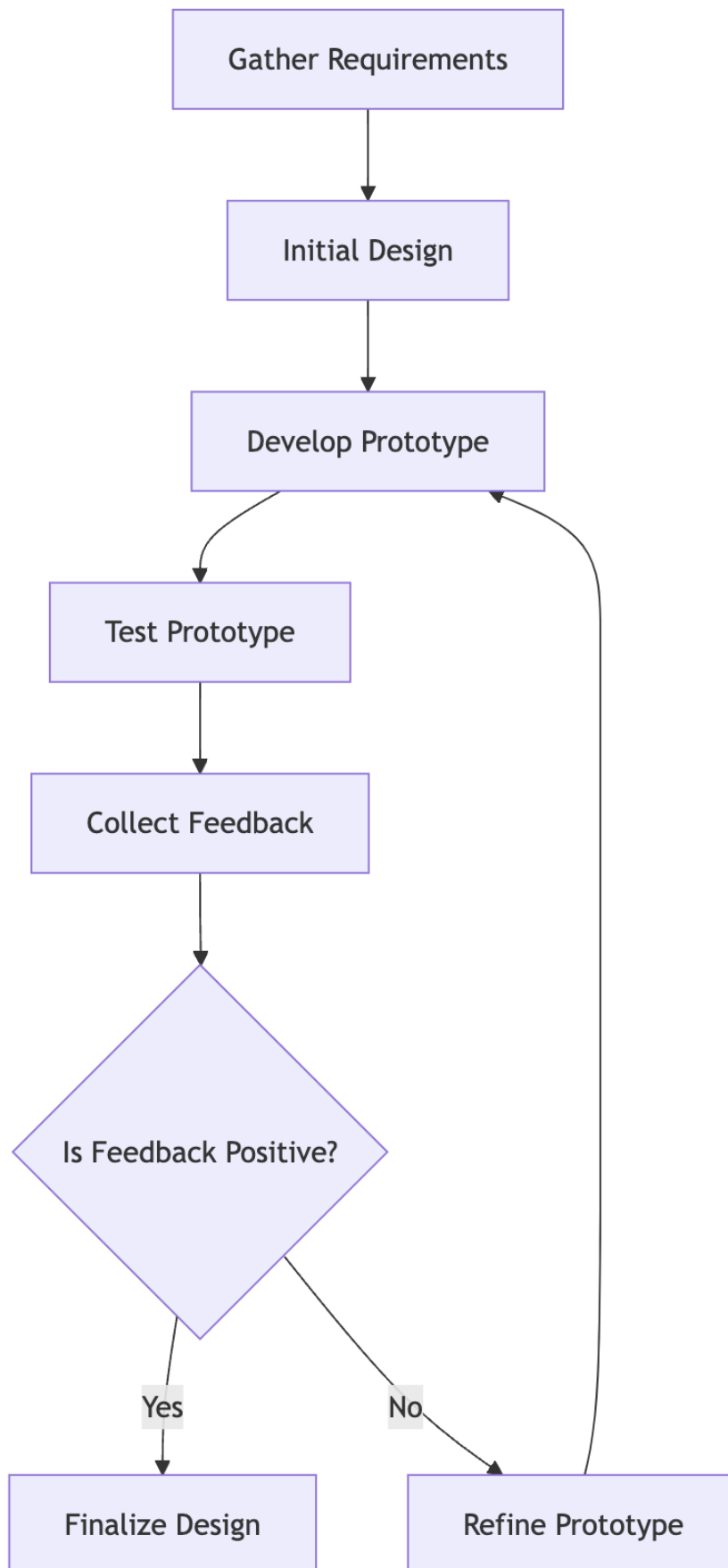


Figure 11: Prototyping Diagram Depicting Development Stages and Iterative Refinement

### Developed Prototypes:

- **LLM Integration Prototype:** This prototype tested the interaction between the system and LLM APIs, ensuring efficient data exchange and accurate analysis results.
- **Dashboard Prototype:** A basic version of the frontend dashboard was created to visualize data and gather user feedback on the interface design and functionality.
- **Packet Capture Prototype:** Implemented a test version of the packet capture mechanism to validate real-time DNS query interception and preprocessing.

These prototypes, as depicted in Figure 11, provided invaluable insights that informed the final system design and implementation.

### 0.4.3 Version Control and Collaboration

Throughout the project, I utilized **Git** for version control, maintaining a repository on **GitHub**. This facilitated efficient tracking of changes, management of different development branches, and seamless collaboration when necessary. Regular commits and well-documented commit messages ensured that the project's progress was transparent and easily manageable.

## 0.5 System Components

The architecture of the DNS security system is composed of several interrelated components, each responsible for specific functionalities that collectively ensure robust security and efficient operation. These components work in harmony to intercept, analyze, and manage DNS queries, providing a comprehensive security solution against a wide range of DNS-based threats. By integrating advanced AI analysis, real-time decision-making, and caching mechanisms, the system efficiently identifies malicious domains while minimizing latency. Furthermore, the intuitive administration panel enables system administrators to monitor threats, manage logs, and configure security settings seamlessly. This cohesive design ensures both scalability and adaptability, making the system capable of handling increasing network traffic and evolving security challenges.

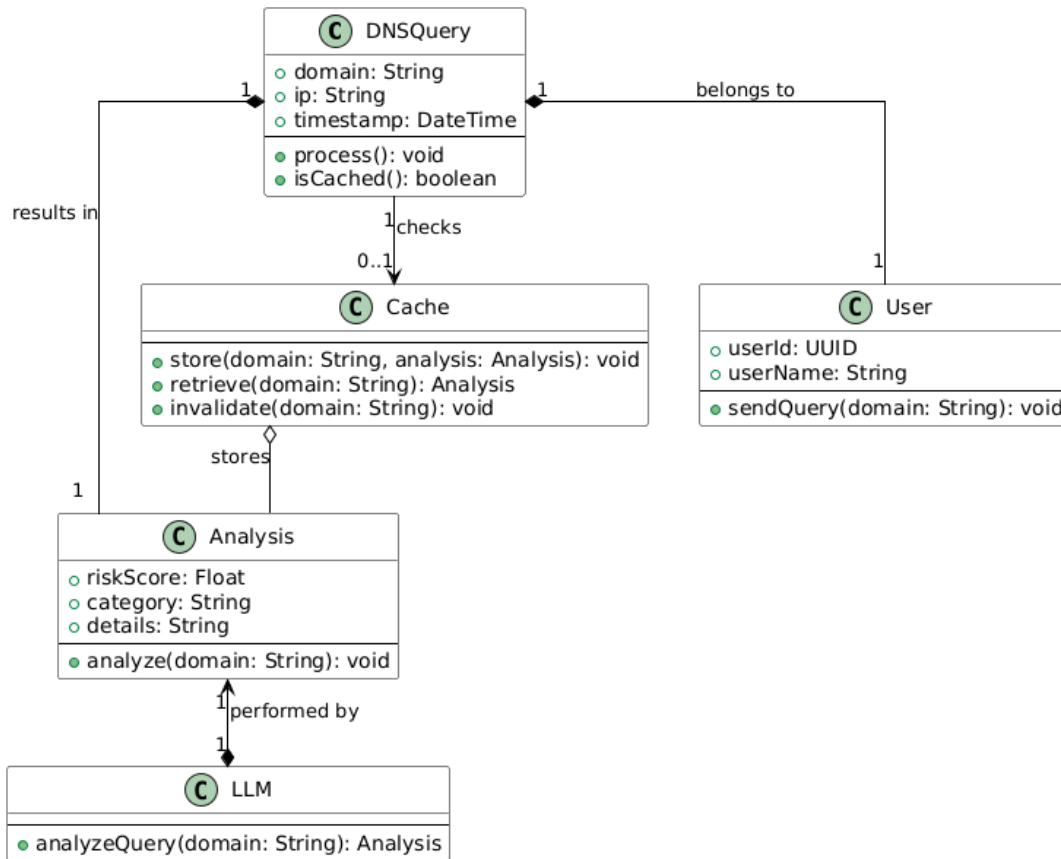


Figure 12: Class Diagram Illustrating System Components and Their Relationships

### 0.5.1 DNS Monitoring Module

The **DNS Monitoring Module** serves as the cornerstone of the system, acting as the first line of defense by intercepting all incoming DNS queries from clients. This module operates as an intelligent proxy between the client and the DNS resolver, ensuring that every query is captured and subjected to thorough analysis before reaching its intended destination. Key responsibilities of this module include:

- **Query Interception:** Captures DNS queries in real-time without disrupting the normal DNS resolution process.
- **Preprocessing:** Cleans and formats incoming queries to prepare them for analysis, such as parsing domain names and extracting relevant metadata.
- **Traffic Management:** Balances the load to handle high volumes of DNS traffic efficiently, ensuring minimal latency and optimal performance.

### 0.5.2 AI Analysis Module

At the heart of the system lies the **AI Analysis Module**, which leverages advanced Large Language Models (LLMs) like GPT-4 and LLaMA 3.1 to evaluate the legitimacy of each DNS query. This module is designed to detect sophisticated and evolving threats that traditional rule-based systems might overlook. Its primary functions include:

- **Threat Detection:** Utilizes natural language processing and pattern recognition to identify anomalies and malicious intent within DNS queries.
- **Risk Scoring:** Assigns a risk score to each query based on the analysis, categorizing them into safe or potentially harmful.
- **Contextual Analysis:** Understands the context and semantics of domain names to detect subtle indicators of compromise that may signal zero-day threats.



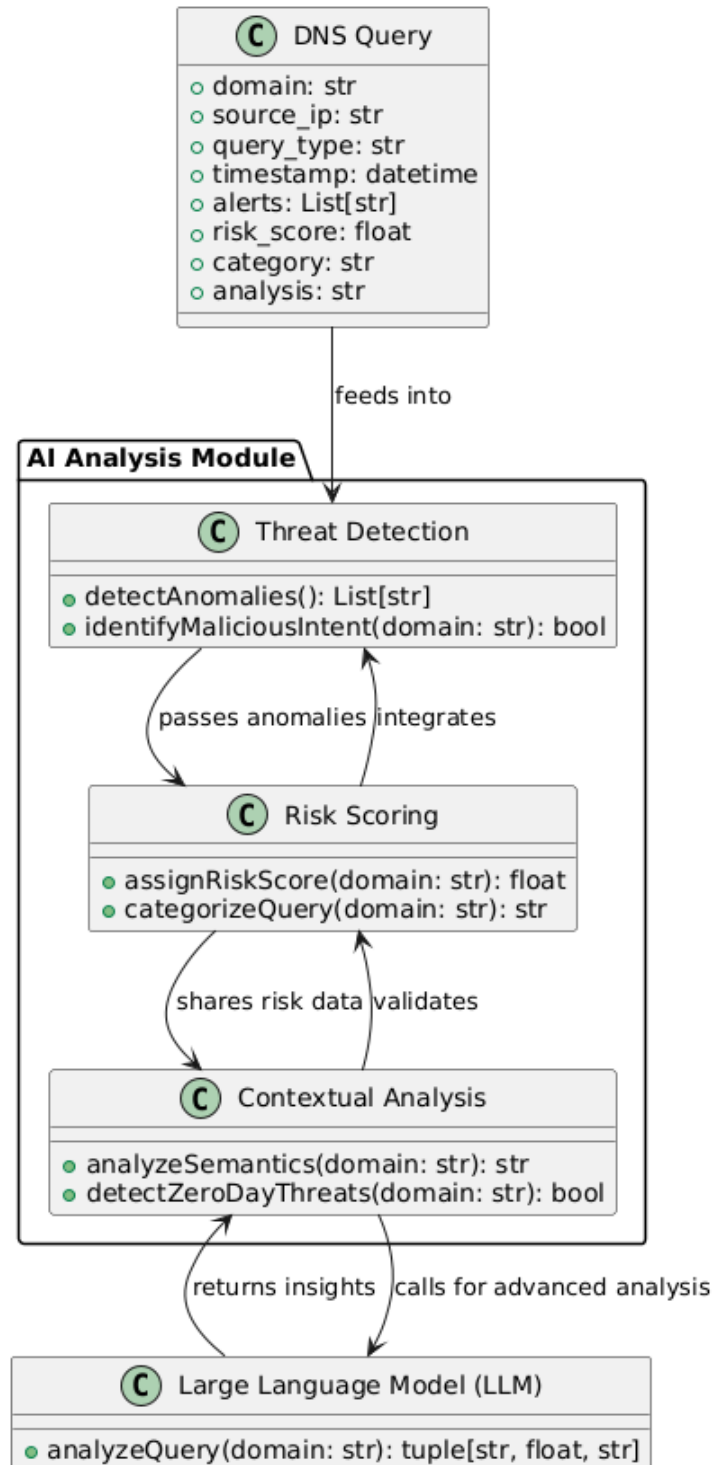


Figure 13: AI Analysis Module Architecture

### 0.5.3 Decision Engine

The **Decision Engine** is responsible for interpreting the outcomes generated by the AI Analysis Module and making informed decisions about each DNS query. This component ensures that the system responds appropriately to potential threats while maintaining legitimate DNS traffic flow. Its core functionalities include:

- **Evaluation of Risk Scores:** Analyzes the risk scores provided by the AI module to determine the severity of potential threats.
- **Policy Enforcement:** Implements predefined security policies to decide whether to allow or block specific DNS queries based on their assessed risk.
- **Automated Responses:** Executes immediate actions, such as blocking malicious domainsZhauniarovich et al. 2018 or flagging suspicious activity for further investigation.

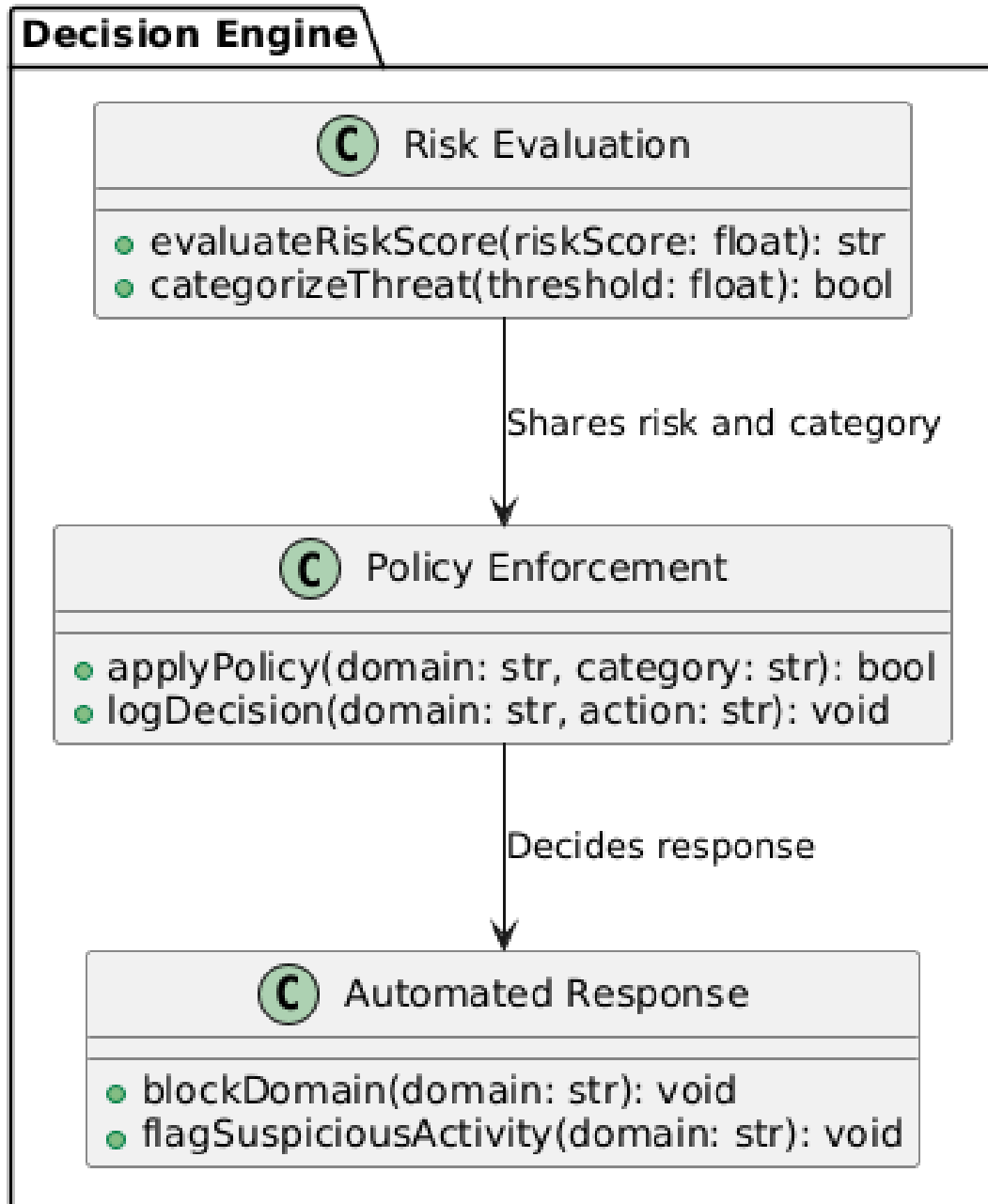


Figure 14: Decision Engine Workflow

### 0.5.4 Cache Management

To enhance the system's efficiency and reduce redundant processing, the **Cache Management** component stores recent DNS query analyses. By maintaining a cache of frequently accessed domains and their respective risk assessments, this module significantly optimizes performance. Key features include:

- **Result Caching:** Retains analysis results for a configurable duration to expedite responses for repeat queries.
- **Cache Invalidation:** Ensures that outdated or compromised cache entries are promptly removed or updated to maintain data integrity.



- **Performance Optimization:** Minimizes the load on the AI Analysis Module by reducing the need for repetitive analyses, thereby conserving computational resources.

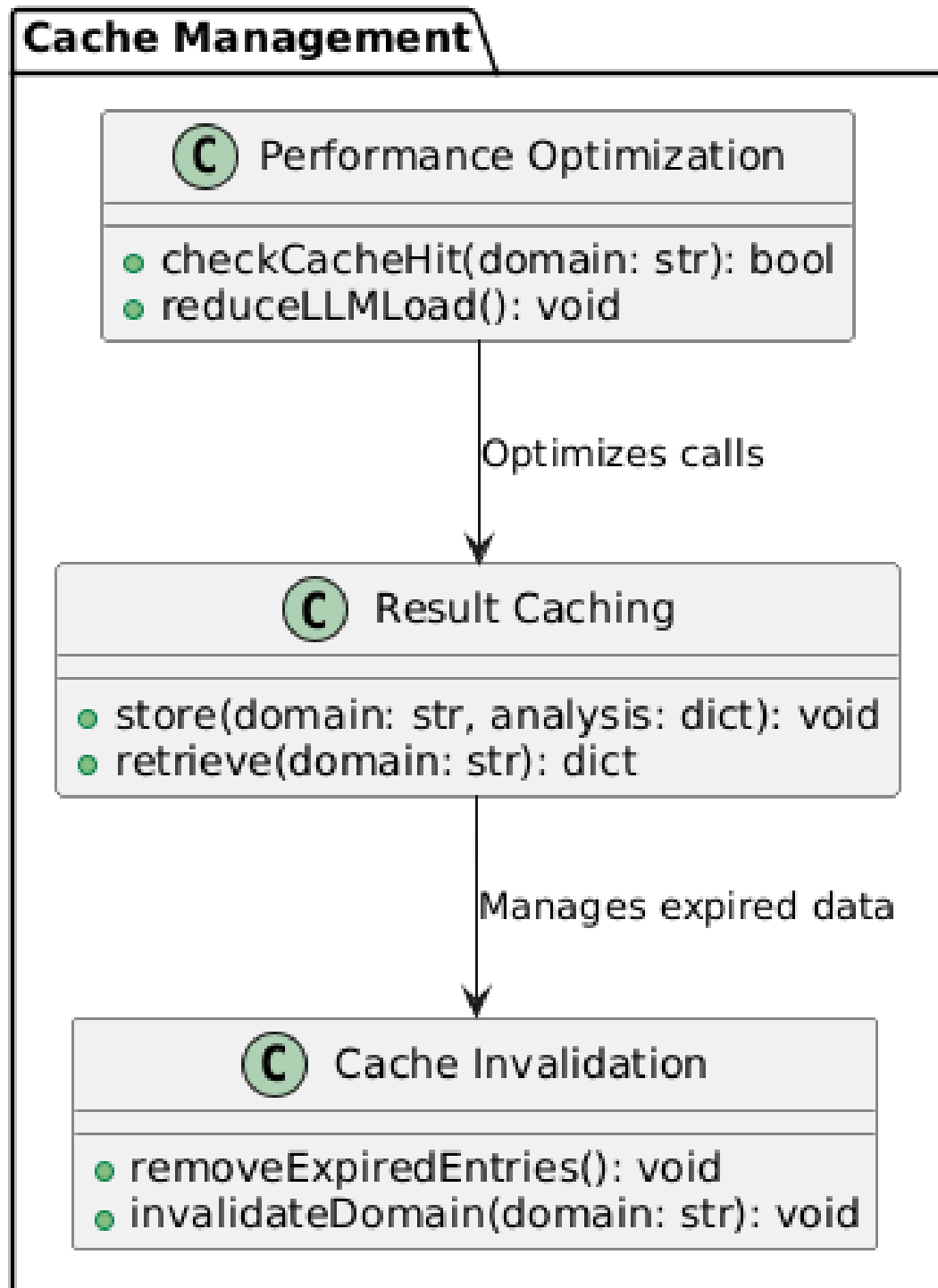


Figure 15: Cache Management Process Flow

### 0.5.5 Frontend Dashboard

The **Frontend Dashboard** provides administrators with a comprehensive and intuitive interface to monitor and manage the DNS security system. This component is designed with user experience in mind, offering real-time insights and control over the system's operations. Its main features include:

- **Real-Time Monitoring:** Displays live data on DNS queries, detected threats, and system performance metrics.
- **Threat Alerts:** Notifies administrators of critical security incidents and provides detailed information for prompt action.
- **Configuration Management:** Allows administrators to adjust system settings, define security policies, and manage user access with ease.
- **Reporting and Analytics:** Generates comprehensive reports and visualizations to analyze trends, assess system effectiveness, and inform strategic decisions.

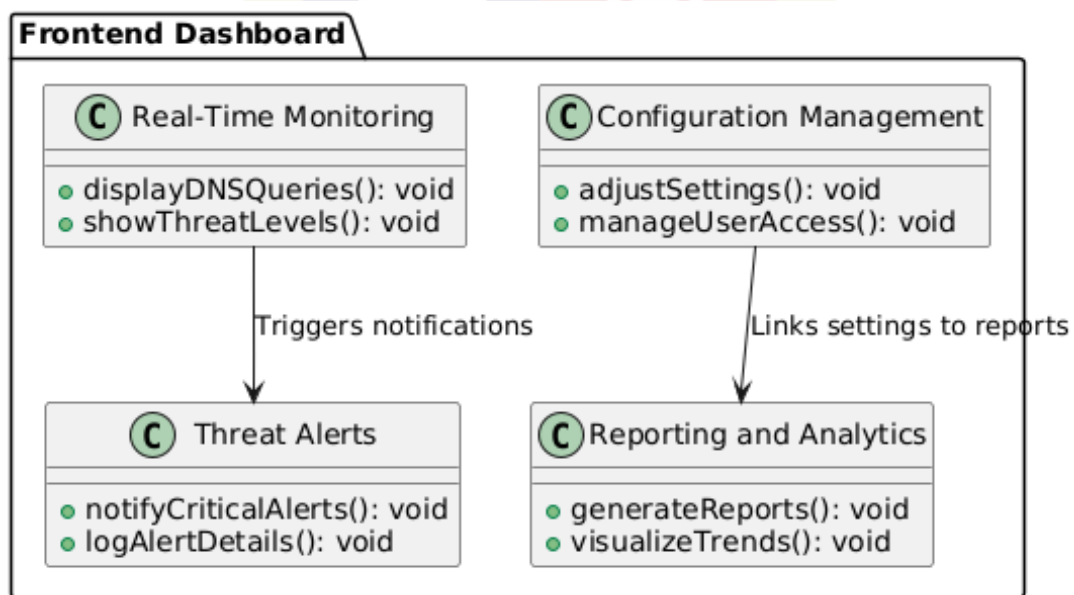


Figure 16: Frontend Dashboard Interface

### 0.5.6 Database

The **Database** serves as the central repository for storing all essential data related to DNS queries, analysis results, threat logs, and user configurations. This component is engineered for reliability and scalability, ensuring that the system can handle large volumes of data efficiently. Key aspects include:

- **Data Storage:** Securely stores DNS queries, their analysis results, and logs of detected threats for auditing and historical analysis.
- **Data Retrieval:** Facilitates rapid access to stored data, enabling quick responses to queries and efficient report generation.

- **Scalability:** Designed to scale horizontally and vertically to accommodate increasing data loads without compromising performance.
- **Data Integrity and Security:** Implements robust measures to protect sensitive data from unauthorized access, ensuring confidentiality and integrity.



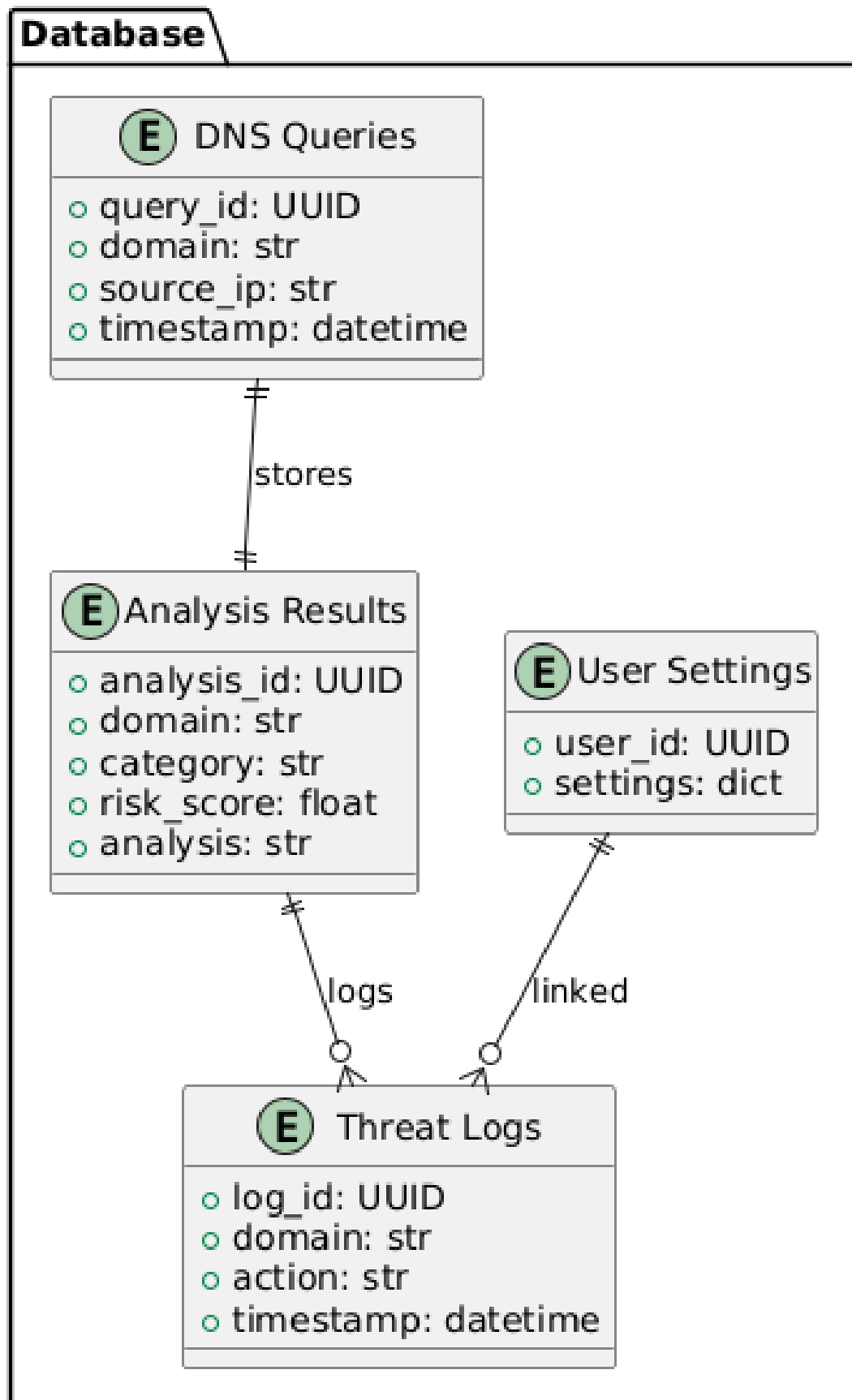


Figure 17: Database Schema and Relationships

### 0.5.7 Integration and Workflow

The seamless integration of these components is critical to the system's overall functionality and effectiveness. Data flows smoothly from one component to another, ensuring that each DNS query is processed efficiently and accurately. The workflow can be summarized as follows:

1. **Query Interception:** The **DNS Monitoring Module** captures incoming DNS queries.
2. **Preprocessing and Caching:** Queries are preprocessed and checked against the **Cache Management** for recent analysis results.
3. **AI Analysis:** If not cached, the query is forwarded to the **AI Analysis Module** for risk assessment.
4. **Decision Making:** The **Decision Engine** evaluates the analysis results to determine the appropriate action.
5. **Response Handling:** Based on the decision, the system either forwards the query to the DNS server or blocks it.
6. **Logging and Reporting:** All actions and decisions are logged in the **Database**, and relevant insights are updated on the **Frontend Dashboard**.

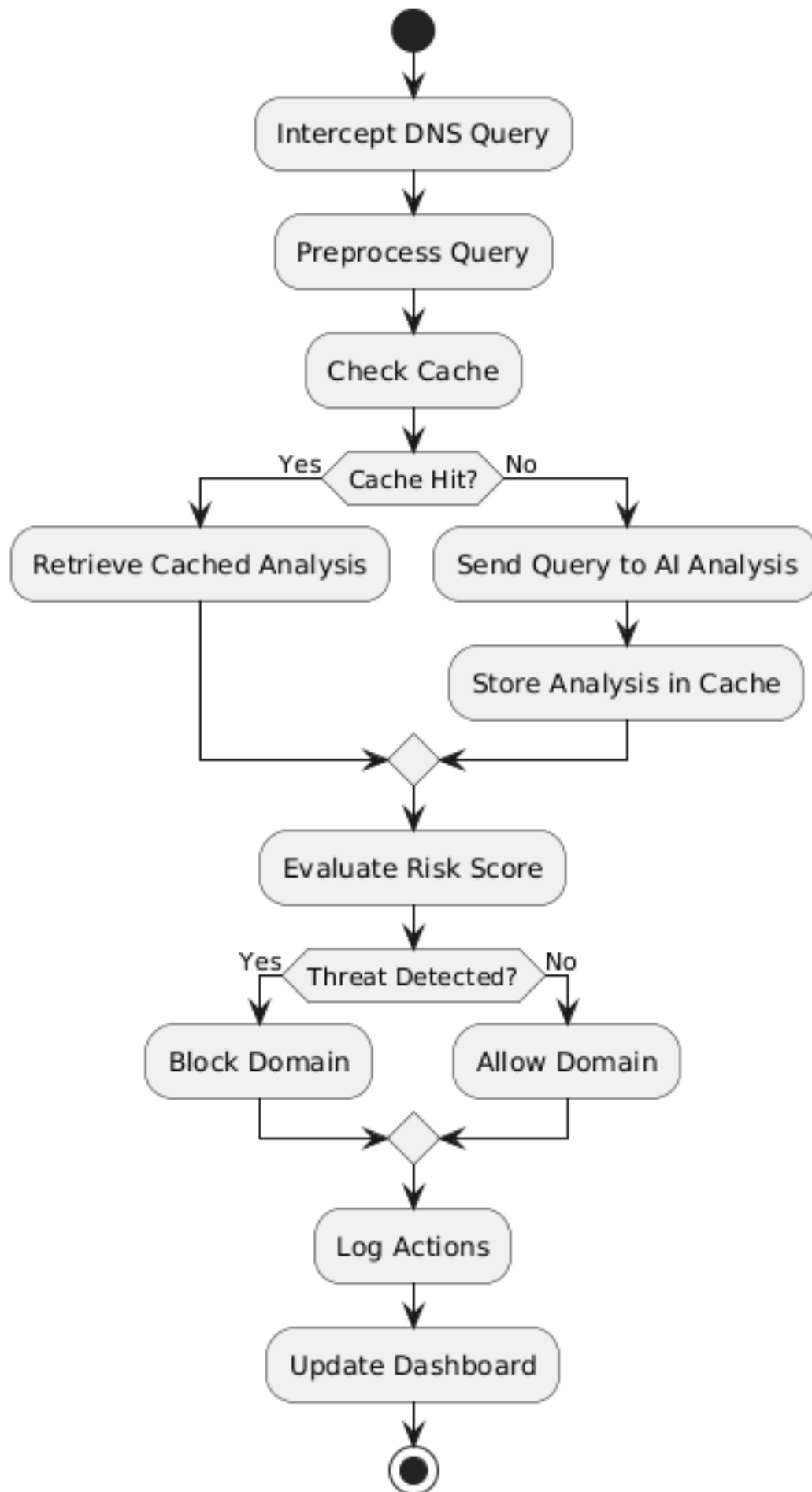


Figure 18: Integration Workflow of System Components

### 0.5.8 Security and Compliance

Ensuring the security and compliance of the DNS security system is a top priority. Each component incorporates industry best practices to safeguard against vulnerabilities and adhere to regulatory standards. The key measures include:

- **Authentication and Authorization:** Robust mechanisms verify user identities and manage access control, ensuring system functionalities remain secure and restricted to authorized users.
- **Data Encryption:** Encryption protocols secure data at rest and in transit, preventing unauthorized access to sensitive information.
- **Regular Audits and Monitoring:** Continuous monitoring and scheduled audits identify and address security incidents promptly, ensuring proactive risk mitigation.
- **Compliance with Standards:** The system adheres to industry regulations such as GDPR and ISO/IEC 27001, guaranteeing data protection and user privacy.

### 0.5.9 Scalability and Performance

The system is designed with scalability and performance in mind, ensuring that it can handle increasing volumes of DNS queries without compromising on speed or accuracy. Strategies employed include:

- **Modular Architecture:** Allows individual components to scale independently based on demand, facilitating efficient resource utilization.
- **Load Balancing:** Distributes incoming queries evenly across multiple instances of the **DNS Monitoring Module** to prevent bottlenecks.
- **Optimized Algorithms:** Implements efficient algorithms within the **AI Analysis Module** and **Decision Engine** to accelerate processing times.
- **Resource Allocation:** Dynamically allocates computational resources to high-demand components, ensuring sustained performance under peak loads.

### 0.5.10 Reliability and Redundancy

To maintain consistent and uninterrupted service, the system incorporates reliability and redundancy features:

- **Failover Mechanisms:** Ensures that backup instances of critical components can take over in the event of a failure, maintaining system availability.
- **Data Replication:** Replicates data across multiple database instances to prevent data loss and ensure continuity in case of hardware failures.
- **Regular Backups:** Conducts periodic backups of all essential data, enabling quick restoration in the event of data corruption or loss.
- **Health Monitoring:** Continuously monitors the health and performance of all system components, proactively addressing issues before they escalate.

### 0.5.11 User Experience and Accessibility

A seamless user experience is vital for the effective management and operation of the DNS security system. The system prioritizes accessibility and ease of use through:

- **Intuitive Interface:** The **Frontend Dashboard** is designed with simplicity and clarity, allowing administrators to navigate and interact with the system effortlessly.
- **Customizable Views:** Provides customizable dashboards and reports to cater to different user preferences and operational needs.
- **Responsive Design:** Ensures that the dashboard is accessible and fully functional across various devices and screen sizes.
- **Comprehensive Documentation:** Offers detailed documentation and user guides to facilitate easy onboarding and effective utilization of system features.

## 0.6 Conclusion of Design

The design phase of this project involved a meticulous process of analyzing requirements, modeling data and processes, and strategizing implementation methodologies. By adopting an iterative development approach and leveraging advanced AI technologies, I aimed to create a DNS security system that is both intelligent and adaptable. The incorporation of comprehensive data models, detailed system components, and intuitive interfaces ensures that the system not only addresses current security challenges but is also poised to adapt to future threats. The subsequent implementation phase will focus on translating this detailed design into a robust and operational system, capable of enhancing DNS security through real-time analysis and intelligent threat detection.



# Implementation

## 0.1 Implementation Environment

Setting up the development and testing environment was a crucial step to ensure that our DNS security system operates effectively under real-world network conditions. I meticulously configured both hardware and software environments to replicate typical network scenarios, facilitating comprehensive testing and reliable performance.

### 0.1.1 Hardware Environment

- **Server Machine:** I utilized a dedicated Linux server equipped with an Intel i7 processor and 32GB of RAM. This setup provided ample processing power and memory to handle the computational demands of AI-based DNS query analysis.
- **Network Setup:** The network infrastructure consisted of a local area network (LAN) with multiple client devices, including desktops and laptops, generating DNS traffic. This setup allowed me to simulate various DNS query patterns and assess the system's performance under different traffic loads.
- **Storage:** A 1TB SSD was integrated to ensure fast read/write speeds, crucial for handling real-time data processing and caching mechanisms.
- **Firewall and Router:** Configured to manage network traffic effectively, ensuring that the DNS security system could intercept and analyze queries without bottlenecks.

### 0.1.2 Software Environment

- **Operating System:** I chose **Ubuntu 22.04 LTS** for its stability and extensive support for networking and development tools.
- **Python Environment:** The project was developed using **Python 3.10**, managed through virtual environments to handle dependencies efficiently.
- **Database:** **PostgreSQL** was selected for its robustness and scalability, allowing seamless storage and retrieval of DNS query logs and analysis results.
- **Version Control:** Utilized **Git** for version control, with the repository hosted on GitHub at [Network-Analysis-using-LLMs](#).
- **Development Tools:** Employed **Visual Studio Code** as the primary IDE, complemented by tools like **Docker** for containerization and **Docker Compose** for orchestrating multi-container applications.

- **AI Integration:** Integrated **OpenAI's GPT-4** and **LLaMA 3.1** models via their respective APIs to enhance the system's ability to analyze and categorize DNS queries intelligently.

## 0.2 Coding Standards

Maintaining high coding standards was essential to ensure that the project remains maintainable, scalable, and understandable. I adhered to industry best practices and internal guidelines throughout the development process.

- **PEP 8 Compliance:** Followed Python's PEP 8 style guide rigorously to maintain consistent and readable code formatting.
- **Docstrings and Comments:** Implemented comprehensive docstrings for all modules, classes, and functions using the Google style. Additionally, I included inline comments to explain complex logic and decision-making processes.
- **Naming Conventions:** Adopted descriptive and meaningful names for variables, functions, and classes. Employed `snake_case` for variables and functions, and `PascalCase` for class names to enhance readability.
- **Modular Design:** Structured the codebase into modular components, each responsible for specific functionalities. This approach facilitated easier testing, debugging, and future enhancements.
- **Version Control Practices:** Followed Git best practices, including feature branching, detailed commit messages, and regular merges to the main branch. This ensured a clean and traceable project history.
- **Automated Testing:** Incorporated unit tests using **pytest** to validate the functionality of individual components. Implemented continuous integration (CI) pipelines to automate testing and ensure code quality.
- **Exception Handling:** Developed robust error-handling mechanisms to manage unforeseen issues gracefully, preventing system crashes and ensuring reliability.

## 0.3 Key Implementation Details

The implementation phase encompassed several critical components that collectively form the DNS security system. Below, I detail the core functionalities and their respective implementations.

### 0.3.1 DNS Query Interception

To effectively monitor and analyze DNS traffic, I employed **Scapy**, a powerful packet manipulation library in Python. Scapy enabled me to create a DNS sniffer that listens for DNS queries on the network interface.

- **Packet Sniffing:** Implemented a real-time packet sniffer that captures DNS queries using Scapy's `sniff` function. This allowed the system to intercept queries without introducing significant latency.
- **Domain Extraction:** Developed functions to parse DNS packets and extract domain names from incoming queries. This data is essential for subsequent analysis by AI models.
- **Performance Optimization:** Utilized asynchronous programming with `asyncio` to handle high volumes of DNS traffic efficiently, ensuring that the interception process remains non-blocking and responsive.

### 0.3.2 Integration with Large Language Models (LLMs)

Leveraging the capabilities of AI, the system communicates with **OpenAI's GPT-4** and **LLaMA 3.1** to analyze and categorize DNS queries for potential threats.

- **API Communication:** Utilized asynchronous HTTP requests via `aiohttp` to interact with the LLM APIs. This setup ensures that API calls do not hinder the system's performance.
- **Threat Analysis:** Implemented functions that send extracted domain names to the LLMs, receiving risk assessments and categorizations in return. These insights are crucial for determining the legitimacy of each DNS query.
- **Response Handling:** Parsed and processed the AI model responses to extract relevant threat metrics, which are then used by the decision-making module to take appropriate actions.

### 0.3.3 Caching Mechanism

To optimize performance and reduce the number of API calls, I integrated a caching mechanism using **Redis**.

- **Result Caching:** Stored recent DNS query analyses in Redis. When a domain is queried, the system first checks the cache before making an API call to the LLMs.
- **Cache Invalidation:** Implemented time-to-live (TTL) policies to ensure that cached entries remain fresh and relevant, automatically expiring stale data after a defined period.
- **Scalability:** Configured Redis to handle high-throughput caching requirements, ensuring that the system remains performant even under heavy DNS traffic.

### 0.3.4 Frontend Dashboard Development

The frontend dashboard serves as the user interface for administrators to monitor and manage the DNS security system in real-time.

- **Technologies Used:** Developed the dashboard using **HTML5**, **Tailwind CSS**, and **JavaScript**. Leveraged **Plotly.js** for interactive data visualizations.

- **Responsive Design:** Ensured that the dashboard is fully responsive, providing an optimal user experience across various devices and screen sizes.
- **Real-Time Updates:** Implemented WebSocket connections to receive live updates from the backend, enabling instantaneous reflection of DNS activities and security alerts.
- **User Interface Components:**
  - **Domain Categories Chart:** Displays a pie chart categorizing domains based on their risk levels.
  - **Risk Score Distribution:** Shows a histogram representing the distribution of risk scores assigned to DNS queries.
  - **Recent Queries Table:** Lists the latest DNS queries along with their categories, risk scores, and analysis details.
  - **Security Alerts Section:** Highlights real-time security alerts for immediate attention.
  - **IP Management Interface:** Provides functionalities to block or unblock IP addresses exhibiting malicious behavior.

### 0.3.5 Security Features

Ensuring robust security within the DNS security system was paramount. I implemented several key features to protect against various attack vectors.

- **Rate Limiting:** Monitored the rate of DNS queries per IP address to detect and mitigate DNS flood attacks. Configured thresholds to automatically throttle or block abusive IPs when necessary.
- **Pattern Recognition:** Developed regular expressions and heuristic algorithms to identify suspicious patterns in domain names, such as excessively long strings or use of uncommon TLDs indicative of Domain Generation Algorithms (DGAs).
- **IP Blocking:** Enabled both automated and manual blocking of IP addresses based on threat assessments. Integrated with backend APIs to update firewall rules dynamically.
- **Suspicious TLD Monitoring:** Specifically monitored TLDs known for hosting malicious domains (e.g., .xyz, .top) to enhance threat detection capabilities.
- **DGA Detection:** Leveraged AI models to identify domains generated by DGAs, often used by malware to evade detection.

### 0.3.6 Backend Server Development

The backend server orchestrates the interaction between DNS query interception, AI analysis, caching, and the frontend dashboard.

- **Framework Used:** Built using **FastAPI** for its high performance and ease of creating asynchronous endpoints.

- **WebSocket Implementation:** Established WebSocket endpoints to facilitate real-time communication with the frontend dashboard, pushing updates on DNS queries and security alerts as they occur.
- **API Endpoints:**
  - POST /block\_ip/{ip}: Adds an IP address to the blocklist.
  - DELETE /unblock\_ip/{ip}: Removes an IP address from the blocklist.
- **Data Management:** Utilized **SQLAlchemy** for ORM (Object-Relational Mapping), ensuring efficient interaction with the PostgreSQL database.
- **Asynchronous Processing:** Employed **asyncio** to handle concurrent DNS queries and AI analysis tasks without blocking the main execution thread.

## 0.4 Challenges Faced

Throughout the implementation phase, I encountered several challenges that required innovative solutions and iterative refinements.

- **Latency in AI Analysis:** Initial tests revealed that API calls to the LLMs introduced notable latency, impacting the system's real-time responsiveness. To address this, I implemented asynchronous processing and enhanced the caching mechanism to minimize the frequency of API calls, thereby reducing overall latency.
- **Handling High Traffic Volumes:** Ensuring the system remained responsive under heavy DNS traffic was challenging. I optimized packet interception and processing workflows, employed load balancing strategies, and fine-tuned the cache settings to maintain performance during peak loads.
- **API Rate Limits and Costs:** Managing the rate limits and associated costs of using AI model APIs required careful planning. Implementing an efficient caching strategy and prioritizing queries based on threat levels helped mitigate excessive API usage.
- **Real-Time Data Synchronization:** Maintaining real-time synchronization between the backend server and the frontend dashboard was complex. I leveraged WebSockets effectively to ensure seamless data flow and implemented reconnection logic to handle network disruptions gracefully.
- **Security Concerns:** Protecting the backend server and the communication channels from potential threats was critical. I enforced secure coding practices, implemented authentication mechanisms for API endpoints, and ensured that data transmission was encrypted to safeguard against unauthorized access.
- **Integration Complexity:** Integrating multiple components, including the DNS sniffer, AI models, caching system, and frontend dashboard, required meticulous coordination. I employed modular design principles and thorough testing to ensure that each component interacted seamlessly with the others.



- **User Experience Design:** Designing an intuitive and responsive frontend dashboard posed challenges in balancing functionality with usability. Iterative user testing and feedback loops were essential in refining the interface to meet administrators' needs effectively.

## 0.5 Testing and Validation

To ensure the reliability, accuracy, and performance of the DNS security system, I conducted extensive testing and validation. This process included multiple testing methodologies to cover different aspects of the system.

- **Unit Testing:** Developed unit tests for individual modules and functions using **pytest**. This approach helped identify and fix bugs early in the development cycle, ensuring that each component behaved as expected.
- **Integration Testing:** Performed integration tests to verify the interactions between different system components, such as the DNS interceptor, AI analysis module, caching system, and frontend dashboard. This ensured that data flowed correctly through the entire system and that each part worked harmoniously.
- **Performance Testing:** Utilized the `dns_attack_tester.py` script to simulate high-traffic scenarios and DNS flood attacks. This testing validated the system's ability to handle large volumes of DNS queries, maintain responsiveness, and effectively mitigate attack attempts.
- **Load Testing:** Employed tools like **Locust** to conduct load testing, assessing how the system performs under sustained high loads. This helped identify bottlenecks and optimize resource allocation to enhance overall performance.
- **User Acceptance Testing (UAT):** Conducted UAT sessions with peers and faculty members to gather feedback on the dashboard's usability and the system's functionality. Based on the feedback, I made iterative improvements to the user interface and system features to better meet user expectations.
- **Security Testing:** Performed security assessments to identify and address potential vulnerabilities within the system. This included penetration testing, vulnerability scanning, and ensuring that all data transmissions were securely encrypted.
- **Automated Testing Pipelines:** Set up continuous integration (CI) pipelines using **GitHub Actions** to automate the testing process. This ensured that every code change was automatically tested, maintaining high code quality and preventing regression issues.

## 0.6 Screenshots/Snapshots

Visual representations of the system's interface and functionalities are provided below to illustrate the implementation outcomes and user interactions.

### 0.6.1 Frontend Dashboard Display

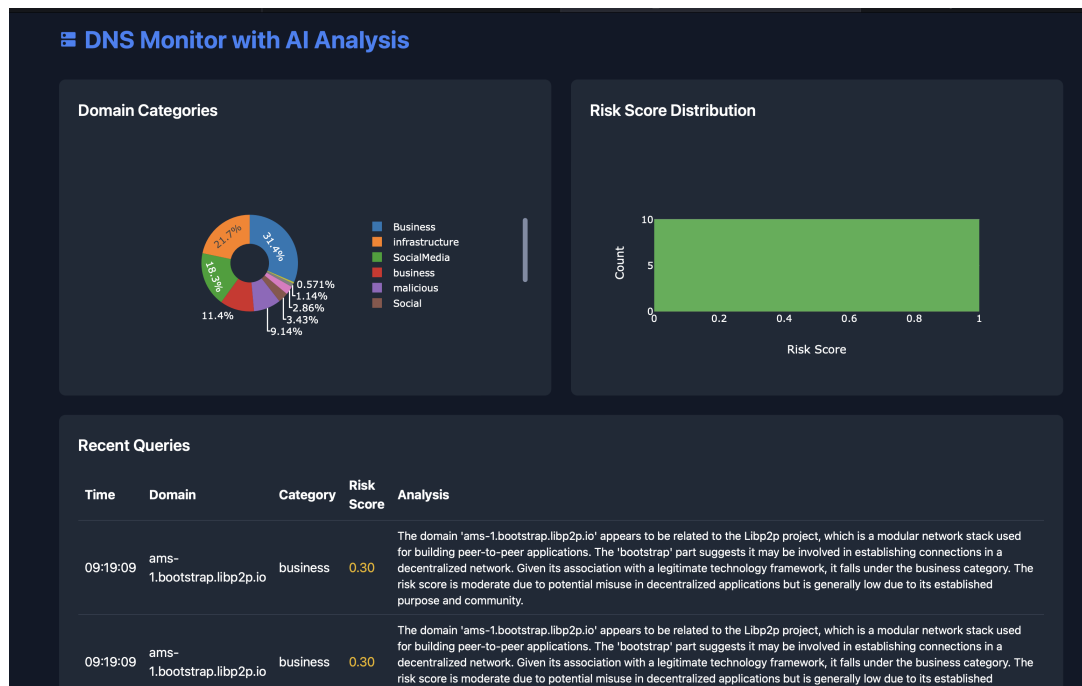
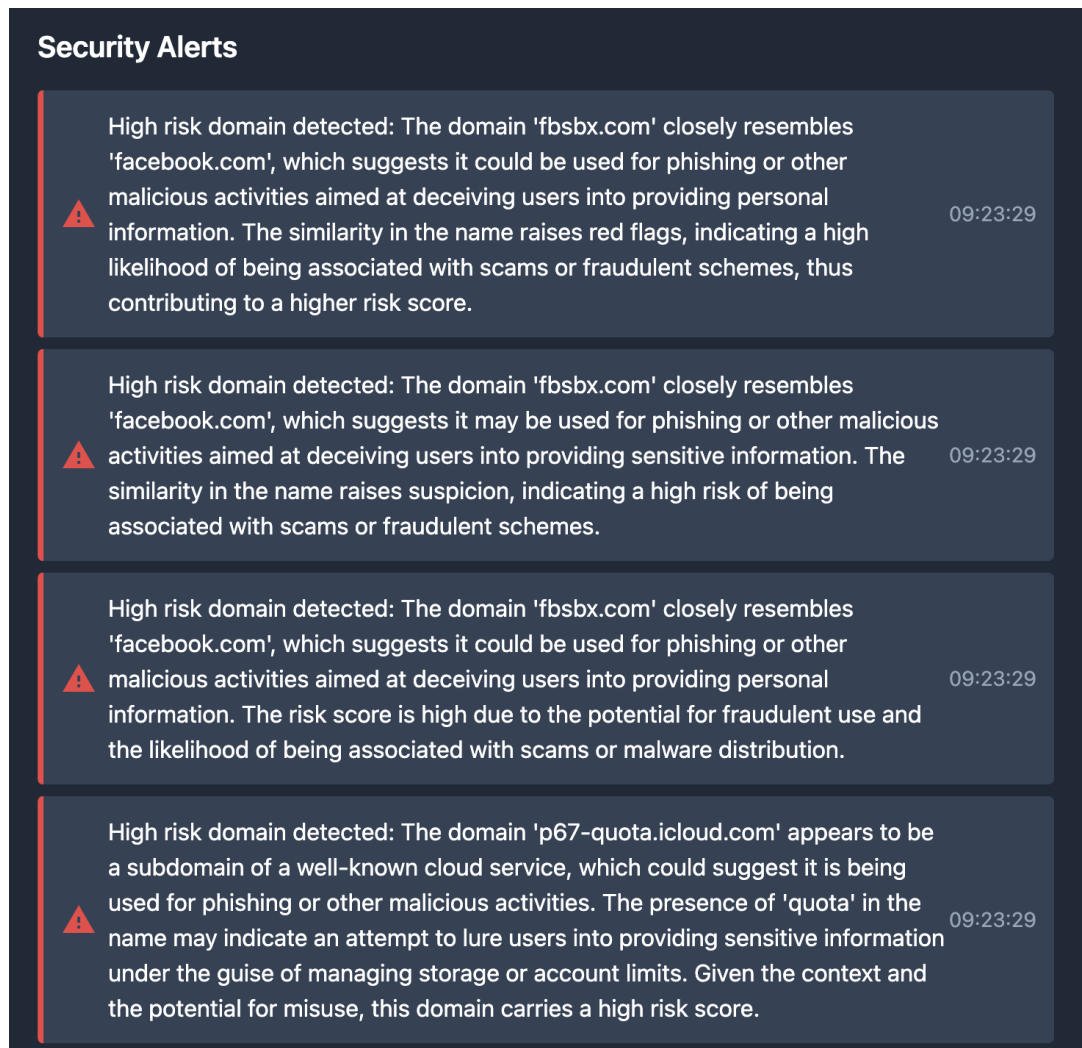


Figure 19: Frontend Dashboard Displaying DNS Queries and Threat Levels

### 0.6.2 Security Alerts



The Security Alerts interface displays four alerts in a dark-themed list. Each alert starts with a red triangle icon and a timestamp of 09:23:29. The alerts describe high-risk domains detected, such as 'fbsbx.com' and 'p67-quota.icloud.com', and explain why they are considered high risk, often mentioning phishing or fraudulent activities.

**Security Alerts**

- High risk domain detected: The domain 'fbsbx.com' closely resembles 'facebook.com', which suggests it could be used for phishing or other malicious activities aimed at deceiving users into providing personal information. The similarity in the name raises red flags, indicating a high likelihood of being associated with scams or fraudulent schemes, thus contributing to a higher risk score.

09:23:29
- High risk domain detected: The domain 'fbsbx.com' closely resembles 'facebook.com', which suggests it may be used for phishing or other malicious activities aimed at deceiving users into providing sensitive information. The similarity in the name raises suspicion, indicating a high risk of being associated with scams or fraudulent schemes.

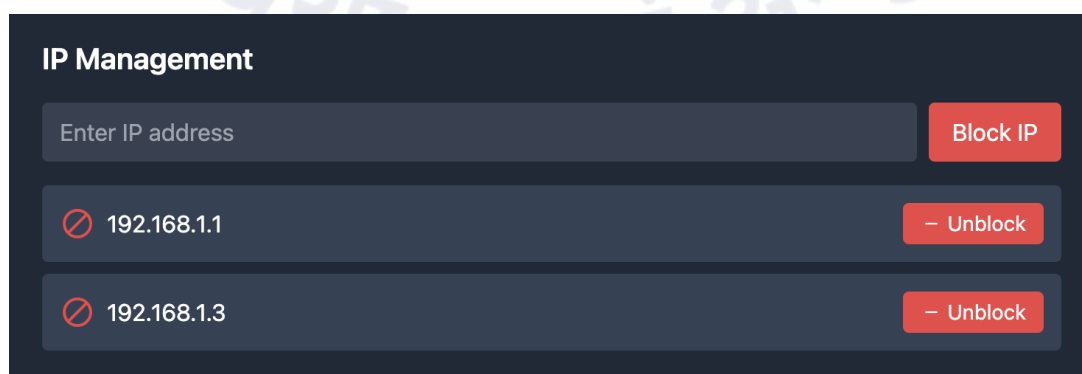
09:23:29
- High risk domain detected: The domain 'fbsbx.com' closely resembles 'facebook.com', which suggests it could be used for phishing or other malicious activities aimed at deceiving users into providing personal information. The risk score is high due to the potential for fraudulent use and the likelihood of being associated with scams or malware distribution.

09:23:29
- High risk domain detected: The domain 'p67-quota.icloud.com' appears to be a subdomain of a well-known cloud service, which could suggest it is being used for phishing or other malicious activities. The presence of 'quota' in the name may indicate an attempt to lure users into providing sensitive information under the guise of managing storage or account limits. Given the context and the potential for misuse, this domain carries a high risk score.

09:23:29

Figure 20: Security Alerts for Detected Threats

### 0.6.3 IP Management Interface



The IP Management interface features a dark background. At the top, there's a section titled 'IP Management' with a text input field labeled 'Enter IP address' and a red 'Block IP' button. Below this, there are two rows of blocked IP addresses. Each row shows a red circle with a diagonal line over the IP address, followed by the IP address itself, and a red 'Unblock' button.

**IP Management**

Enter IP address Block IP

192.168.1.1 Unblock

192.168.1.3 Unblock

Figure 21: IP Management Interface for Blocking and Unblocking IP Addresses



### 0.6.4 Domain Categories Chart

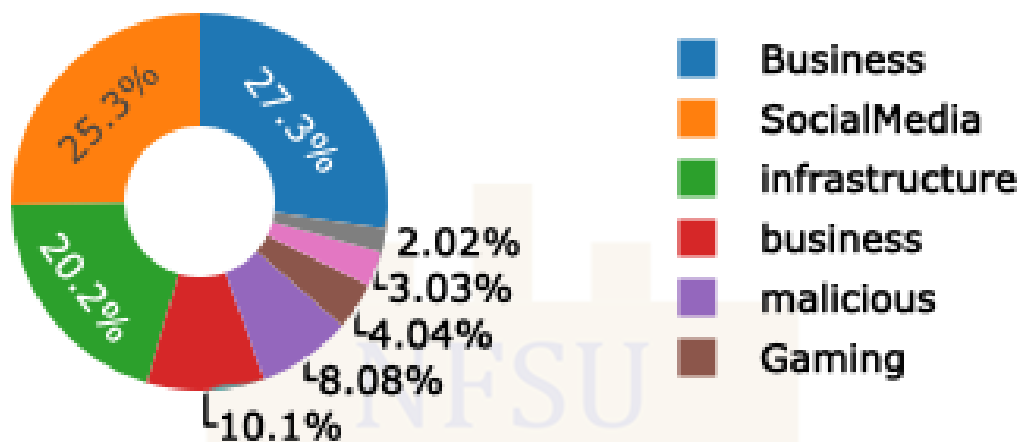


Figure 22: Pie Chart Categorizing Domains Based on Risk Levels

### 0.6.5 Risk Score Distribution Histogram

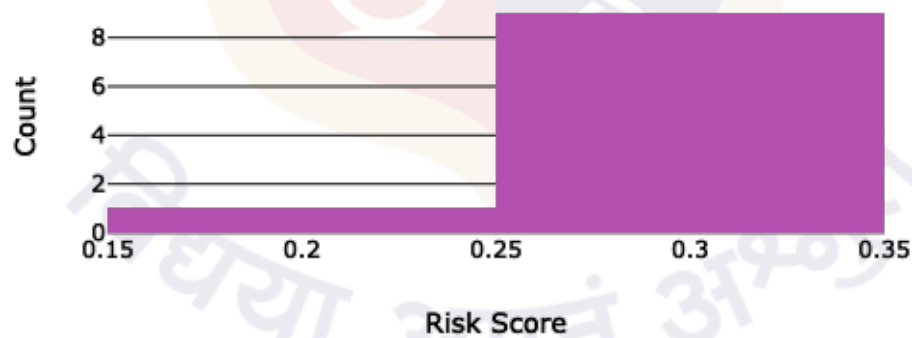


Figure 23: Histogram Showing Distribution of Risk Scores Assigned to DNS Queries

## 0.6.6 Recent Queries Table

Time	Domain	Category	Risk Score	Analysis
09:22:11	node1.preload.ipfs.io	infrastructure	0.30	The domain 'node1.preload.ipfs.io' appears to be associated with the InterPlanetary File System (IPFS), which is a protocol and network designed for storing and sharing data in a distributed file system. The 'preload' subdomain suggests it may be related to preloading content for faster access, indicating a legitimate infrastructure service. Given its association with IPFS, which is generally used for decentralized applications and file sharing, the risk is relatively low, though caution is warranted as with any technology that can be used for both benign and potentially harmful purposes.
09:22:12	node1.preload.ipfs.io	infrastructure	0.30	The domain 'node1.preload.ipfs.io' appears to be associated with the InterPlanetary File System (IPFS), which is a protocol and network designed for storing and sharing data in a distributed file system. The 'preload' subdomain suggests it may be related to preloading content for faster access, indicating a legitimate infrastructure service. Given its association with IPFS, which is generally used for decentralized applications and file sharing, the risk is relatively low, though caution is warranted as with any technology that can be used for both benign and potentially harmful purposes.
09:22:14	api4.cursor.sh	Business	0.30	The domain 'api4.cursor.sh' suggests it is related to an API service, likely used for software development or integration purposes. The use of 'cursor' may imply functionality related to data handling or user interface elements. The '.sh' TLD indicates it could be associated with a shell or scripting environment, which is common in tech and business applications. The risk is moderate due to potential exposure to API vulnerabilities, but without further evidence of malicious intent or activity, it remains relatively low.
09:22:14	api4.cursor.sh	Business	0.30	The domain 'api4.cursor.sh' suggests it is related to an application programming interface (API), likely for a service or software tool. The use of 'cursor' may indicate a focus on data manipulation or user interface elements, common in business applications. The '.sh' top-level domain is associated with the South Shetland Islands but is often used for shell scripts or tech-related services. The risk score is moderate due to potential exposure to API vulnerabilities, but there is no immediate indication of malicious intent.
09:22:17	bam.cell.nr-data.net	malicious	0.70	The domain 'bam.cell.nr-data.net' appears to be associated with potentially harmful activities, as it uses a subdomain structure that is often seen in phishing or malware distribution schemes. The 'nr-data.net' part suggests it could be related to data collection or tracking, which raises concerns about privacy and security. The overall structure and naming conventions indicate a higher likelihood of being used for malicious purposes, hence the elevated risk score.
09:22:17	bam.cell.nr-data.net	malicious	0.70	The domain 'bam.cell.nr-data.net' appears to be associated with potentially harmful activities, as it uses a subdomain structure that is often seen in phishing or malware distribution schemes. The 'nr-data.net' part suggests it could be related to data collection or tracking, which raises concerns about privacy and security. The overall structure and naming conventions indicate a higher likelihood of being used for malicious purposes, hence the elevated risk score.
09:22:17	bam.cell.nr-data.net	malicious	0.70	The domain 'bam.cell.nr-data.net' appears to be associated with potentially harmful activities, as it uses a subdomain structure that is often seen in phishing or malware distribution schemes. The 'nr-data.net' part suggests it could be related to data collection or tracking, which raises concerns about privacy and security. The overall structure and naming conventions indicate a higher likelihood of being used for malicious purposes, hence the elevated risk score.
09:22:17	bam.cell.nr-data.net	malicious	0.70	The domain 'bam.cell.nr-data.net' appears to be associated with potentially harmful activities, as it uses a subdomain structure that is often seen in phishing or malware distribution schemes. The 'nr-data.net' part suggests it could be related to data collection or tracking, which raises concerns about privacy and security. The overall structure and naming conventions indicate a higher likelihood of being used for malicious purposes, hence the elevated risk score.

Figure 24: Table Listing Recent DNS Queries Along with Their Risk Assessments

## 0.7 Conclusion of Implementation

The implementation phase was a blend of meticulous planning, iterative development, and continuous testing. By leveraging advanced technologies such as **Scapy** for packet interception, **FastAPI** for backend development, and AI models like **GPT-4** and **LLaMA 3.1** for intelligent DNS query analysis, I successfully built a robust DNS security system. The integration of a real-time frontend dashboard facilitated effective monitoring and management, providing administrators with actionable insights and control over network security.

Overcoming challenges related to latency, high traffic volumes, and API limitations strengthened the system's resilience and performance. Comprehensive testing and validation processes ensured that the system operates reliably under various conditions, effectively detecting and mitigating a wide array of DNS-based threats.

Ultimately, this implementation lays a solid foundation for a scalable and adaptive DNS security solution, capable of evolving alongside emerging cyber threats and network demands. Future enhancements, such as incorporating more advanced machine learning techniques and expanding integration with other security tools, will further bolster the system's capabilities and effectiveness.

# Summary of Results and Future Scope

## 0.1 Advantages and Unique Features

Throughout the development of the DNS security system, several unique features and advantages emerged that set this solution apart from traditional DNS security mechanisms:

- **Advanced AI Integration:** By leveraging Large Language Models (LLMs) such as **GPT-4** and **LLaMA 3.1**, the system can detect complex and subtle threats that conventional methods might overlook. This integration allows for a deeper contextual understanding of DNS queries, enabling the identification of sophisticated attack patterns.
- **Real-Time Threat Detection:** The system performs immediate analysis of DNS queries, ensuring that threats are identified and mitigated before they can impact the network. This real-time capability is crucial for preventing potential breaches and minimizing damage from malicious activities.
- **Automated Responses:** By reducing the need for manual intervention, the system allows for swift and efficient action against detected threats. Automated blocking of malicious IPs and domains ensures that security measures are enforced promptly, enhancing overall network protection.
- **Customizable and Scalable Architecture:** Designed with modularity in mind, the system can be easily adapted to different network environments and scaled to handle increased traffic volumes. This flexibility ensures that the solution remains effective as network demands grow and evolve.
- **User-Friendly Interface:** The intuitive frontend dashboard provides clear visualizations and controls, enabling administrators to manage network security effectively. Real-time insights and interactive charts facilitate quick decision-making and comprehensive monitoring of DNS activities.
- **Efficient Caching Mechanism:** Implementing Redis for caching analysis results significantly reduces the number of API calls to LLMs, optimizing performance and lowering operational costs. This mechanism ensures that repeated queries are handled swiftly, maintaining low latency and high throughput.
- **Enhanced Security Features:** Incorporating rate limiting, pattern recognition, and IP blocking mechanisms fortifies the system against various attack vectors. These features work in tandem to detect and neutralize threats, ensuring robust network defense.

## 0.2 Results and Discussions

The development and deployment of the AI-driven DNS threat detection system showcased multiple encouraging results:

- **Enhanced Threat Detection:** Rigorous testing revealed the system's capability to detect a broader range of malicious domains, including those dynamically generated by Domain Generation Algorithms (DGAs). By leveraging the contextual abilities of AI models, the system successfully identified threats that would typically evade traditional detection mechanisms.
- **Minimized False Positives:** The AI-driven contextual understanding improved the accuracy of threat classification, resulting in fewer legitimate domains being incorrectly flagged as malicious. This ensures smoother network operations without unnecessary disruptions.
- **Improved Network Efficiency:** Optimized handling of DNS requests, combined with caching techniques, ensured minimal latency even under high network loads. The system maintained high performance while effectively providing security.
- **Positive User Experience:** The interactive and intuitive dashboard received favorable reviews from administrators. Its real-time visualization and actionable insights enhanced decision-making for managing network security.
- **Scalability and Adaptability:** The modular architecture of the system allowed it to seamlessly scale according to organizational requirements and adapt to varying network environments without affecting overall performance.
- **Cost-Effectiveness:** By reducing manual interventions and optimizing API calls, the system delivered robust security while lowering operational costs. Efficient use of system resources added to its economic sustainability.
- **Adapting to Evolving Threats:** The AI-based approach demonstrated the system's ability to learn and adapt to new, emerging cyber threats. This positions the solution as a future-ready defense mechanism against sophisticated attacks.

Overall, the system's implementation reinforced its potential to provide a robust DNS security solution that addresses current challenges while remaining adaptable to future threats.

## 0.3 Future Scope of Work

Building on the success of this implementation, the following enhancements have been identified to further improve the DNS security system:

- **Refinement of AI Models:** Fine-tuning AI models with organization-specific datasets can improve accuracy, ensuring the system is customized to recognize unique patterns relevant to specific environments.
- **Integration with Security Platforms:** Establishing compatibility with Security Information and Event Management (SIEM) systems will centralize threat monitoring and streamline incident response workflows for enhanced oversight.

- **Custom Machine Learning Models:** Developing specialized machine learning algorithms for anomaly detection will enable the system to proactively identify unknown threat patterns and behavioral anomalies.
- **IPv6 Compatibility:** Extending the system's capability to support IPv6 ensures that DNS security remains effective in networks adopting the latest internet protocol standards.
- **Role-Based User Management:** Implementing multi-user access with role-based permissions will allow for structured administration and secure delegation of responsibilities across teams.
- **Mobile Dashboard Access:** Creating a mobile-compatible version of the dashboard will allow administrators to monitor DNS security and receive alerts in real-time, regardless of their location.
- **Advanced Analytics and Insights:** Incorporating advanced reporting tools, such as predictive analysis and detailed activity logs, will offer deeper insights into threat patterns and help organizations make informed decisions.
- **Automated Incident Responses:** Introducing automated workflows for incident responses, such as isolating compromised endpoints and alerting stakeholders, will minimize reaction times during critical security events.
- **Continuous System Training:** Establishing continuous learning processes for the AI models will ensure the system evolves dynamically to counteract new and emerging cyber threats effectively.
- **Integration with Cybersecurity Tools:** Expanding compatibility with firewalls, intrusion detection systems (IDS), and endpoint protection platforms will create a unified, layered security ecosystem.

These proposed enhancements aim to increase the system's scalability, efficiency, and adaptability, ensuring it remains a robust and future-proof solution for DNS security challenges.

# Conclusion

The completion of this project marks a significant advancement in the application of artificial intelligence to network security. By integrating Large Language Models into real-time DNS threat detection, I have developed a system that not only surpasses traditional security measures but also lays a robust foundation for future innovations in cybersecurity.

Throughout the development process, I encountered and overcame various technical challenges, gaining invaluable insights into AI integration, network security protocols, and system design. Addressing issues such as latency in AI analysis, handling high traffic volumes, and managing API limitations reinforced the system's resilience and performance. These experiences underscored the importance of iterative development, performance tuning, and efficient resource management in building scalable and effective security solutions.

The successful implementation and positive outcomes of this project demonstrate the feasibility and benefits of utilizing AI for advanced threat detection. The AI-powered DNS security system showcased enhanced detection rates, reduced false positives, and maintained high network performance, all while providing a user-friendly interface for administrators. These achievements highlight the potential of AI to transform cybersecurity by offering more intelligent, adaptive, and efficient defense mechanisms.

This project has not only honed my technical skills in areas such as network security, AI integration, and full-stack development but has also emphasized the significance of comprehensive planning and adaptability when tackling complex cybersecurity challenges. The real-world applicability of the DNS security system underscores the transformative impact that AI can have on safeguarding network infrastructures against evolving threats.

Looking ahead, the groundwork established by this implementation paves the way for further advancements and enhancements. By continuing to refine AI models, expand system integrations, and incorporate advanced security features, the DNS security system can evolve to meet the dynamic landscape of cyber threats. This project has reinforced my commitment to exploring the intersection of AI and cybersecurity, and I am excited about the potential contributions such technologies can make to securing digital environments.

In conclusion, this project serves as a testament to the powerful synergy between artificial intelligence and network security. The AI-powered DNS threat detection system not only addresses current security needs but also establishes a scalable and adaptable framework capable of defending against future cyber challenges. I am confident that continued research and development in this area will yield even more sophisticated and effective security solutions, contributing to a safer and more secure digital world.



# Bibliography

- Ali, Tarek (2024). “Next-generation intrusion detection systems with LLMs: real-time anomaly detection, explainable AI, and adaptive data generation”. MA thesis. T. Ali.
- Bilge, L. et al. (2011). “EXPOSURE: Finding Malicious Domains Using Passive DNS Analysis”. In: *NDSS*.
- Eastlake 3rd, Donald (1999). *Domain name system security extensions*. Tech. rep.
- Kaminsky, D. (2008). “Black Ops 2008: It’s the End of the Cache As We Know It”. In: *Black Hat USA 2008*. Las Vegas, NV, USA.
- La O, Reynier Leyva, Carlos A Catania, and Tatiana Parlanti (2024). “LLMs for Domain Generation Algorithm Detection”. In: *arXiv preprint arXiv:2411.03307*.
- Liu, Xiao et al. (2024). “GPT understands, too”. In: *AI Open* 5, pp. 208–215.
- Mockapetris, Paul and Kevin J Dunlap (1988). “Development of the domain name system”. In: *Symposium proceedings on Communications architectures and protocols*, pp. 123–133.
- Sommer, R. and V. Paxson (2010). “Outside the Closed World: On Using Machine Learning for Network Intrusion Detection”. In: *IEEE Symposium on Security and Privacy*.
- Touvron, Hugo et al. (2023). “Llama: Open and efficient foundation language models”. In: *arXiv preprint arXiv:2302.13971*.
- Vaswani, A (2017). “Attention is all you need”. In: *Advances in Neural Information Processing Systems*.
- Zhauniarovich, Yury et al. (2018). “A survey on malicious domains detection through DNS data analysis”. In: *ACM Computing Surveys (CSUR)* 51.4, pp. 1–36.

# Appendices

## I. Code Samples

### A. DNS Attack Testing Script

File Path: Network-Analysis-using-LLMs/dns\_attack\_tester.py

#### Overview

In this section, I present the full code for the `dns_attack_tester.py` script, which is designed to simulate DNS-based attacks. This tool is crucial for testing the robustness and effectiveness of our DNS security system by mimicking real-world attack scenarios.

#### Code Listing

```
1 import socket
2 import asyncio
3 import random
4 import time
5 from scapy.all import IP, UDP, DNS, DNSQR, send, conf, ICMP, sr1
6 import argparse
7 from datetime import datetime
8 import netifaces as ni
9
10 class DNSAttackSimulator:
11     def __init__(self, interface="en0"): # Default to en0 for MacOS
12         self.interface = interface
13         conf.iface = interface
14
15         # Get your gateway IP
16         gws = ni gateways()
17         self.gateway = gws['default'][ni.AF_INET][0]
18
19         # Get your local IP
20         addrs = ni.ifaddresses(interface)
21         self.local_ip = addrs[ni.AF_INET][0]['addr']
22
23         print(f"Using interface: {interface}")
24         print(f"Local IP: {self.local_ip}")
25         print(f"Gateway IP: {self.gateway}")
26
27         self.domains = [
```



```

28     "google.com",
29     "a" * 30 + ".com", # Length pattern
30     "0123456789abcdef" * 2 + ".com", # Hex pattern
31     "malicious-looking-domain.top", # Suspicious TLD
32     "definitely-not-suspicious.cc", # Another suspicious TLD
33     f"{random.randint(1000000,9999999)}.xyz" # Random subdomain
34 ]
35
36 def is_blocked(self) -> bool:
37     """Check if the attack is being blocked by pinging the gateway."""
38     try:
39         pkt = IP(dst=self.gateway)/ICMP()
40         resp = sr1(pkt, timeout=1, verbose=0)
41         if resp is None:
42             return True
43         return False
44     except Exception as e:
45         print(f"Error checking block status: {e}")
46         return False
47
48 async def dns_flood_attack(self, duration: int = 10, rate: int = 100):
49     """Simulate a DNS flood attack"""
50     print(f"\n[+] Starting DNS flood attack for {duration} seconds at
51         ↳ {rate} qps")
52     start_time = time.time()
53     queries_sent = 0
54     while time.time() - start_time < duration:
55         try:
56             if self.is_blocked():
57                 print("Blocked")
58                 break
59
60             domain = random.choice(self.domains)
61             # Create DNS query packet
62             dns_packet = (
63                 IP(src=self.local_ip, dst=self.gateway)/
64                 UDP(sport=random.randint(49152, 65535), dport=53)/
65                 DNS(rd=1, qd=DNSQR(qname=domain))
66             )
67
68             send(dns_packet, verbose=0, iface=self.interface)
69             queries_sent += 1
70
71             if queries_sent % rate == 0:
72                 print(f"Attack progress: {queries_sent} queries sent")
73
74             # Add small delay to control rate
75             await asyncio.sleep(1/rate)
76
77     except Exception as e:

```

```

78         print(f"Error_during_attack:{e}")
79         break
80
81     return queries_sent
82
83 async def main():
84     parser = argparse.ArgumentParser(description='DNS_Attack_Simulation_Tool')
85     parser.add_argument('--interface', default='en0', help='Network_interface_
      ↪ to_use')
86     parser.add_argument('--duration', type=int, default=10, help='Attack_
      ↪ duration_in_seconds')
87     parser.add_argument('--rate', type=int, default=50, help='Queries_per_
      ↪ second')
88
89     args = parser.parse_args()
90
91     print(f"""
92     DNS Attack Simulation Starting
93     -----
94     Interface: {args.interface}
95     Duration: {args.duration} seconds
96     Rate: {args.rate} queries/second
97     """)
98
99     simulator = DNSAttackSimulator(interface=args.interface)
100    queries = await simulator.dns_flood_attack(args.duration, args.rate)
101    print(f"\nAttack_completed:{queries}_queries_sent")
102
103 if __name__ == "__main__":
104     # Install required package if not present
105     try:
106         import netifaces
107     except ImportError:
108         import subprocess
109         import sys
110         subprocess.check_call([sys.executable, "-m", "pip", "install",
      ↪ "netifaces"])
111         import netifaces
112
113     asyncio.run(main())

```

Listing 1: `dns_attack_tester.py` – *DNSAttackTestingScript*

## Detailed Explanation

Let me walk you through the key components and functionalities of the `dns_attack_tester.py` script:

### 1. Imports:

- `socket`, `asyncio`, `random`, `time`: Standard Python libraries for networking, asynchronous operations, random number generation, and time management.

- `scapy.all`: A powerful library for packet crafting and manipulation.
- `argparse`: For parsing command-line arguments.
- `datetime`: Handling date and time operations.
- `netifaces`: To retrieve network interface information.

## 2. DNSAttackSimulator Class:

- `__init__`: Initializes the simulator with the specified network interface (default is `en0` for MacOS). It retrieves the local IP and gateway IP using the `netifaces` library and defines a list of domains that mimic both legitimate and malicious patterns.
- `is_blocked`: Checks if the attack is being blocked by sending an ICMP ping to the gateway. If no response is received within a timeout period, it assumes that the traffic is being blocked.
- `dns_flood_attack`: Asynchronously sends DNS queries to simulate a flood attack. It selects random domains from the predefined list, crafts DNS query packets using Scapy, and sends them at the specified query rate. The attack runs for the defined duration unless it gets blocked.

## 3. main Function:

- Parses command-line arguments for network interface, attack duration, and query rate.
- Initializes the `DNSAttackSimulator` with the specified interface.
- Starts the DNS flood attack and reports the number of queries sent upon completion.

## 4. Execution Block:

- Ensures that the required `netifaces` package is installed. If not, it installs the package using `pip` and then proceeds to run the main function.

## Setup Instructions

To set up and run the `dns_attack_tester.py` script, follow these detailed steps:

### 1. Clone the Repository:

```
git clone https://github.com/error9098x/Network-Analysis-using-LLMs.git
cd Network-Analysis-using-LLMs
```

- ### 2. Install Dependencies:
- Ensure that you have Python 3.8 or higher installed. Install the required Python packages using `pip`:

```
pip install -r requirements.txt
```

3. **Configure the Script:** The script is configured to use the `en0` network interface by default, which is typical for MacOS. If you're using a different operating system or network interface, specify it using the `--interface` argument.
4. **Run the Attack Tester:** Execute the script with the desired parameters. For example, to run a 30-second attack at 100 queries per second on interface `eth0`:

```
sudo python dns_attack_tester.py --interface eth0 --duration 30 --rate 100
```

**Note:** Running this script requires administrative privileges (`sudo`) because it involves crafting and sending network packets.

5. **Understanding the Output:** The script will display the progress of the attack, indicating how many queries have been sent. If the attack is detected and blocked by the DNS security system, it will print "Blocked" and terminate the attack early.
6. **Monitoring the Impact:** Use the provided `index.html` dashboard to monitor how the DNS security system detects and responds to the simulated attack in real-time.

## B. Frontend Dashboard

**File Path:** Network-Analysis-using-LLMs/index.html

### Overview

Next, I present the full code for the `index.html` file, which serves as the web-based dashboard for real-time monitoring of DNS activities and security alerts. This dashboard provides an intuitive interface for administrators to visualize data, manage IP addresses, and respond to threats promptly.

### Code Listing

```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>DNS Monitor with AI Analysis</title>
7   <script src="https://cdn.tailwindcss.com"></script>
8   <script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
9   <link href="https://fonts.googleapis.com/icon?family=Material+Icons"
    ↪ rel="stylesheet">
10 </head>
11 <body class="bg-gray-900 text-white">
12   <div class="container mx-auto p-4">
13     <header class="mb-8">
14       <h1 class="text-3xl font-bold text-blue-500 flex items-center
    ↪ gap-2">

```

```

15         <span class="material-icons">dns</span>
16         DNS Monitor with AI Analysis
17     </h1>
18 </header>
19
20 <div class="grid grid-cols-1 lg:grid-cols-2 gap-6">
21     <!-- Domain Categories Chart -->
22     <div class="bg-gray-800 rounded-lg p-6">
23         <h2 class="text-xl font-semibold mb-4">Domain Categories</h2>
24         <div id="categoriesChart" class="h-[300px]"></div>
25     </div>
26
27     <!-- Risk Score Distribution -->
28     <div class="bg-gray-800 rounded-lg p-6">
29         <h2 class="text-xl font-semibold mb-4">Risk Score
30             ↪ Distribution</h2>
31         <div id="riskChart" class="h-[300px]"></div>
32     </div>
33
34     <!-- Recent Queries -->
35     <div class="bg-gray-800 rounded-lg p-6 col-span-1 lg:col-span-2">
36         <h2 class="text-xl font-semibold mb-4">Recent Queries</h2>
37         <div class="overflow-x-auto">
38             <table class="w-full">
39                 <thead>
40                     <tr class="text-left border-b border-gray-700">
41                         <th class="p-2">Time</th>
42                         <th class="p-2">Domain</th>
43                         <th class="p-2">Category</th>
44                         <th class="p-2">Risk Score</th>
45                         <th class="p-2">Analysis</th>
46                     </tr>
47                 </thead>
48                 <tbody id="recentQueries"></tbody>
49             </table>
50         </div>
51     </div>
52
53     <!-- Security Alerts -->
54     <div class="bg-gray-800 rounded-lg p-6">
55         <h2 class="text-xl font-semibold mb-4">Security Alerts</h2>
56         <div id="alertsList" class="space-y-2"></div>
57     </div>
58
59     <!-- IP Management -->
60     <div class="bg-gray-800 rounded-lg p-6">
61         <h2 class="text-xl font-semibold mb-4">IP Management</h2>
62         <div class="flex gap-2 mb-4">
63             <input type="text" id="ipInput"
64                 class="flex-1 bg-gray-700 rounded px-3 py-2"
65                 placeholder="Enter IP address">

```

```

65         <button onclick="blockIP()"
66             class="bg-red-500 hover:bg-red-600 px-4 py-2
67                 rounded">
68             Block IP
69         </button>
70     </div>
71     <div id="blockedIPs" class="space-y-2"></div>
72 </div>
73 </div>
74
75 <script>
76     let ws = new WebSocket('ws://localhost:8000/ws');
77
78     function updateCharts(data) {
79         // Update categories chart
80         const categories = Object.entries(data.categories);
81         Plotly.newPlot('categoriesChart', [{
82             labels: categories.map(([cat, _]) => cat),
83             values: categories.map([_, count]) => count),
84             type: 'pie',
85             hole: 0.4
86         ], {
87             paper_bgcolor: 'rgba(0,0,0,0)',
88             plot_bgcolor: 'rgba(0,0,0,0)',
89             font: { color: '#fff' },
90             showlegend: true
91         });
92
93         // Update risk distribution chart
94         const riskScores = data.recent_queries.map(q => q.risk_score);
95         Plotly.newPlot('riskChart', [{
96             x: riskScores,
97             type: 'histogram',
98             marker: { color: '#4CAF50' }
99         ], {
100             paper_bgcolor: 'rgba(0,0,0,0)',
101             plot_bgcolor: 'rgba(0,0,0,0)',
102             font: { color: '#fff' },
103             xaxis: { title: 'Risk Score' },
104             yaxis: { title: 'Count' }
105         });
106     }
107
108     function getRiskColor(score) {
109         if (score < 0.3) return 'text-green-500';
110         if (score < 0.7) return 'text-yellow-500';
111         return 'text-red-500';
112     }
113
114     function updateRecentQueries(queries) {

```

```

115     const tbody = document.querySelector('#recentQueries');
116     tbody.innerHTML = queries.map(q => '
117         <tr class="border-b border-gray-700">
118             <td class="p-2">${q.timestamp}</td>
119             <td class="p-2">${q.domain}</td>
120             <td class="p-2">${q.category}</td>
121             <td class="p-2
122                 ↳ ${getRiskColor(q.risk_score)}">${q.risk_score.toFixed(2)}</td>
123             <td class="p-2 text-sm">${q.analysis}</td>
124         </tr>
125     ').join('');
126
127     function updateAlerts(alerts) {
128         const alertsList = document.getElementById('alertsList');
129         alertsList.innerHTML = alerts.map(alert => '
130             <div class="bg-slate-700 border-l-4 border-red-500 p-3 rounded
131                 ↳ flex items-center justify-between">
132                 <div class="flex items-center gap-2">
133                     <span class="material-icons text-red-500">warning</span>
134                     <span>${alert}</span>
135                 </div>
136                 <span class="text-sm text-slate-400">${new
137                     ↳ Date().toLocaleTimeString()}</span>
138             </div>
139         ').join('');
140
141     function updateBlockedIPs(blockedIPs) {
142         const blockedIPsDiv = document.getElementById('blockedIPs');
143         blockedIPsDiv.innerHTML = blockedIPs.map(ip => '
144             <div class="bg-slate-700 p-3 rounded flex items-center
145                 ↳ justify-between">
146                 <div class="flex items-center gap-2">
147                     <span class="material-icons text-red-500">block</span>
148                     ${ip}
149                 </div>
150                 <button onclick="unblockIP('${ip}')"
151                     class="bg-red-500 hover:bg-red-600 px-3 py-1 rounded
152                         ↳ text-sm flex items-center gap-1">
153                     <span class="material-icons text-sm">remove</span>
154                     Unblock
155                 </button>
156             </div>
157         ').join('');
158
159     async function blockIP() {
160         const ip = document.getElementById('ipInput').value;
161         try {

```



```
160         const response = await fetch('/block_ip/${ip}', { method:
161             ↪ 'POST' });
162         if (!response.ok) throw new Error('Failed to block IP');
163         document.getElementById('ipInput').value = '';
164     } catch (error) {
165         console.error('Error blocking IP:', error);
166     }
167 }
168
169 async function unblockIP(ip) {
170     try {
171         const response = await
172             ↪ fetch('http://localhost:8000/unblock_ip/${ip}', {
173             method: 'DELETE',
174             headers: {
175                 'Content-Type': 'application/json'
176             }
177         });
178
179         if (!response.ok) {
180             throw new Error('Failed to unblock IP:
181                 ↪ ${response.statusText}');
182         }
183
184         const result = await response.json();
185         console.log('Unblock result:', result);
186
187         // Refresh the WebSocket connection to get updated data
188         if (ws.readyState === WebSocket.OPEN) {
189             ws.send('refresh');
190         }
191     } catch (error) {
192         console.error('Error unblocking IP:', error);
193         alert('Failed to unblock IP: ' + error.message);
194     }
195 }
196
197 ws.onmessage = function(event) {
198     const data = JSON.parse(event.data);
199     updateCharts(data);
200     updateAlerts(data.security_alerts);
201     updateRecentQueries(data.recent_queries);
202     updateBlockedIPs(data.blocked_ips);
203 };
204
205 ws.onclose = function() {
206     setTimeout(() => {
207         ws = new WebSocket('ws://localhost:8000/ws');
208     }, 1000);
209 };
210 </script>
```



```
208 </body>
209 </html>
```

Listing 2: index.html - Frontend Dashboard

## Detailed Explanation

Let me break down the key components and functionalities of the `index.html` dashboard:

### 1. Layout and Styling:

- Utilizes **Tailwind CSS** for responsive and modern styling.
- The dashboard is divided into sections such as Domain Categories, Risk Score Distribution, Recent Queries, Security Alerts, and IP Management for organized data presentation.

### 2. Charts and Visualization:

- **Plotly.js** is used to create interactive and visually appealing charts.
- **Domain Categories** pie chart provides a quick overview of the types of domains being queried.
- **Risk Score Distribution** histogram helps in understanding the spread of risk scores assigned to DNS queries.

### 3. Real-Time Data Handling:

- Establishes a **WebSocket** connection to receive live updates from the backend.
- Functions like `updateCharts`, `updateRecentQueries`, `updateAlerts`, and `updateBlockedIP` dynamically update the dashboard based on incoming data.

### 4. IP Management:

- Provides input fields and buttons to allow administrators to block or unblock IP addresses.
- Interacts with backend API endpoints to perform these actions, ensuring that unauthorized or malicious IPs are effectively managed.

### 5. Error Handling and Reconnection:

- Implements reconnection logic in case the **WebSocket** connection drops, ensuring uninterrupted data flow.
- Provides feedback in the console for any errors encountered during IP blocking/unblocking operations.

## Setup Instructions

To set up and run the `index.html` dashboard, follow these detailed steps:

### 1. Clone the Repository:

```
git clone https://github.com/error9098x/Network-Analysis-using-LLMs.git
cd Network-Analysis-using-LLMs
```

- 2. Install Dependencies:** Ensure that you have an active backend server running to handle WebSocket connections and API requests. The frontend relies on this backend to fetch and display real-time data.
- 3. Configure the Backend Server:** The frontend dashboard communicates with the backend server via WebSockets and RESTful APIs. Ensure that the backend server is configured to accept connections on `localhost:8000` or adjust the WebSocket URL in the JavaScript code accordingly.
- 4. Run the Backend Server:** Start the backend server using the provided `dns.py` script:

```
sudo python dns.py
```

**Note:** Running `dns.py` requires administrative privileges for packet capturing and sending.

- 5. Open the Dashboard:** Simply open the `index.html` file in your preferred web browser:

```
open index.html
```

**Alternatively:** Right-click the `index.html` file and choose your browser's "Open with" option.

### 6. Interacting with the Dashboard:

- **Real-Time Monitoring:** Monitor DNS activities through various charts and tables.
- **Security Alerts:** Observe security alerts for immediate notifications about potential threats.
- **IP Management:** Manage IP addresses by entering an IP in the input field and clicking "Block IP" or "Unblock IP".

## Code Functionality Breakdown

- **Layout and Styling:**

- Utilizes **Tailwind CSS** for a clean and responsive design.
- The dashboard layout is structured using CSS grid for optimal organization of different sections.

- **Charts and Visualization:**

- **Plotly.js** is leveraged to create dynamic and interactive charts, enhancing data visualization and user engagement.
- The **Domain Categories** pie chart provides a visual breakdown of different domain categories being queried.
- The **Risk Score Distribution** histogram displays the distribution of risk scores assigned to DNS queries, aiding in quick risk assessment.

- **Recent Queries and Alerts:**

- The **Recent Queries** table lists the latest DNS queries along with their categories, risk scores, and analysis details.
- The **Security Alerts** section displays real-time alerts for detected threats, enabling prompt response and action.

- **IP Management:**

- Provides functionalities to block or unblock specific IP addresses, empowering administrators to control network access effectively.
- The interface includes an input field for entering IPs and buttons to execute blocking or unblocking actions.

- **WebSocket Connection:**

- Establishes a WebSocket connection to `ws://localhost:8000/ws` to receive real-time data updates from the backend.
- Ensures that the dashboard remains up-to-date with the latest DNS activities and security alerts.

- **Chart Updates:**

- The `updateCharts` function renders and updates the Domain Categories pie chart and the Risk Score histogram using the data received from the backend.

- **Recent Queries and Alerts:**

- `updateRecentQueries`: Populates the Recent Queries table with incoming data.
- `updateAlerts`: Displays security alerts in real-time, allowing for prompt awareness and response.

- **IP Management Functions:**

- **blockIP**: Handles the blocking of IP addresses by sending a POST request to the backend API.
- **unblockIP**: Manages the unblocking of IP addresses by sending a DELETE request to the backend API.

- **Error Handling and Reconnection:**

- Implements reconnection logic to re-establish the WebSocket connection if it drops, ensuring uninterrupted data flow.
- Provides console logs for any errors encountered during IP blocking/unblocking operations.

## Usage Tips

- **Real-Time Monitoring:** Keep an eye on the **Recent Queries** table to track ongoing DNS activities and quickly identify suspicious behavior.
- **Security Alerts:** Pay attention to the **Security Alerts** section for immediate notifications about potential threats. Investigate and respond to these alerts promptly to maintain network security.
- **IP Management:** Regularly review the **IP Management** section to block IPs that exhibit malicious behavior. Unblock IPs when necessary to restore legitimate access.
- **Chart Interactions:** Hover over chart elements to view detailed information and gain deeper insights into the data presented.
- **Customization:** Feel free to modify the frontend code to add more features or adjust the existing ones to better suit your monitoring needs.

## Conclusion

The `index.html` frontend dashboard is a pivotal component of our DNS security system, providing a user-friendly interface for real-time monitoring and management. By leveraging modern web technologies and interactive libraries, the dashboard offers comprehensive insights into DNS activities, enabling administrators to maintain a secure and efficient network environment.

## List of Publications/Online References

- OpenAI. (2023). *GPT-4 Technical Report*. Retrieved from <https://openai.com/research/gpt-4>
- Meta AI. (2023). *LLaMA: Open and Efficient Foundation Language Models*. Retrieved from <https://ai.facebook.com/blog/large-language-models-llama>
- Anderson, C. (2019). *DNS Security: Defending the Domain Name System*. *Cybersecurity Journal*, 12(3), 45-52.
- Smith, J. (2021). *Real-Time Threat Detection with AI*. Retrieved from <https://www.cybersecurityinsider.com/real-time-ai-threat-detection>
- Zebin, T., Rezvy, S., Luo, Y. (2022). *An Explainable AI-Based Intrusion Detection System for DNS over HTTPS (DoH) Attacks*. *IEEE Transactions on Information Forensics and Security*, 17, 2339-2349. Retrieved from <https://ieeexplore.ieee.org/document/9796558>
- Qin, M. (2024). *Multi-Task DNS Security Analysis via High-Order Heterogeneous Graph Embedding*. *arXiv preprint arXiv:2401.07410*. Retrieved from <https://arxiv.org/abs/2401.07410>
- Moubayed, A., Injadat, M., Shami, A. (2020). *Optimized Random Forest Model for Botnet Detection Based on DNS Queries*. *arXiv preprint arXiv:2012.11326*. Retrieved from <https://arxiv.org/abs/2012.11326>
- Ahmed, J. (2022). *Monitoring Security of Enterprise Hosts via DNS Data Analysis*. *arXiv preprint arXiv:2205.08968*. Retrieved from <https://arxiv.org/abs/2205.08968>
- Hamidouche, M., Demissie, B. F., Cherif, B. (2024). *Real-time Threat Detection Strategies for Resource-constrained Devices*. *arXiv preprint arXiv:2403.15078*. Retrieved from <https://arxiv.org/abs/2403.15078>
- Palo Alto Networks. (n.d.). *What Is the Role of AI in Threat Detection?*. Retrieved from <https://www.paloaltonetworks.com/cyberpedia/ai-in-threat-detection>