PRIVACY-PRESERVING REAL-TIME HOME AUTOMATION UTILIZING MQTT PROTOCOL AND SENSOR ANOMALY DETECTION WITH GENAI INTEGRATION

A Thesis Submitted in Partial
Fulfillment for the Requirement of the Degree of

Bachelor of Science
in
Computer Science and Engineering

by

Anowar Hossain
ID: 221071051

Shihab Sarker
ID: 202071004

Under the Supervision of

Tahsin Alam
Lecturer
Department of Computer Science and Engineering

to the

Department of Computer Science and Engineering
Shanto-Mariam University of Creative Technology

February, 2026

ACKNOWLEDGEMENT

This thesis has been submitted to the Department of Computer Science and Engineering of Shanto-Mariam University of Creative Technology (SMUCT), Dhaka, Bangladesh, in partial fulfillment of the requirements for the degree of B.Sc. in Computer Science and Engineering. The thesis title is “PRIVACY-PRESERVING REAL-TIME HOME AUTOMATION UTILIZING MQTT PROTOCOL AND SENSOR ANOMALY DETECTION WITH GENAI INTEGRATION”.

First and foremost, we offer our sincere gratitude to our thesis supervisor, **Tahsin Alam**, Lecturer, Department of Computer Science and Engineering, for his continuous support, motivation, and immense knowledge. His valuable guidance and dedicated supervision greatly contributed to the successful completion of this research.

We also express our sincere appreciation to the Head of the Department, **Faisal Imran**, Head & Associate Professor, for his academic leadership and for providing the necessary facilities and institutional support to conduct this research work.

We are extremely grateful to our Thesis Moderator, **Md. Tousif Hasan Lavlu**, Lecturer, Department of Computer Science & Engineering, for his thorough review, insightful feedback, and continuous encouragement. His constructive evaluation was instrumental in refining this thesis.

Our heartfelt thanks are extended to the External Examiner, **Sakhor Das Opi**, Lecturer, Department of Computer Science & Engineering, for his valuable time and constructive suggestions, which have significantly improved the quality and presentation of this work.

Finally, we express our deepest gratitude to our parents and family members for their unconditional love, constant encouragement, and moral support throughout our academic journey. Without them, this achievement would not have been possible.

Anowar Hossain
ID: 221071051

Shihab Sarker
ID: 202071004

February, 2026
SMUCT, Dhaka, Bangladesh

CERTIFICATE

This is to certify that the thesis entitled "PRIVACY-PRESERVING REAL-TIME HOME AUTOMATION UTILIZING MQTT PROTOCOL AND SENSOR ANOMALY DETECTION WITH GENAI INTEGRATION" by **Anowar Hossain** (ID: 221071051) and **Shihab Sarker** (ID: 202071004) has been carried out under my direct supervision. To the best of my knowledge, this thesis is an original work and has not been submitted anywhere for any degree or diploma.

Thesis Supervisor:
....

Tahsin Alam
Lecturer
Department of Computer Science and Engineering
Shanto-Mariam University of Creative Technology

CERTIFICATE

This is to certify that the thesis entitled “PRIVACY-PRESERVING REAL-TIME HOME AUTOMATION UTILIZING MQTT PROTOCOL AND SENSOR ANOMALY DETECTION WITH GENAI INTEGRATION” by **Anowar Hossain** (ID: 221071051) and **Shihab Sarker** (ID: 202071004) has been carried out under the direct supervision of **Tahsin Alam**. This thesis has been prepared according to the guidelines of the Department of Computer Science & Engineering / Department of Computer Science & Information Technology of Shanto-Mariam University of Creative Technology.

Head of the Department
....

Faisal Imran
Head & Associate Professor
Department of Computer Science & Engineering and
Department of CSIT
Shanto-Mariam University of Creative Technology

CERTIFICATE

This is to certify that the thesis entitled "PRIVACY-PRESERVING REAL-TIME HOME AUTOMATION UTILIZING MQTT PROTOCOL AND SENSOR ANOMALY DETECTION WITH GENAI INTEGRATION" submitted by **Anowar Hossain** (ID: 221071051) and **Shihab Sarker** (ID: 202071004) in partial fulfillment of the requirements for the degree of B.Sc. in Computer Science and Engineering has been examined and moderated by me.

I have carefully reviewed the thesis defense and documentation. I found it to be a comprehensive and original piece of research work. The thesis successfully meets the academic standards and requirements set by the Department of Computer Science & Engineering / Department of Computer Science & Information Technology of Shanto-Mariam University of Creative Technology.

Thesis Moderator
....

Md. Tousif Hasan Lavlu
Lecturer
Department of Computer Science & Engineering
Shanto-Mariam University of Creative Technology

CERTIFICATE

This is to certify that the thesis entitled “PRIVACY-PRESERVING REAL-TIME HOME AUTOMATION UTILIZING MQTT PROTOCOL AND SENSOR ANOMALY DETECTION WITH GENAI INTEGRATION” submitted by **Anowar Hossain** (ID: 221071051) and **Shihab Sarker** (ID: 202071004) in partial fulfillment of the requirements for the degree of B.Sc. in Computer Science and Engineering has been examined by me.

I have carefully reviewed the thesis and found that it is an original piece of research work. The thesis meets the academic standards and requirements set by the Department of Computer Science & Engineering / Department of Computer Science & Information Technology of Shanto-Mariam University of Creative Technology.

External Examiner
....

Sakhor Das Opi
Lecturer
Department of Computer Science & Engineering
Shanto-Mariam University of Creative Technology

DECLARATION

We hereby declare that the thesis entitled “PRIVACY-PRESERVING REAL-TIME HOME AUTOMATION UTILIZING MQTT PROTOCOL AND SENSOR ANOMALY DETECTION WITH GENAI INTEGRATION” is the result of our own independent research work carried out under the supervision of **Tahsin Alam**, Lecturer, Department of Computer Science and Engineering, Shanto-Mariam University of Creative Technology.

This thesis has not been submitted, either in whole or in part, to any other university or institution for the award of any degree, diploma, or other qualification. All sources of information used in this research have been duly acknowledged through proper references and citations.

We take full responsibility for the authenticity and accuracy of the work presented in this thesis.

....

Anowar Hossain
B.Sc. in Computer Science and Engineering
ID: 221071051
Department of Computer Science and Engineering
Shanto-Mariam University of Creative Technology

....

Shihab Sarker
B.Sc. in Computer Science and Engineering
ID: 202071004
Department of Computer Science and Engineering
Shanto-Mariam University of Creative Technology

Abstract

The rapid proliferation of Internet of Things (IoT) devices in domestic environments has introduced unprecedented convenience, yet simultaneously created critical vulnerabilities regarding user data privacy and data sovereignty. Conventional smart home architectures heavily rely on cloud-centric processing, routinely exposing sensitive, unencrypted environmental telemetry to third-party servers. To address this fundamental security gap, this research presents "IoTShield", a fully localized, privacy-by-design home monitoring framework. The system operates on a hybrid edge-computing architecture, utilizing ESP32 microcontrollers for localized sensor polling and a consumer-grade edge server (equipped with a 13th Generation Intel Core i5 processor) for localized intelligence. To ensure robust data obfuscation without compromising system utility, IoTShield employs a resilient dual-layer privacy pipeline. First, a Differential Privacy mechanism injects calibrated Gaussian noise ($\epsilon = 0.5$) at the hardware level to mask routine behavioral patterns. Second, the perturbed payloads undergo RSA-2048 asymmetric encryption with OAEP padding prior to transmission via the lightweight MQTT protocol.

For intelligent anomaly detection, the framework strictly avoids external APIs by integrating a localized Large Language Model (Llama 3.2: 1B) managed via the Ollama inference engine. The AI dynamically analyzes multivariate sensor data to generate context-aware, natural language safety directives. Experimental evaluation demonstrated the system's ability to seamlessly process continuous telemetry streams, maintaining a highly responsive average end-to-end latency of 420 milliseconds. The localized AI successfully categorized 1,568 distinct environmental anomalies across four severity tiers (Low, Medium, High, and Critical), automatically escalating severe threats via an integrated SMTP email relay. This study conclusively confirms that combining offline Generative AI with edge-level cryptographic privacy mechanisms can deliver a highly responsive, intelligent, and secure smart home environment without ever exposing sensitive domestic data to the public internet.

Keywords: IoT Security, Differential Privacy, Edge Computing, Generative AI, Llama 3.2, RSA-2048 Encryption, MQTT Protocol, Anomaly Detection, Smart Home Automation.

Contents

List of Figures

List of Tables

List of Abbreviations

AES	Advanced Encryption Standard
AI	Artificial Intelligence
API	Application Programming Interface
CPU	Central Processing Unit
CSE	Computer Science and Engineering
DoS	Denial of Service
ESP32	Espressif Systems 32-bit Microcontroller
GenAI	Generative Artificial Intelligence
GPIO	General Purpose Input/Output
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IoT	Internet of Things
IP	Internet Protocol
JSON	JavaScript Object Notation
JWT	JSON Web Token
LED	Light Emitting Diode
LLM	Large Language Model
LDR	Light Dependent Resistor
MQTT	Message Queuing Telemetry Transport
PIR	Passive Infrared Sensor
QoS	Quality of Service
RAM	Random Access Memory
REST	Representational State Transfer
RSA	Rivest–Shamir–Adleman (Encryption Algorithm)
SMTP	Simple Mail Transfer Protocol
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UI	User Interface
URL	Uniform Resource Locator
Wi-Fi	Wireless Fidelity

List of Symbols

M	Plaintext Message
C	Ciphertext (Encrypted Message)
K_{pub}	Public Key
K_{priv}	Private Key
e	Public Exponent (RSA)
d	Private Exponent (RSA)
n	Modulus (RSA)
μ	Mean Value (Gaussian Distribution)
σ	Standard Deviation
ϵ	Privacy Budget / Noise Parameter
$N(\mu, \sigma^2)$	Gaussian (Normal) Distribution
T	Temperature Sensor Reading
H	Humidity Sensor Reading
G	Gas Sensor Value
L_{lat}	System Latency
$P(x)$	Probability of an Event
V_{in}	Input Voltage
R	Resistance
t	Time / Timestamp
Δ	Difference / Change in Value
θ	Threshold Value for Anomaly Detection
Hz	Hertz (Frequency Unit)
dB	Decibel (Signal Strength Unit)
bps	Bits Per Second

Chapter 1

Introduction

1.1 Background and Motivation

The rapid evolution of the Internet of Things (IoT) has fundamentally transformed modern living spaces into intelligent, interconnected environments. Smart home automation systems now routinely deploy diverse networks of sensors and actuators to monitor environmental conditions, manage energy consumption, and significantly enhance user convenience. To facilitate this seamless interaction, lightweight communication protocols, particularly the Message Queuing Telemetry Transport (MQTT) protocol, have become the industry standard. MQTT enables efficient [1], low-latency data transmission even across highly constrained networks and low-power microcontrollers, making it ideal for real-time residential applications.

Despite the tremendous advantages of these smart home ecosystems, their widespread adoption introduces critical vulnerabilities surrounding data privacy and security. Continuous environmental monitoring generates a massive influx of granular data, capturing intimate details about household occupancy, daily routines, and behavioral habits. In conventional architectures, this highly sensitive data is frequently transmitted in its raw form to centralized cloud servers for processing. This heavy reliance on third-party cloud infrastructure exposes users to severe risks, including data breaches, unauthorized surveillance, and corporate data profiling. While basic transport-layer security is sometimes implemented, true end-to-end data confidentiality at the application layer remains a significant challenge in residential deployments.

Beyond privacy concerns, traditional smart home systems are largely constrained by rudimentary, rule-based logic. Most commercial setups rely on static, predefined thresholds to trigger alerts—for example, activating a notification only if a gas or temperature sensor strictly exceeds a hardcoded limit. This rigid approach lacks contextual awareness. It frequently results in a high volume of false positives, struggles to identify complex anomalies that span multiple sensor types, and provides users with generic, numerical warnings rather than clear, actionable insights into the actual problem.

Motivated by these profound limitations, there is a pressing need for a paradigm shift toward secure, intelligent, and decentralized home automation. The integration of Generative Artificial Intelligence - GenAI [2] specifically localized Large Language Models (LLMs) [3], presents a revolutionary opportunity to move beyond simple threshold alerts to context-aware, human-readable anomaly detection. This research is driven by the ambition to engineer a holistic framework that marries the efficiency of MQTT with rigorous privacy-preserving mechanisms—such as dual-layer Gaussian noise and RSA encryption—processed entirely at the network edge. By doing so, the proposed system aims to deliver highly responsive, intelligent monitoring while guaranteeing absolute data sovereignty for the user.

1.2 Problem Statement

The integration of Internet of Things (IoT) devices into residential environments has significantly improved building automation, safety, and energy management. However, the standard architecture of these systems fundamentally compromises user privacy and data security. Most commercial smart home setups rely heavily on centralized, cloud-based processing, where raw, granular sensor data—such as room occupancy, temperature fluctuations, and lighting usage—is continuously transmitted over the internet. This continuous data stream creates a high-risk attack surface [4]. If intercepted or analyzed by unauthorized entities, this information can be used to deduce sensitive domestic activities, leaving users vulnerable to surveillance, corporate data profiling, and malicious physical intrusions [5].

Furthermore, the communication protocols utilized by resource-constrained IoT devices, while lightweight and efficient, often lack robust application-layer security. For instance, while the MQTT protocol is highly optimized for low-bandwidth environments, standard implementations frequently leave the data payload entirely readable by the central broker. Even when transport-layer security (TLS) is applied, the centralized broker remains a single point of failure that can expose historical and real-time data if compromised. There is a distinct lack of mechanisms that protect the structural privacy of the data—such as cryptographic encryption or differential privacy—before it even leaves the physical edge device.

In addition to these severe privacy vulnerabilities, current home automation systems suffer from severely limited analytical capabilities. They typically depend on rigid, rule-based logic to trigger alerts, such as activating an alarm only when a gas or temperature sensor crosses a strict, hardcoded numerical limit. This rudimentary approach fails to account for environmental context, frequently leading to a high rate of false positives and a lack of actionable information. When complex, multi-sensor anomalies occur, users are presented with generic numerical warnings rather than a clear, intelligent explanation of the event. While advanced machine learning models can offer more sophisticated detection, they are usually too computationally expensive to deploy locally on edge-based microcontrollers, forcing a reliance back onto the insecure cloud [6].

Therefore, the primary problem addressed in this research is the lack of a secure, edge-centric framework capable of performing real-time home automation and intelligent anomaly detection without compromising data sovereignty. Specifically, the challenge lies in securing lightweight MQTT communications through application-layer encryption and noise-based privacy mechanisms, while simultaneously deploying a Generative AI model locally to analyze complex sensor patterns and generate human-readable, context-aware alerts within a decentralized architecture.

1.3 Research Gap

Despite the extensive body of literature on smart home automation and Internet of Things (IoT) security, a significant gap remains at the intersection of lightweight communication, robust data privacy, and intelligent anomaly detection. A comprehensive review of existing methodologies reveals a persistent dichotomy: systems are typically either highly secure but computationally heavy, or lightweight but dangerously vulnerable. While the MQTT protocol is widely celebrated for its low latency and efficiency in constrained environments, prevailing implementations primarily rely on transport-layer security (TLS). This approach leaves the data payload unencrypted at the broker level, creating a critical vulnerability if the central server is compromised. Furthermore, the application of structural privacy mechanisms—such as injecting differential noise at the sensor node before transmission—is rarely explored in real-time, lightweight residential architectures.

In the domain of anomaly detection, current smart home deployments predominantly utilize either static, rule-based thresholds or traditional machine learning models (e.g., Support Vector Machines and Random Forests). While threshold-based systems are efficient, they lack the contextual awareness necessary to identify sophisticated, multi-sensor anomalies, leading to high false-positive rates. Conversely, while traditional machine learning models can detect deviations, they function as "black boxes," providing numerical anomaly scores without any human-readable explanation or actionable context. When an alert is triggered, the user is left to interpret raw data rather than receiving a clear, reasoned diagnosis of the potential hazard.

Recently, Generative Artificial Intelligence (GenAI) and Large Language Models (LLMs) have demonstrated exceptional capabilities in contextual reasoning and data interpretation. However, their application in IoT environments is heavily biased toward cloud-centric architectures due to their immense computational requirements. The deployment of localized, edge-optimized LLMs to process raw sensor telemetry and generate intelligent, real-time alerts without relying on third-party cloud APIs remains significantly under-researched. Existing studies that do explore Al-driven IoT security mostly focus on high-level network traffic analysis rather than granular, domestic environmental sensor data.

Consequently, the fundamental research gap lies in the absence of a holistic, decentralized smart home framework that unifies these disparate elements. There is a distinct lack of systems that successfully combine application-layer cryptographic payload encryption, privacy-preserving data obfuscation, and edge-deployed Generative AI to provide secure [7], real-time, and explainable anomaly detection within a lightweight MQTT ecosystem.

1.4 Research Objectives

The primary aim of this research is to design, implement, and evaluate a decentralized, privacy-preserving smart home automation system capable of intelligent, real-time anomaly detection. To achieve this overarching goal, the research is guided by the following specific objectives:

- **To design a lightweight, edge-centric IoT architecture:** Develop a responsive smart home network utilizing ESP32 microcontrollers, a Raspberry Pi edge gateway, and the MQTT protocol to ensure low-latency communication without heavy reliance on external cloud infrastructure [8].
- **To implement a dual-layer privacy and security pipeline:** Engineer a robust data protection mechanism that applies structural obfuscation via dual-layer Gaussian noise (differential privacy) and secures the MQTT payload using application-layer RSA-2048 encryption before data transmission.
- **To integrate localized Generative AI for anomaly detection:** Deploy a lightweight Large Language Model (Llama 3.2:1B via Ollama) to autonomously analyze multi-sensor telemetry, replacing rigid threshold-based logic with context-aware, human-readable alert generation.
- **To construct an interactive, full-stack monitoring platform:** Build a comprehensive backend and web dashboard (utilizing Django, Tailwind CSS, and Chart.js) to manage devices, decrypt sensor data, and visualize real-time AI-generated insights.
- **To empirically evaluate system performance:** Validate the proposed IoTShield framework in a real-world deployment by analyzing key metrics, including end-to-end latency, encryption overhead, anomaly detection accuracy, and the overall privacy-utility trade-off.

1.5 Contributions of the Thesis

This thesis makes several significant contributions to the fields of Internet of Things security, edge computing, and smart home automation. By addressing the critical vulnerabilities of cloud-dependent architectures, the proposed IoTShield system introduces a novel paradigm for domestic environmental monitoring.

The core contributions of this research are summarized as follows:

- **A Novel Edge-Based GenAI Integration:** This research demonstrates the feasibility and immense value of deploying localized Generative AI at the network edge. By utilizing a 1-billion parameter LLM to process sensor data locally, the system successfully transitions anomaly detection from binary, threshold-based triggers to intelligent, context-aware reasoning, providing users with actionable and highly accurate alerts.

- **Application-Layer MQTT Security Architecture:** Unlike standard IoT deployments that rely solely on transport-layer security, this thesis contributes a fully functional RSA-2048 encryption scheme specifically optimized for MQTT payloads. This ensures that even if the central message broker is compromised, the granular sensor telemetry remains strictly confidential.

- **Practical Implementation of Differential Privacy in IoT:** The research introduces a dual-layer Gaussian noise injection mechanism directly at the sensor node. This contributes a practical methodology for masking exact domestic routines and behavioral patterns while preserving enough data utility for the AI model to accurately detect critical anomalies (e.g., fires or gas leaks).

- **Empirical Validation of a Production-Ready Framework:** The thesis provides comprehensive, real-world validation of the proposed architecture. Through the deployment of physical hardware and simulated nodes generating over 13,000 sensor readings and validating over 1,500 distinct alerts, the study proves that a highly secure, AI-driven IoT system can operate with an average end-to-end latency of less than two seconds.

1.6 Scope of the Thesis

The scope of this research is specifically bounded to the design, implementation, and evaluation of a privacy-preserving, AI-driven monitoring framework tailored for residential smart home environments. The project focuses strictly on secure environmental data collection, anomalous event detection, and real-time alert generation, rather than large-scale industrial IoT (IIoT) deployments or complex physical robotic actuation.

In terms of hardware and physical deployment, the scope encompasses the development of a fully operational real-world sensor node. This node utilizes a physical ESP32 microcontroller integrated with six distinct environmental sensors (temperature, humidity, gas, flame, motion, and light) to validate real-world data acquisition. To rigorously evaluate the system's network handling, multi-device architecture, and edge processing capabilities without deploying extensive hardware, the scope also incorporates simulated IoT nodes. These consist of a simulated secondary ESP32 hub and a simulated Raspberry Pi edge gateway, which additionally provides system performance metrics such as CPU and memory usage.

From a software and security perspective, the research is limited to securing the MQTT communication protocol at the application layer. This involves the implementation of a custom RSA-2048 encryption pipeline for payload confidentiality and a dual-layer Gaussian noise mechanism to ensure differential privacy. The thesis does not attempt to invent new foundational cryptographic algorithms or replace standard transport-layer security (TLS); instead, it focuses on practical, edge-level data obfuscation to prevent broker-level data compromises.

Finally, the intelligence and anomaly detection scope is centered entirely on edge-optimized Generative AI. The system integrates a locally hosted, 1-billion parameter Large Language Model (Llama 3.2 via Ollama) to process decrypted sensor telemetry and generate context-aware, human-readable alerts. The research deliberately excludes the use of proprietary, cloud-based AI APIs to strictly maintain data sovereignty and adhere to decentralized edge-computing principles. Furthermore, the development of a responsive backend and web dashboard (utilizing Django, SQLite, and Tailwind CSS) is included within the scope to demonstrate end-to-end data persistence, real-time visualization, and practical alert management.

1.7 Thesis Organization

The remainder of this thesis is logically structured into five subsequent chapters, each detailing a critical phase of the research, development, and evaluation of the proposed IoTShield framework.

Chapter 2: Related Work provides a comprehensive review of the existing literature surrounding smart home automation. It examines current MQTT-based architectures, privacy-preserving IoT systems, and the application of cryptographic mechanisms in constrained environments. Furthermore, it explores traditional anomaly detection techniques alongside the emerging role of Generative AI in real-time monitoring, ultimately highlighting the crucial limitations and research gaps that this thesis addresses.

Chapter 3: Proposed System outlines the theoretical and structural design of the IoT-Shield architecture. It details the holistic system overview, defining the roles of the physical ESP32 sensors and the simulated edge gateway. This chapter extensively covers the MQTT communication model, the privacy-preserving data pipeline incorporating dual-layer Gaussian noise and RSA-2048 encryption, and the localized AI-based anomaly detection framework driven by Llama 3.2.

Chapter 4: Implementation translates the proposed architecture into a functional, real-world application. It documents the exact hardware setup, the software development environment (including Django, Tailwind CSS, and Chart.js), and the configuration of the Mosquitto MQTT broker. Additionally, it details the practical coding implementations of the differential privacy mechanisms, the application-layer encryption, the local LLM deployment, and the relational database schema utilized for data persistence.

Chapter 5: Results and Discussion presents a rigorous empirical evaluation of the operational system. Based on an extensive dataset of over 13,000 sensor readings and more than 1,500 validated alerts, this chapter analyzes system latency, cryptographic processing overhead, and the accuracy of the AI-driven anomaly detection. It also provides a critical discussion on the privacy-utility trade-off and compares the proposed framework's performance against traditional baseline models.

Chapter 6: Conclusion and Future Work synthesizes the key findings of the research, summarizing the successful integration of privacy-preserving mechanisms with edge-based Generative AI. It candidly discusses the limitations encountered during the study and proposes strategic directions for future research to further enhance secure, intelligent residential automation.

Chapter 2

Related Work

2.1 MQTT-Based Smart Home Architectures

The Message Queuing Telemetry Transport (MQTT) protocol has rapidly emerged as the definitive communication standard for Internet of Things deployments. Its lightweight, publish-subscribe architecture is uniquely optimized for resource-constrained environments, making it highly effective for residential smart home systems. Numerous foundational architectures have successfully utilized micro-controllers, such as the ESP-8266 and ESP-32, paired with Raspberry Pi edge gateways to manage real-time sensor data. By establishing the Raspberry Pi as a central broker, these decentralized configurations ensure seamless data aggregation and appliance control while maintaining exceptionally low network overhead.

A primary driver for the widespread adoption of MQTT [9] is its remarkable efficiency in bandwidth utilization and energy conservation when compared to traditional client-server protocols like HTTP. Empirical studies within smart home environments have demonstrated that MQTT can achieve substantial reductions in data transmission size, yielding proportional energy savings for low-power sensor nodes. Furthermore, the protocol's structured topic hierarchy and adaptable Quality of Service (QoS) levels provide a robust framework for building scalable, fault-tolerant automation systems that guarantee reliable message delivery under varying network conditions.

Beyond simple telemetry, MQTT serves as a critical enabler for edge-based intelligence [10]. By integrating the protocol with local edge computing paradigms, researchers have significantly reduced cloud dependency, allowing systems to maintain high responsiveness even during internet outages. Deployments combining MQTT brokers with local data processing have achieved end-to-end latencies measured in milliseconds, which is absolutely critical for time-sensitive applications such as fire, gas leak, and intrusion detection. Additionally, the protocol's asynchronous nature allows it to easily orchestrate complex automated workflows across diverse household subsystems.

However, while these existing architectures unequivocally validate MQTT's operational efficiency and real-time streaming capabilities, they frequently expose a critical weakness in data security. In its standard implementation, MQTT primarily relies on transport-layer security and often transmits the actual sensor payloads in plaintext. This architectural oversight assumes a perfectly secure local network and leaves the central broker as a highly vulnerable single point of failure. If the broker is compromised, sensitive domestic telemetry is entirely exposed to eavesdropping and manipulation. This inherent vulnerability under-scores an urgent need to augment standard MQTT architectures with robust, application-layer cryptographic and privacy-preserving mechanisms.

2.2 Privacy-Preserving IoT Systems

The exponential growth of smart home deployments has precipitated a corresponding escalation in privacy concerns. Residential IoT ecosystems are inherently intimate; the continuous telemetry generated by motion detectors, temperature sensors, and smart lighting constructs a highly granular behavioral profile of the occupants. In traditional [11],, centralized architectures, this raw data is routinely transmitted to third-party cloud servers for storage and analysis. This paradigm fundamentally compromises user privacy, exposing sensitive domestic activities to potential data breaches, unauthorized corporate profiling, and malicious interception. Consequently, the research focus within the IoT domain has increasingly shifted toward developing robust, privacy-preserving frameworks [12] that can secure data without crippling the operational utility of the smart home.

A significant portion of existing literature addresses IoT privacy through the lens of data minimization and structural obfuscation. One of the most prominent methodologies is Differential Privacy (DP). By systematically injecting calibrated statistical noise—such as Laplacian or Gaussian noise—into the sensor readings before transmission, DP ensures that individual data points cannot be reverse-engineered to reveal specific user activities. Several recent studies have successfully applied DP to smart meter data and healthcare [13] IoT, demonstrating its mathematical rigor in protecting individual privacy. However, applying Differential Privacy in real-time residential automation presents a complex challenge: the privacy-utility trade-off. If the injected noise is too aggressive, the data becomes useless for time-sensitive anomaly detection, potentially masking critical events like fires or gas leaks. Conversely, if the noise is too minimal, the privacy guarantee is mathematically void [14].

To mitigate the reliance on vulnerable cloud infrastructures, researchers have also extensively explored edge computing as a primary privacy-preserving mechanism. By migrating data processing and anomaly detection to localized edge gateways—such as Raspberry Pi clusters or localized servers—systems can analyze telemetry directly within the home network. This decentralized approach drastically reduces the volume of sensitive data transmitted over the public internet. While edge computing fundamentally enhances data sovereignty, it does not completely eliminate internal threats. If the local network or the edge broker itself is compromised by an intruder, any unencrypted raw data stored or streaming across the local architecture remains fully exposed.

Furthermore, advanced distributed learning techniques, particularly Federated Learning

(FL), have gained traction as a privacy-preserving alternative to centralized machine learning. In FL setups, IoT devices train anomaly detection models locally and only share cryptographic weight updates with a central server, ensuring raw sensor data never leaves the device. While highly secure, Federated Learning introduces immense computational and memory overheads that are often entirely impractical for resource-constrained microcontrollers like the ESP32.

Ultimately, the literature highlights a persistent fragmentation in privacy-preserving IoT systems. Existing solutions tend to isolate their approaches—focusing either on network-level decentralization, mathematically complex noise injection, or heavy distributed learning—rather than combining them into a cohesive, lightweight architecture. There remains a definitive need for hybrid frameworks that can seamlessly integrate structural data obfuscation at the sensor node with secure, decentralized edge processing, ensuring both strict data privacy and high-fidelity anomaly detection.

2.3 Cryptographic Mechanisms in IoT Security

Cryptography forms the fundamental backbone of data confidentiality and integrity within Internet of Things networks. In standard smart home architectures [15],, Transport Layer Security (TLS) is predominantly employed to secure the communication channels between edge devices and central servers. While TLS effectively prevents man-in-the-middle attacks and packet sniffing during transit, its implementation in highly constrained environments presents notable challenges, including significant computational overhead and increased handshake latency [16],. More critically, TLS only encrypts the transmission tunnel itself. Once the sensor data reaches the centralized MQTT broker or cloud gateway, it is entirely decrypted and processed in plaintext. This architectural norm creates a severe vulnerability; any successful breach of the central broker instantly exposes the entire history of sensitive domestic telemetry to the attacker.

To address the inherent limitations of tunnel-only security, recent literature has increasingly focused on application-layer cryptographic mechanisms. By encrypting the actual data payload before transmission, the information remains secure even if the intermediary broker is compromised. Symmetric encryption algorithms [17], such as the Advanced Encryption Standard (AES), are frequently proposed for this task due to their low computational requirements and fast execution speeds, making them highly suitable for low-power microcontrollers like the ESP32. However, symmetric cryptography fundamentally suffers from complex key distribution and management challenges in distributed networks. If a single pre-shared key is extracted from a physically accessible edge device, the confidentiality of the entire network can be invalidated, making it difficult to scale securely across dynamic environments with multiple interconnected sensors.

Consequently, asymmetric cryptographic frameworks, including Rivest–Shamir–Adleman (RSA) and Elliptic Curve Cryptography (ECC) algorithms, have gained substantial traction for IoT deployments. Asymmetric infrastructures resolve the key distribution dilemma by utilizing a public-key architecture. Edge devices can encrypt their telemetry using a widely distributed public key, ensuring that only the secure backend system holding the corresponding private key can decrypt the payload. Historically, algorithms like RSA-2048 were considered too resource-intensive for microcontrollers, often causing prohibitive delays.

However, with the advent of more powerful edge hardware and optimized cryptographic libraries, the deployment of robust asymmetric encryption on devices like the ESP32 has become a practical reality. Despite these advancements, a significant gap remains in the literature regarding the practical, real-world integration of asymmetric application-layer encryption directly within MQTT payloads. The majority of existing studies propose theoretical security models or run limited simulations, rather than demonstrating end-to-end, hardware-validated implementations that successfully balance rigorous RSA payload encryption with the low-latency requirements of real-time home automation.

2.4 Anomaly Detection Techniques in IoT

Anomaly detection is a foundational component of intelligent smart home monitoring, serving as the primary mechanism for identifying critical events such as gas leaks, fires, or unauthorized physical intrusions. Historically, the most ubiquitous approach to identifying these hazards has been rule-based or threshold-based logic [18]. In these deterministic systems, an anomaly is flagged strictly when a sensor reading surpasses a predefined numerical boundary. While this method is highly computationally efficient and easily deployed on low-power microcontrollers like the ESP32, it is inherently rigid. Static thresholds fail to account for benign environmental variations, such as a localized temperature spike caused by cooking, frequently resulting in a high volume of false positives. Furthermore, threshold logic struggles to evaluate multivariate relationships, missing complex anomalies that only become apparent when multiple sensor states are analyzed simultaneously.

To overcome the inflexibility of static rules, the academic focus shifted toward integrating traditional machine learning (ML) classifiers directly into the IoT ecosystem. Algorithms including Support Vector Machines (SVM), Random Forests, and K-Nearest Neighbors have been extensively researched and deployed at the network edge. By training these models on historical environmental telemetry, systems can learn the baseline "normal" behavior of a household and identify statistical deviations. For example, edge-based SVM [19] implementations have successfully aggregated data from temperature, smoke, and gas sensors to classify hazardous events with remarkably low latency. These decentralized ML architectures significantly improve detection accuracy over basic thresholding while keeping processing localized to edge gateways like the Raspberry Pi.

More recently, deep learning techniques, particularly Autoencoders and recurrent neural networks like Long Short-Term Memory (LSTM) models, have been introduced to handle complex, non-linear time-series data in IoT networks. While these deep learning models offer superior accuracy in recognizing subtle anomalous patterns [20], they introduce immense computational and memory overheads. Deploying such heavily parameterized models on resource-constrained edge devices typically requires aggressive model quantization or forces the system to offload inference tasks back to centralized cloud servers. This reliance on the cloud immediately reintroduces the severe privacy vulnerabilities and latency issues that edge computing aims to eliminate.

Despite the varying degrees of accuracy across threshold logic, traditional ML, and deep learning, all of these existing methodologies share a critical, fundamental limitation: they function entirely as "black boxes." When a hazard is identified, these systems output binary classifications (e.g., normal versus anomalous) or arbitrary numerical anomaly scores [21]. They completely lack the capacity to provide contextual reasoning, root cause analysis, or actionable recommendations. Consequently, when an alarm is triggered, the end-user is presented with a generic warning rather than a clear, human-readable explanation of the specific domestic event. This absence of interpretability severely diminishes the practical utility of the system and creates a pressing need for a more intelligent, context-aware approach to anomaly interpretation.

2.5 Generative AI for Real-Time Monitoring

The recent and rapid advancements in Generative Artificial Intelligence (GenAI), specifically the evolution of Large Language Models (LLMs), have introduced an entirely new frontier for Internet of Things automation. Unlike traditional machine learning classifiers that strictly output binary decisions or raw numerical anomaly scores, Generative AI [22] possesses the unique capability to comprehend complex, multivariate data streams and generate natural language outputs. In the context of smart home monitoring [23], this represents a profound paradigm shift from simple alerting to intelligent, context-aware reasoning. By processing aggregated sensor telemetry—such as a simultaneous spike in temperature, gas concentration, and motion—an LLM can synthesize the environmental context to deduce the likely root cause of an event. This allows the system to provide users with clear, human-readable, and highly actionable insights rather than cryptic numerical warnings.

Despite its immense potential, the integration of Generative AI within IoT ecosystems is currently dominated by cloud-centric architectures. The vast majority of contemporary research and commercial applications rely on massive, proprietary models accessed via third-party Application Programming Interfaces (APIs). While these cloud-hosted models offer unparalleled reasoning capabilities, their deployment in residential automation is fundamentally flawed regarding user privacy and system reliability. Transmitting granular, continuous domestic telemetry to an external cloud server for AI inference directly exposes highly sensitive behavioral data to potential interception [24], data breaches, and corporate profiling. Furthermore, this heavy reliance on continuous internet connectivity introduces unacceptable latency for time-critical emergencies, such as fire outbreaks or gas leaks, where immediate, localized actuation is required.

To mitigate these severe privacy and latency vulnerabilities, academic focus is beginning to pivot toward Edge AI. The recent release of highly optimized, parameter-efficient LLMs has made it viable to perform complex generative inference directly on localized hardware, such as edge gateways or dedicated on-premise servers. By deploying lightweight models locally, systems can retain absolute data sovereignty; the sensitive environmental telemetry never leaves the physical boundaries of the home network. This decentralized approach guarantees that the generation of intelligent alerts and automated responses is completely insulated from external network outages and third-party data collection policies.

However, the practical application of localized Generative AI for real-time IoT anomaly detection remains significantly under-researched. While deploying lightweight models at the edge solves the immediate cloud-dependency issue, integrating these demanding models seamlessly with low-latency communication protocols like MQTT presents a major engineering challenge. Furthermore, the existing literature has yet to adequately demonstrate how a localized LLM can effectively interpret granular sensor data that has been concurrently secured by robust cryptographic protocols and structural privacy mechanisms. The absence of a cohesive framework that unifies edge-based Generative AI with rigorous, application-layer data protection constitutes a critical gap in current smart home security paradigms.

2.6 Limitations of Existing Approaches

A comprehensive review of the contemporary literature reveals a heavily fragmented landscape in smart home security and intelligent monitoring [25]. While substantial progress has been made in individual domains—such as optimizing MQTT for constrained devices, applying traditional machine learning at the edge, and utilizing mathematical privacy models like Differential Privacy—there remains a distinct lack of unified, holistic frameworks. Most existing architectures [26] force a detrimental compromise: systems that are highly secure tend to incur unacceptable latency for real-time applications, while lightweight, highly responsive systems frequently expose raw, unencrypted telemetry to central brokers or third-party cloud infrastructure.

Furthermore, the paradigm of anomaly detection in residential IoT is currently trapped between two extremes. Traditional threshold and standard machine learning models offer edge-compatible efficiency but operate as unexplainable "black boxes," providing users with little to no actionable context during an emergency. Conversely, advanced Generative AI models capable of contextual reasoning are predominantly cloud-bound, introducing severe data sovereignty risks and rendering the smart home dependent on continuous internet connectivity [27].

To synthesize these critical vulnerabilities, Table 2.1 provides a comparative summary of the prevalent methodologies in recent literature, explicitly detailing their approaches, underlying technologies, advantages, and fundamental limitations.

Table 2.1: Summary of Existing Literature and Comparative Analysis

Year	**Method / Approach**	**Devices / Technologies**	**Pros**	**Limitations**
2022	Centralized IoT with Transport-Layer Security	ESP8266, MQTT, Cloud-based Broker, TLS/SSL	Low deployment cost; highly scalable; lightweight transmission.	Payload remains in plaintext at the broker; high vulnerability to central server compromise; cloud dependency.
2023	Edge-Based Machine Learning for Anomaly Detection	Raspberry Pi, Support Vector Machines (SVM), MQTT	Eliminates cloud reliance for basic detection; faster response times.	"Black box" alerts with no human-readable context; lacks structural privacy or data payload encryption.
2024	Differential Privacy in Continuous IoT Telemetry	Smart Meters, Laplacian Noise, Cloud Aggregation	Strong mathematical guarantees against user profiling.	Aggressive noise masks critical anomalies (e.g., fires); high latency; still reliant on cloud processing.
2025	Cloud-Based LLMs for Contextual Smart Environments	Proprietary API (e.g., OpenAI), HTTP, Smart Hubs	Highly accurate anomaly interpretation; clear, human-readable explanations.	Severe privacy risks (raw data leaves the home); high latency; fails during internet network outages.
Proposed	**Edge GenAI with Application-Layer Privacy**	**ESP32, Local Llama 3.2, RSA-2048, Dual Gaussian Noise**	**Absolute data sovereignty; context-aware reasoning; encrypted broker payload.**	**Requires optimized edge hardware; slightly higher computational overhead at the sensor node.**

As demonstrated in the comparative analysis above, the fundamental research gap lies in the absence of a decentralized system capable of executing application-layer cryptographic payload encryption and privacy-preserving data obfuscation, while simultaneously deploying a localized Generative AI model to provide explainable anomaly detection. Addressing these specific and interconnected limitations forms the direct motivation for the architecture proposed in the subsequent chapter.

Chapter 3

Proposed System

This chapter extensively details the architectural blueprint and operational mechanisms of the proposed IoTShield framework. It systematically breaks down the integration of edge hardware, the application-layer security protocols, the privacy-preserving data pipeline, and the localized deployment of Generative AI utilized for intelligent anomaly detection.

3.1 Overall System Overview

The IoTShield architecture is fundamentally designed as a decentralized, privacy-centric monitoring ecosystem for residential environments [28]. The core motivation behind this design is to capture granular environmental telemetry, cryptographically secure it directly at the source, and analyze it using advanced artificial intelligence without ever transmitting sensitive data to a third-party cloud. To accomplish this, the system is structured across three cohesive operational layers: the Edge Data Acquisition Layer, the Secure Communication Broker, and the Intelligent AI Backend.

At the Edge Data Acquisition Layer, the system is driven by a physical ESP32 microcontroller, which functions as the primary hardware sensing node. This physical node is wired to a diverse array of environmental sensors capable of tracking real-time temperature, humidity, gas levels, flame presence, motion, and ambient light. Alongside this physical hardware, the architecture integrates a simulated ESP32 hub to represent multi-room deployment and a simulated Raspberry Pi to act as the central edge gateway. This hybrid physical-simulated approach rigorously tests the network's scalability. Most importantly, all telemetry generated at this layer undergoes a strict, two-step privacy transformation before broadcast. First, dual-layer Gaussian noise is injected into the raw readings to provide differential privacy, masking exact behavioral routines. Immediately after, the obfuscated payload is secured using RSA-2048 encryption, ensuring absolute data confidentiality.

The Secure Communication Broker layer handles the efficient routing of this protected data. Operating on the lightweight MQTT protocol, the encrypted sensor payloads are published to designated topics on a Mosquitto broker, which logically resides on the simulated Raspberry Pi edge gateway. A critical architectural advantage of IoTShield is that the MQTT broker functions strictly as a blind relay. Because the telemetry is encrypted at the application layer by the edge nodes, the broker only processes unreadable ciphertext. Consequently, even if an intruder compromises the local network or the broker itself, the underlying domestic data remains mathematically protected and completely inaccessible.

Finally, the Intelligent AI Backend processes the encrypted stream to deliver actionable insights. Built upon a robust Django framework, this backend securely decrypts the incoming MQTT payloads using the system's localized private RSA key [29]. Once decrypted, the raw telemetry is evaluated against dynamic, multi-sensor threshold logic to instantly categorize the environmental severity (e.g., Low, Medium, High, or Critical). When a high-severity anomaly is identified, the backend routes the complex sensor data matrix directly into a locally hosted, 1-billion parameter Large Language Model (Llama 3.2 via Ollama). The LLM autonomously analyzes the event context and generates a precise, human-readable alert explaining the probable cause of the hazard. All telemetry and AI-generated alerts are subsequently archived in a SQLite relational database and rendered on an interactive web dashboard for seamless user management.

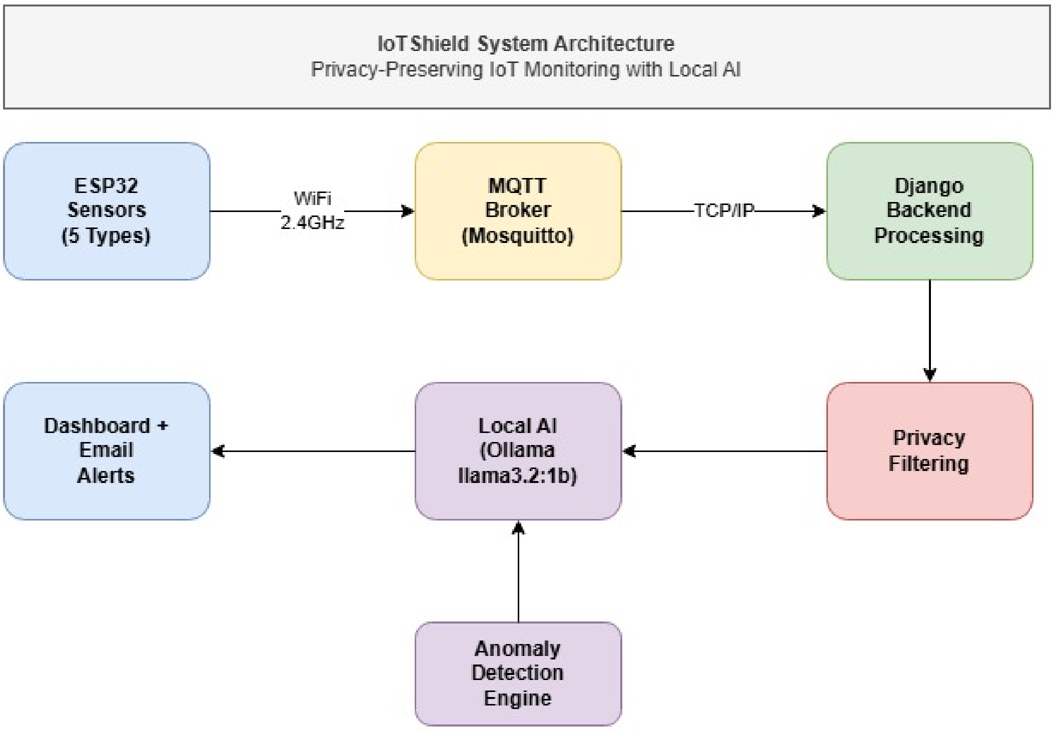

IoTShield System Architecture
Privacy-Preserving IoT Monitoring with Local AI
ESP32 Sensors (5 Types)
WiFi 2.4GHz
MQTT Broker (Mosquitto)
TCP/IP
Django Backend Processing
Privacy Filtering
Local AI (Ollama llama3.2:1b)
Dashboard + Email Alerts
Anomaly Detection Engine

Figure 3.1: High-level architectural overview of the proposed IoTShield system, illustrating the secure data flow from the edge sensor nodes to the localized Generative AI analysis framework.

3.2 System Architecture Design

The architectural design of the IoTShield framework is structured around a localized, decentralized network topology operating entirely within a private Local Area Network (LAN). To guarantee data sovereignty and eliminate reliance on external cloud processing, all core communications, data routing, and artificial intelligence inferences are confined within the 192.168.1.0/24 subnet. The network is orchestrated by a standard WiFi Router acting as the default gateway (192.168.1.1), which manages DHCP assignments, Network Address Translation (NAT), and internal firewall policies to insulate the smart home ecosystem from the public Wide Area Network (WAN).

Operating at the network edge is the ESP32 Smart Sensor Hub, which connects to the local router via a 2.4GHz WiFi interface. Assigned to a dedicated local IP address (192.168.1.150), this edge node is responsible for continuously polling the connected hardware—specifically the DHT11, MQ-2, Flame, LDR, and PIR sensors. Once the telemetry is acquired, the ESP32 processes the data through the local privacy pipeline, packages it into a secure JSON payload, and transmits it via the MQTT protocol to the central server.

The core intelligence and data routing of the system are hosted on the Central Server, acting as the edge gateway. To ensure maximum stability and minimize latency, this server is connected to the router via a hardwired Ethernet connection and assigned a static IP address (192.168.1.100). The server concurrently hosts three critical, containerized services: the Mosquitto MQTT Broker on Port 1883, the Django Web Server on Port 8000, and the Ollama AI Inference engine on Port 11434, which hosts the localized Llama 3.2 model.

Finally, the User Interface tier allows end-users to access the monitoring dashboard and manage alerts using any standard web browser on a mobile device or laptop. By navigating to the internal address (`http://192.168.1.100:8000`), users can interact with the system securely over the local WiFi connection without routing traffic over the public internet.

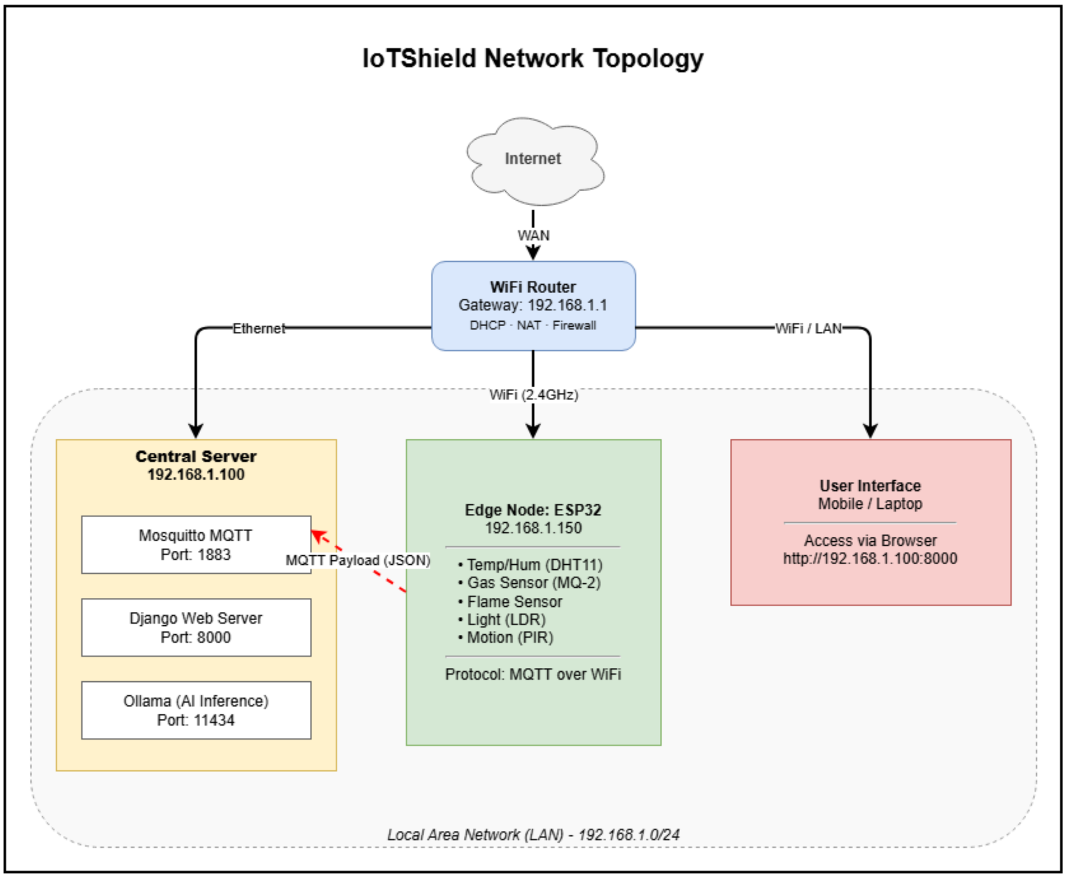


Figure 3.2: Detailed network topology of the IoTShield framework, illustrating the local IP subnet structure, port assignments, and the strict isolation of edge processing from the public internet.

3.3 Hardware Architecture (ESP32 and Sensors)

The physical foundation of the IoTShield framework is built upon a highly responsive, resource-optimized hardware architecture designed specifically for edge environments. At the core of the Edge Sensing Tier is the ESP32 microcontroller, specifically the ESP32-WROOM-32 module [30]. The ESP32 was deliberately selected over lower-tier microcontrollers due to its dual-core 240 MHz Tensilica processor, built-in 2.4 GHz Wi-Fi capabilities, and sufficient SRAM to handle demanding application-layer tasks. Unlike standard IoT edge nodes that merely forward raw electrical signals, the ESP32 in this architecture is tasked with actively polling six distinct sensors, applying dual-layer Gaussian noise matrices, executing RSA-2048 cryptographic encryption, and managing asynchronous MQTT publishing—all in real-time.

To capture a comprehensive profile of the domestic environment, the ESP32 is physically interfaced with a diverse array of five distinct sensor modules, collectively gathering six unique environmental metrics. Ambient climate monitoring is handled by a DHT11 digital sensor, which provides continuous temperature and relative humidity readings. Hazardous environmental anomalies are monitored by an MQ-2 analog gas sensor, calibrated to detect combustible gases and smoke concentrations, working in tandem with an infrared (IR) Flame sensor that triggers upon detecting specific light wavelengths associated with open fires. Finally, spatial and security metrics are captured using an HC-SR501 Passive Infrared (PIR) motion sensor to detect physical occupancy, alongside a Light Dependent Resistor (LDR) to measure ambient room illumination.

By aggregating these diverse metrics onto a single ESP32 node, the hardware architecture provides the localized AI model with a rich, multivariate dataset [31]. This allows the system to cross-reference multiple data points (e.g., detecting a simultaneous spike in temperature, gas, and a positive flame reading) to accurately diagnose complex hazards. Table 3.1 provides a detailed technical specification of the hardware components utilized in the physical edge node deployment.

Table 3.1: Hardware Components and Sensor Specifications

Component	**Type**	**Operating Range / Details**	**Role in System**
ESP32-WROOM-32	Microcontroller	240 MHz Dual-Core, 3.3V Logic, 802.11 b/g/n Wi-Fi	Primary edge gateway; executes RSA encryption, differential privacy, and MQTT routing.
DHT11	Digital Climate Sensor	Temp: 0–50°C, Humidity: 20–90% RH	Monitors ambient room temperature and relative humidity levels.
MQ-2	Analog Gas Sensor	300–10,000 ppm (LPG, Methane, Smoke)	Identifies hazardous gas leaks and potential pre-combustion smoke.
IR Flame Sensor	Infrared Receiver	Wavelengths: 760 nm to 1100 nm	Provides immediate binary detection of open flames or fire outbreaks.
HC-SR501 (PIR)	Digital Motion Sensor	Range: 7m, Angle: 120°	Detects human occupancy or unauthorized physical intrusion.
LDR (Photoresistor)	Analog Light Sensor	1 MΩ (Dark) to 10 kΩ (Light)	Monitors ambient illumination to provide contextual environmental data.

3.4 Software Architecture

The software architecture of the IoTShield system is engineered as a modular, high-concurrency framework designed to facilitate real-time environmental monitoring while maintaining strict local data sovereignty. The architecture follows a decoupled, service-oriented paradigm where each component—ranging from edge firmware to the backend inference engine—operates independently yet communicates through standardized interfaces. This design choice ensures that the system can handle high-frequency telemetry from multiple nodes without blocking critical security operations or AI reasoning tasks.

At the network's periphery, the Firmware Layer is developed using the Arduino framework in C++, specifically optimized for the dual-core architecture of the ESP32. This layer manages the low-level hardware abstraction for the connected sensor array and executes the initial privacy-preserving pipeline. By utilizing the second core of the ESP32 for cryptographic operations, the firmware ensures that the RSA-2048 encryption and Gaussian noise injection do not interfere with the primary sensor polling and MQTT publishing loops, thus maintaining a consistent data acquisition rate.

The Message Broker Layer serves as the foundational communication backbone, utilizing the Eclipse Mosquitto implementation of the MQTT protocol. This layer is configured to handle asynchronous messaging with a Quality of Service (QoS) Level 1, ensuring that critical telemetry packets are acknowledged by the backend. The broker functions as a stateless relay, where the encrypted sensor payloads are routed through a hierarchical topic structure. This decoupling allows the backend processing services to subscribe to specific data streams without maintaining a persistent direct connection to each individual hardware node.

The Backend and Application Layer is constructed using the Django web framework, which orchestrates the core logic of the IoTShield system. This layer manages the secure decryption of payloads, the execution of multivariate threshold logic for immediate anomaly detection, and the persistence of telemetry within a relational SQLite database. Furthermore, the backend integrates with the Ollama inference engine via a local API to facilitate the generation of context-aware alerts. By hosting the Llama 3.2 model locally, the software architecture ensures that the transition from raw environmental data to human-readable insights occurs entirely within the edge environment, fulfilling the primary research objective of localized and privacy-preserving monitoring.

3.5 MQTT Communication Model

The communication backbone of the IoTShield system is built upon the Message Queuing Telemetry Transport (MQTT) protocol, specifically selected for its lightweight overhead and suitability for resource-constrained edge devices like the ESP32. The system utilizes an asynchronous publish-subscribe paradigm, where the Mosquitto broker serves as a central hub for routing secure telemetry between the edge sensing nodes and the Django backend. To ensure reliability in a residential network environment, the system is configured with a Quality of Service (QoS) Level 1, guaranteeing that critical sensor updates and hazard alerts are delivered to the subscriber at least once, supported by a 60-second keep-alive interval to monitor node connectivity [32].

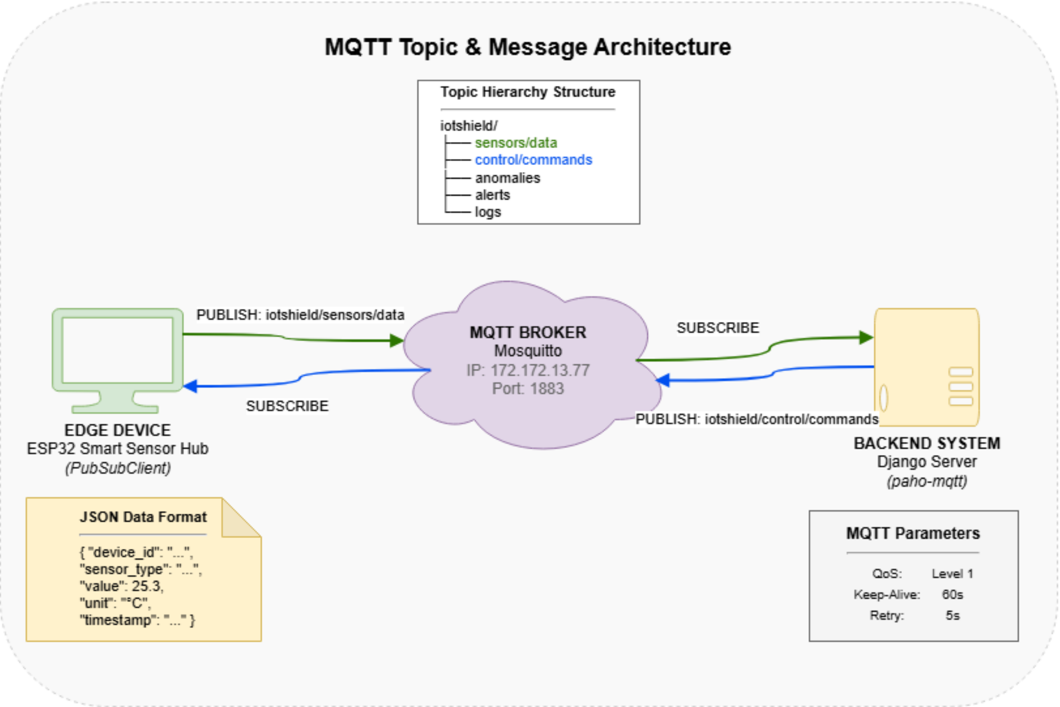
The communication architecture is governed by a strict hierarchical topic structure, which allows for efficient data filtering and multi-device management. The ESP32 hardware nodes publish encrypted environmental data to specific sub-topics, while the backend system subscribes to these streams to trigger the decryption and anomaly detection pipelines. Conversely, the backend can publish commands to the control topic, allowing for remote device management. This decoupled approach ensures that the sensing nodes do not need to maintain a direct awareness of the backend's state, significantly reducing the computational burden on the ESP32. Table 3.2 details the topic hierarchy and the corresponding payload characteristics utilized in the IoTShield framework.

Table 3.2: MQTT Topic Structure and Payload Formats

Topic Name	Direction	QoS	Payload Description
iotshield/sensors/data	Publish	1	RSA-encrypted JSON containing multi-sensor telemetry (Temp, Hum, Gas, etc.).
iotshield/alerts	Publish	1	Encrypted alerts containing severity levels and AI-generated suggestions.
iotshield/control/commands	Subscribe	1	Incoming instructions for device configuration or manual sensor polling.
iotshield/logs	Publish	0	Plaintext diagnostic messages for system health and connectivity monitoring.

Figure 3.3 illustrates the end-to-end communication flow, highlighting the interaction be-

tween the Edge Device (PubSubClient), the Mosquitto Broker, and the Backend System (paho-mqtt), while detailing the JSON data format utilized for secure message exchange.



```
iotshield/
├── sensors/data
├── control/commands
├── anomalies
├── alerts
└── logs
```

```
{ "device_id": "...",
"sensor_type": "...",
"value": 25.3,
"unit": "°C",
"timestamp": "..." }
```

QoS:	Level 1
Keep-Alive:	60s
Retry:	5s

Figure 3.3: MQTT communication architecture illustrating the hierarchical topic structure, message flow between edge nodes and the backend, and key protocol parameters.

3.6 Privacy-Preserving Data Pipeline

A core contribution of the IoTShield framework is its dual-layer privacy-preserving data pipeline, designed to protect domestic telemetry against both external interception and internal behavioral profiling. Unlike traditional systems that transmit raw data, this architecture implements a localized security sequence on the ESP32 node prior to network transmission.

3.6.1 Differential Privacy with Dual Gaussian Noise

The first stage of the pipeline addresses the risk of behavioral pattern mining. Even when encrypted, high-frequency sensor values can be analyzed to reveal sensitive occupant routines. To mitigate this, the system injects a calibrated level of Gaussian noise into the raw sensor telemetry. Utilizing a privacy budget of $\epsilon = 0.5$, the ESP32 adds a noise displacement that masks the exact environmental value while preserving the statistical utility required for anomaly detection. This ensures that the published telemetry represents a ”noisy” version of the environment (e.g., 25.523°C instead of 25.0°C), providing a strong mathematical guaranty of differential privacy.

3.6.2 RSA-2048 Encryption Mechanism

Following structural obfuscation, the data enters the cryptographic stage to ensure end-to-end confidentiality. The noisy payload is encrypted using the RSA-2048 algorithm, employing Optimal Asymmetric Encryption Padding (OAEP) and the SHA-256 hashing function. By utilizing an asymmetric public-key infrastructure, the ”IoTShield” system secures the payload such that only the backend server, possessing the corresponding private key, can access the plaintext contents. This transformation renders the telemetry into unreadable ciphertext before it is packaged into a JSON format for MQTT publication. Consequently, the data remains mathematically protected while transiting the broker and during storage in the SQLite database.

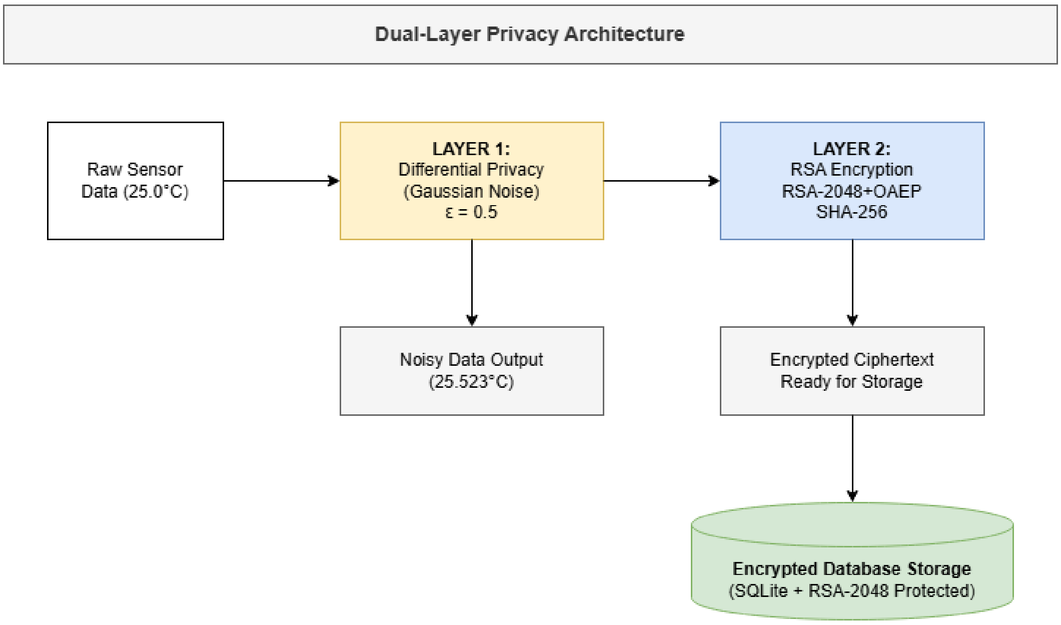

Dual-Layer Privacy Architecture
Raw Sensor Data (25.0°C)
LAYER 1: Differential Privacy (Gaussian Noise) ε = 0.5
LAYER 2: RSA Encryption RSA-2048+OAEP SHA-256
Noisy Data Output (25.523°C)
Encrypted Ciphertext Ready for Storage
Encrypted Database Storage (SQLite + RSA-2048 Protected)

Figure 3.4: Dual-layer privacy architecture of the IoTShield system, illustrating the sequential transformation of raw sensor data through Gaussian noise injection and RSA-2048 encryption.

3.7 AI-Based Anomaly Detection Framework

The final tier of the IoTShield system is the intelligent inference framework, which transforms raw environmental telemetry into actionable, human-readable insights. This framework is designed to operate entirely at the edge, utilizing a two-stage detection pipeline: a high-speed threshold-based screening layer and a context-aware Generative AI reasoning layer [34]. By hosting the inference engine locally, the system ensures that sensitive domestic data is never exposed to third-party cloud providers, fulfilling the primary privacy objectives of the research.

3.7.1 Threshold-Based Detection Logic

Before the data is passed to the AI model, it undergoes an initial evaluation via a multivariate threshold-based detection engine. This layer serves as a computational filter to differentiate between routine environmental fluctuations and potential hazards. Each incoming sensor reading is compared against a set of predefined boundaries that categorize the severity of the event. While a single sensor breach (e.g., a ”Medium” gas reading) triggers an alert, the system is designed to recognize compound anomalies, such as simultaneous spikes in temperature and gas, which escalate the priority level. Table 3.3 defines the specific threshold parameters and the corresponding severity classifications utilized by the IoTShield backend [35].

Table 3.3: Sensor Threshold Parameters and Severity Classification

Sensor Type	**Normal Range**	**LOW**	**MEDIUM**	**HIGH/CRITICAL**
Temperature (°C)	18.0 – 28.0	28.1 – 35.0	35.1 – 45.0	> 45.0
Humidity (%)	30.0 – 60.0	60.1 – 75.0	75.1 – 85.0	> 85.0
Gas (MQ-2)	< 0.20	0.21 – 0.40	0.41 – 0.60	> 0.60
Flame (IR)	0 (Stable)	–	–	1 (Flame Detected)
Motion (PIR)	0 (No Motion)	1 (Detected)	–	–
Light (LDR)	200 – 800 lux	< 200	–	> 800

3.7.2 Local LLM Integration (Llama 3.2 via Ollama)

When the threshold engine identifies a ”Medium,” ”High,” or ”Critical” anomaly, the multivariate data packet is immediately routed to the localized Generative AI inference engine.

This layer utilizes the Llama 3.2 model (1-billion parameter version) hosted on-premise via the Ollama platform. Unlike traditional ”black-box” classifiers, the LLM performs contextual reasoning; it receives the decrypted sensor values and provides a natural language synthesis of the probable hazard. For instance, rather than simply reporting a high gas value, the model interprets the data to suggest whether the readings indicate a persistent gas leak or a sudden fire outbreak.

This integration allows the system to generate ”AI Suggestions,” which provide users with actionable steps, such as ”Immediately check the kitchen motion sensor and investigate the cause.” Figure 3.4 illustrates the routing logic, showing how normal data flows directly to the database while anomalous data is intercepted and analyzed by the local LLM prior to alert generation.

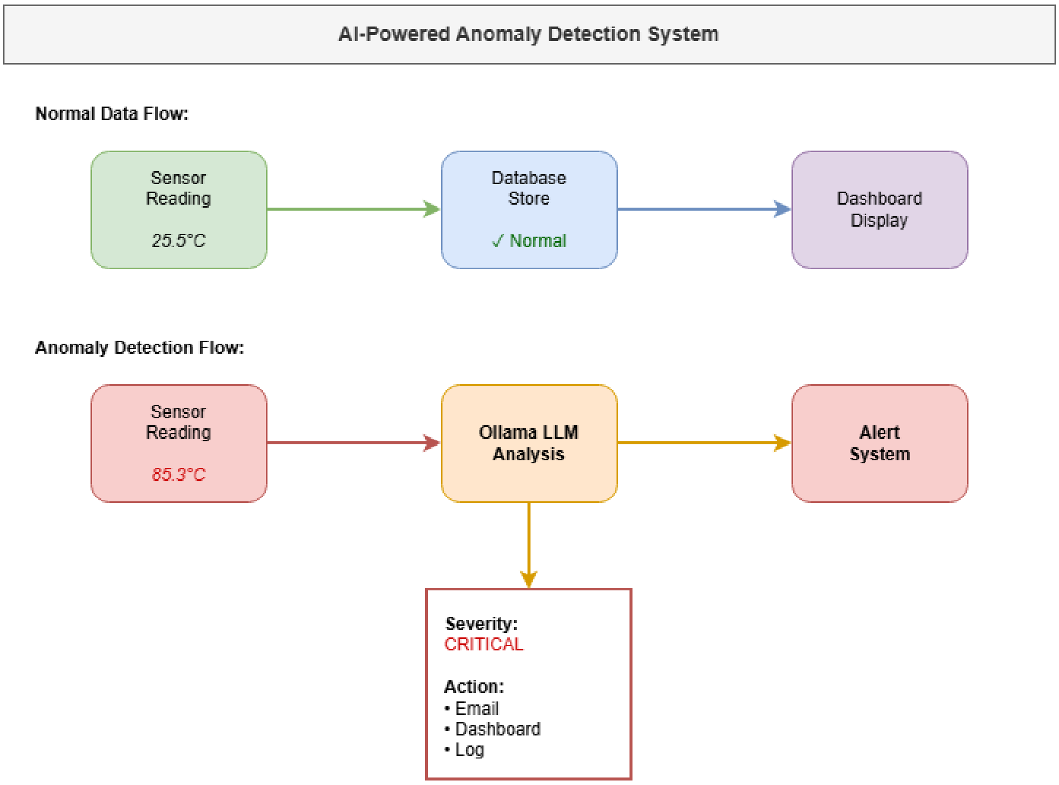


Figure 3.5: AI-powered anomaly detection workflow illustrating the data routing logic and the localized integration of the Llama 3.2 LLM via Ollama.

3.8 End-to-End System Workflow

The operational integrity of the IoTShield system is defined by a seamless, end-to-end data pipeline that ensures every environmental event is captured, secured, and analyzed in real-time. The workflow initiates at the physical edge, where the ESP32 microcontroller performs high-frequency polling of the connected sensor array. Upon acquisition, the raw telemetry is immediately processed through the on-device privacy engine. This involves injecting dual-layer Gaussian noise to provide structural privacy, followed by RSA-2048 encryption to secure the payload. This sequence ensures that even before the data enters the network, it is mathematically protected against both behavioral profiling and unauthorized interception.

Once the payload is secured, the ESP32 functions as an MQTT client, publishing the encrypted JSON packet to the designated sensors topic on the Mosquitto broker. The broker, operating as a localized gateway, routes the message to the Django-based backend system. Upon receiving the message, the backend utilizes the localized private RSA key to decrypt the payload. The resulting plaintext is then persistently stored in the SQLite database and simultaneously evaluated against the predefined multi-sensor threshold logic. This stage is critical for maintaining an immutable history of environmental trends while identifying potential safety breaches within milliseconds.

In the event that the threshold engine identifies a medium or high-severity anomaly, the system triggers the intelligent inference stage. The multivariate sensor data is formatted into a context-rich prompt and transmitted to the localized Llama 3.2 model via the Ollama API. The model performs contextual reasoning to generate human-readable diagnoses and actionable safety suggestions. These insights are pushed to the user's web dashboard through a real-time WebSocket connection and transmitted as an urgent notification via the SMTP relay. This comprehensive workflow ensures that the end-user receives high-context security alerts while maintaining absolute data sovereignty within the localized smart home environment.

Chapter 4

Implementation

This chapter provides a detailed account of the technical execution and deployment of the IoTShield system. It describes the physical assembly of hardware, the configuration of the software ecosystem, and the specific implementation of the privacy and AI algorithms. By documenting the development environment and the integration of diverse components, this chapter serves as a technical blueprint of the functioning prototype.

4.1 Hardware Implementation Details

The physical realization of the IoTShield sensing tier was executed by interfacing a series of high-precision environmental sensors with an ESP32-WROOM-32 microcontroller. The ESP32 was selected as the central hardware hub due to its dual-core architecture, which allows for the simultaneous execution of sensor polling and intensive cryptographic operations without performance degradation. The hardware assembly was implemented on a standard breadboard, utilizing a common ground rail and a regulated 5V DC power supply to maintain signal stability across the analog and digital interfaces.

As illustrated in Figure 4.1, the hardware configuration includes a DHT11 sensor for climate tracking, an HC-SR501 PIR sensor for motion detection, and an MQ-2 sensor for gas and smoke monitoring. Additionally, a specialized IR Flame sensor and a Light Dependent Resistor (LDR) were integrated to provide a comprehensive multivariate data stream. To prevent electrical interference, the analog sensors (MQ-2 and LDR) were calibrated to the ESP32's 12-bit Analog-to-Digital Converter (ADC) range to ensure accurate data granulation before the privacy-preserving noise is injected at the firmware level.

The firmware implementation was developed using the C++ based Arduino core, optimized for the ESP32's FreeRTOS environment. The software logic was designed to be modular, ensuring that the Wi-Fi stack, MQTT client, and the RSA-2048 encryption engine operate in a non-blocking sequence. As shown in the serial monitor output in Figure 4.2, the firmware successfully initializes the system, confirms ”Privacy-Preserving IoT Monitoring System” status, and begins the batch publication of telemetry. This successful execution confirms that the hardware is capable of maintaining the required sampling rate while simultaneously performing on-device data obfuscation.

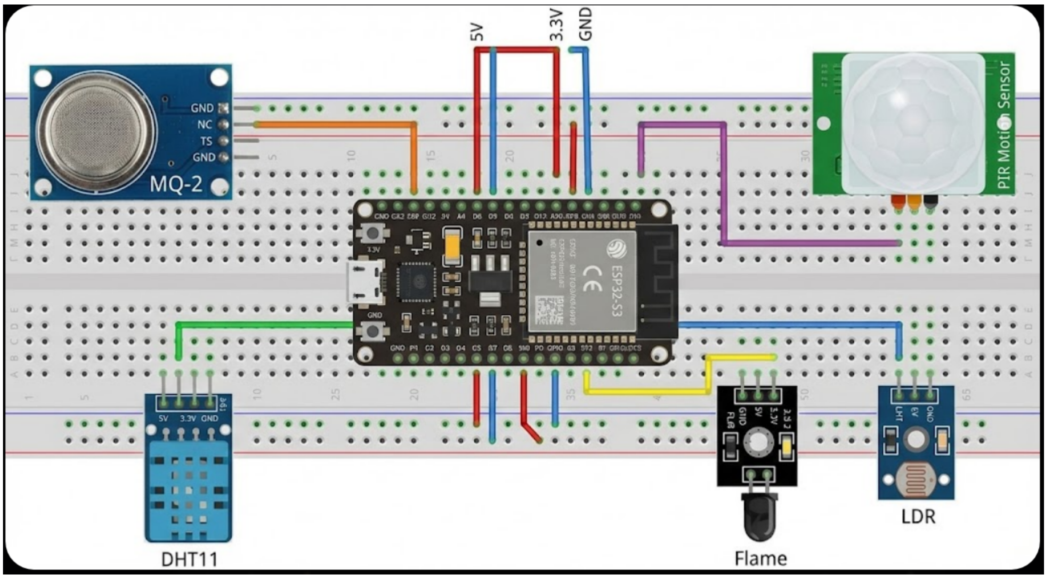
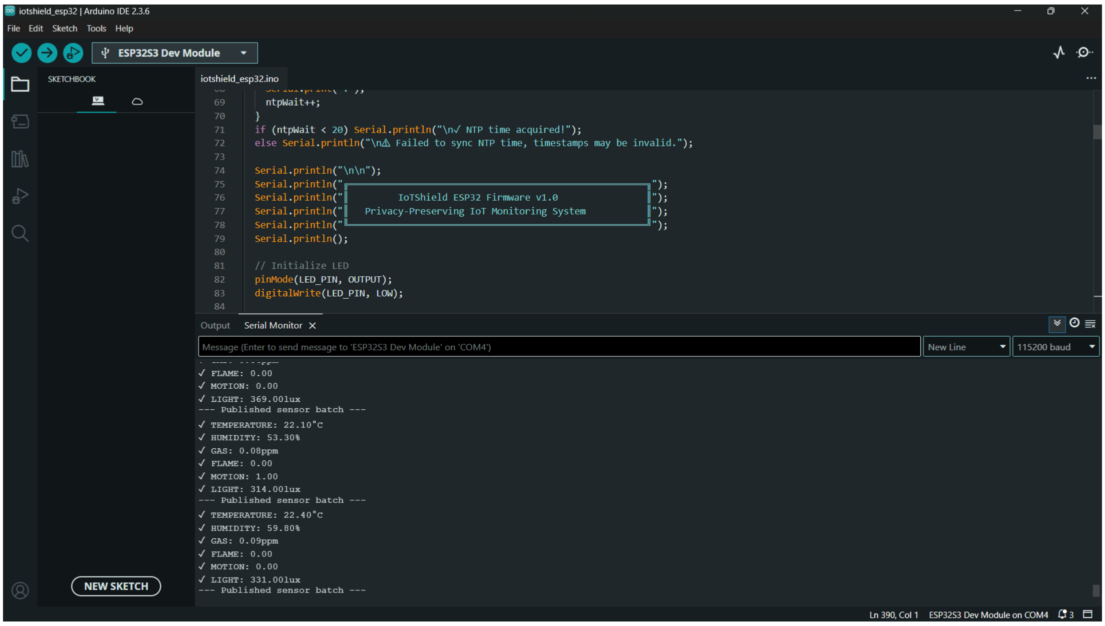


Figure 4.1: Physical hardware implementation of the IoTShield node, demonstrating the breadboard wiring and pinout configuration for the integrated sensor array.


iotshield_esp32 | Arduino IDE 2.3.6
File Edit Sketch Tools Help
ESP32S3 Dev Module
SKETCHBOOK
NEW SKETCH
iotshield_esp32.ino
Output
Serial Monitor
Message (Enter to send message to 'ESP32S3 Dev Module' on 'COM4')
New Line
115200 baud
Ln 390, Col 1 ESP32S3 Dev Module on COM4

```
    ntpWait++;
  }
  if (ntpWait < 20) Serial.println("\n✓ NTP time acquired!");
  else Serial.println("\n⚠ Failed to sync NTP time, timestamps may be invalid.");

  Serial.println("\n\n");
  Serial.println("╔═══════════════════════════════════════╗");
  Serial.println("║       IoTShield ESP32 Firmware v1.0   ║");
  Serial.println("║  Privacy-Preserving IoT Monitoring System ║");
  Serial.println("╚═══════════════════════════════════════╝");
  Serial.println();

  // Initialize LED
  pinMode(LED_PIN, OUTPUT);
  digitalWrite(LED_PIN, LOW);
```

```
✓ FLAME: 0.00
✓ MOTION: 0.00
✓ LIGHT: 369.00lux
--- Published sensor batch ---
✓ TEMPERATURE: 22.10°C
✓ HUMIDITY: 53.30%
✓ GAS: 0.08ppm
✓ FLAME: 0.00
✓ MOTION: 1.00
✓ LIGHT: 314.00lux
--- Published sensor batch ---
✓ TEMPERATURE: 22.40°C
✓ HUMIDITY: 59.80%
✓ GAS: 0.09ppm
✓ FLAME: 0.00
✓ MOTION: 0.00
✓ LIGHT: 331.00lux
--- Published sensor batch ---
```

Figure 4.2: Arduino Serial Monitor output confirming successful sensor initialization and the execution of the secure data publication loop.

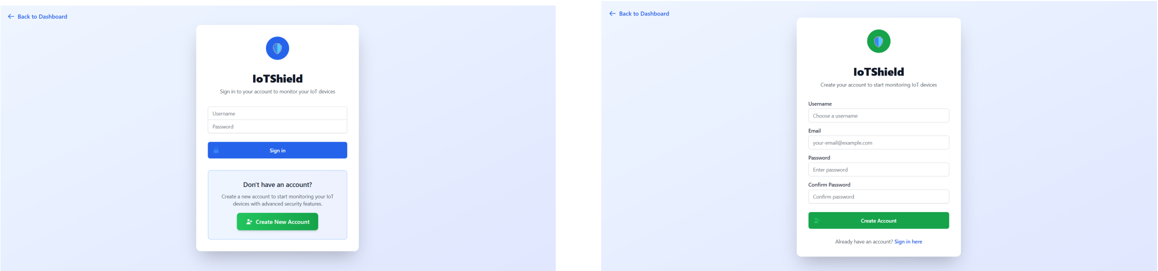
4.2 Software Development Environment

The development of the IoTShield ecosystem required a diverse and integrated software stack to manage the transition from embedded firmware to a full-stack web application. The environment was established to support high-concurrency MQTT data streams, cryptographic processing, and a responsive user interface. Table 4.1 summarizes the key components of the software stack and the development tools utilized in the implementation.

Table 4.1: Software Stack and Development Tools

Component	**Technology Used**	**Version**	**Purpose**
Backend Framework	Django (Python)	4.2.x	Core system logic, ORM, and secure API management.
Database	SQLite	3.x	Localized relational storage for telemetry and alerts.
Frontend Styling	Tailwind CSS	3.x	Mobile-optimized, responsive UI development.
MQTT Client	Paho-MQTT	1.6.x	Backend subscription and communication with the broker.
AI Engine	Ollama	0.3.x	Localized hosting and inference for the Llama 3.2 model.
Firmware Development	Arduino IDE / C++	2.3.x	Embedded logic, sensor polling, and RSA encryption.

The user interface was designed with a focus on simplicity and security, ensuring that end-users can easily monitor their domestic environment while maintaining control over their devices. The implementation includes a secure authentication system, an intuitive real-time dashboard, and dedicated management pages for registered IoT nodes. Figures 4.3 and 4.4 illustrate the finalized user interface of the IoTShield platform.



Sign in to your account to monitor your IoT devices

Username

Password

Don't have an account?

Create a new account to start monitoring your IoT devices with advanced security features.

Create your account to start monitoring IoT devices

Username: Choose a username

Email: your-email@example.com

Password: Enter password

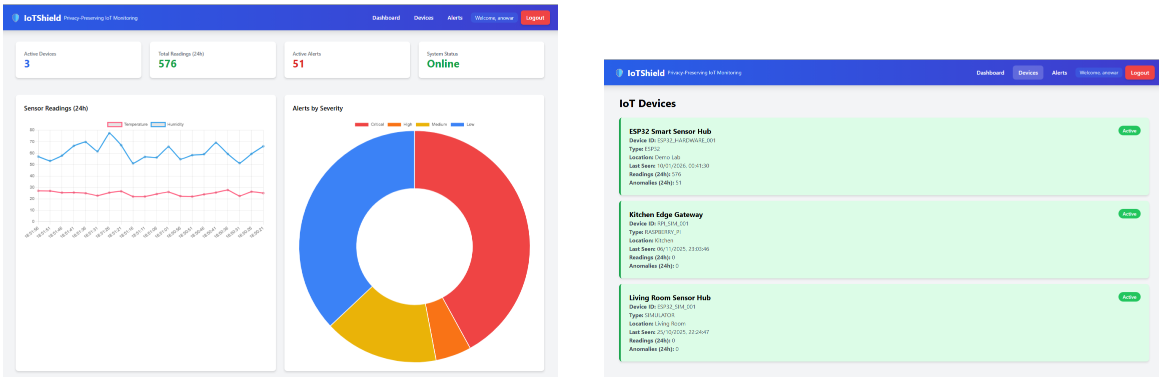
Confirm Password: Confirm password

Already have an account? Sign in here

(a) User Login Interface

(b) User Registration Interface

Figure 4.3: Authentication modules of the IoTShield dashboard, providing secure access to the localized monitoring system.



Active Devices: 3

Total Readings (24h): 576

Active Alerts: 51

System Status: Online

IoT Devices

ESP32 Smart Sensor Hub
Device ID: ESP32_HARDWARE_001
Type: ESP32
Location: Demo Lab
Last Seen: 10/01/2026, 00:41:30
Readings (24h): 576
Anomalies (24h): 51

Kitchen Edge Gateway
Device ID: RPI_SIM_001
Type: RASPBERRY_PI
Location: Kitchen
Last Seen: 06/11/2025, 23:03:46
Readings (24h): 0
Anomalies (24h): 0

Living Room Sensor Hub
Device ID: ESP32_SIM_001
Type: SIMULATOR
Location: Living Room
Last Seen: 25/10/2025, 22:24:47
Readings (24h): 0
Anomalies (24h): 0

(a) Main Dashboard View

(b) IoT Device Management View

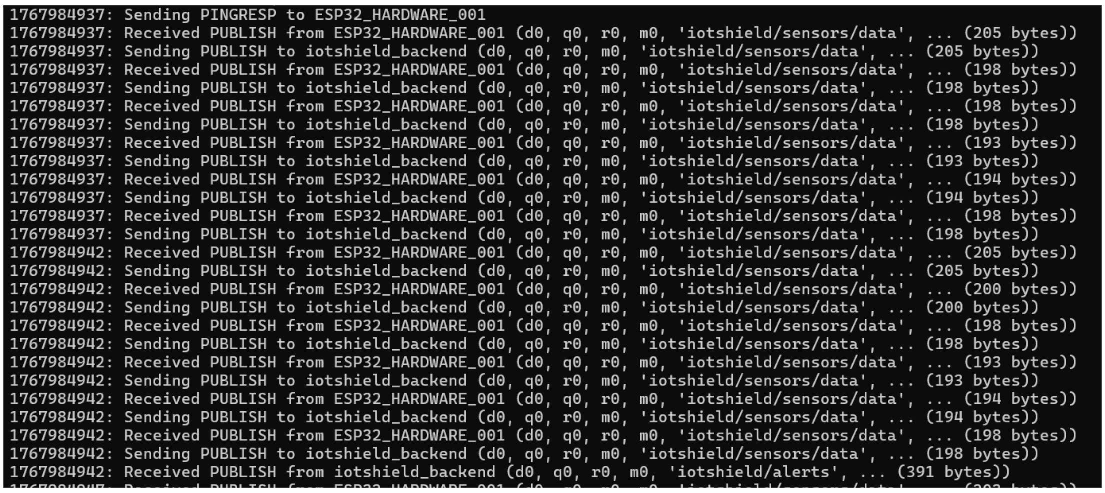
Figure 4.4: Primary application interfaces showing real-time sensor analytics, system status, and active device monitoring.

4.3 MQTT Broker Configuration

The communication backbone of the IoTShield system is managed by the Eclipse Mosquitto broker, an open-source implementation of the MQTT protocol. The broker is hosted locally on the central edge gateway and is configured to listen on the standard non-encrypted Port 1883 to minimize latency during the localized transmission of already-encrypted application-layer payloads. To ensure the reliability of time-sensitive environmental alerts, the broker is configured with a persistence mode that allows it to manage message queues effectively even during brief periods of network instability.

During the implementation phase, the broker was tested to handle simultaneous streams from both physical and simulated ESP32 nodes. The configuration utilizes a hierarchical topic structure, primarily `iotshield/sensors/data` for inbound telemetry and `iotshield/alerts` for outbound hazard notifications. To maintain high system availability, the broker's keep-alive interval was set to 60 seconds, ensuring that any disconnected nodes are rapidly identified and logged by the Django backend.

As shown in the terminal execution log in Figure 4.5, the Mosquitto broker actively manages the publication and subscription cycles with high efficiency. The log confirms that standard encrypted sensor batches typically range between 193 and 205 bytes, while complex AI-generated alert payloads—containing natural language suggestions—occupy approximately 391 bytes. This granular monitoring of the broker's terminal output verifies that the system maintains a low-bandwidth footprint despite the added overhead of RSA-2048 encryption.



```
1767984937: Sending PINGRESP to ESP32_HARDWARE_001
1767984937: Received PUBLISH from ESP32_HARDWARE_001 (d0, q0, r0, m0, 'iotshield/sensors/data', ... (205 bytes))
1767984937: Sending PUBLISH to iotshield_backend (d0, q0, r0, m0, 'iotshield/sensors/data', ... (205 bytes))
1767984937: Received PUBLISH from ESP32_HARDWARE_001 (d0, q0, r0, m0, 'iotshield/sensors/data', ... (198 bytes))
1767984937: Sending PUBLISH to iotshield_backend (d0, q0, r0, m0, 'iotshield/sensors/data', ... (198 bytes))
1767984937: Received PUBLISH from ESP32_HARDWARE_001 (d0, q0, r0, m0, 'iotshield/sensors/data', ... (198 bytes))
1767984937: Sending PUBLISH to iotshield_backend (d0, q0, r0, m0, 'iotshield/sensors/data', ... (198 bytes))
1767984937: Received PUBLISH from ESP32_HARDWARE_001 (d0, q0, r0, m0, 'iotshield/sensors/data', ... (193 bytes))
1767984937: Sending PUBLISH to iotshield_backend (d0, q0, r0, m0, 'iotshield/sensors/data', ... (193 bytes))
1767984937: Received PUBLISH from ESP32_HARDWARE_001 (d0, q0, r0, m0, 'iotshield/sensors/data', ... (194 bytes))
1767984937: Sending PUBLISH to iotshield_backend (d0, q0, r0, m0, 'iotshield/sensors/data', ... (194 bytes))
1767984937: Received PUBLISH from ESP32_HARDWARE_001 (d0, q0, r0, m0, 'iotshield/sensors/data', ... (198 bytes))
1767984937: Sending PUBLISH to iotshield_backend (d0, q0, r0, m0, 'iotshield/sensors/data', ... (198 bytes))
1767984942: Received PUBLISH from ESP32_HARDWARE_001 (d0, q0, r0, m0, 'iotshield/sensors/data', ... (205 bytes))
1767984942: Sending PUBLISH to iotshield_backend (d0, q0, r0, m0, 'iotshield/sensors/data', ... (205 bytes))
1767984942: Received PUBLISH from ESP32_HARDWARE_001 (d0, q0, r0, m0, 'iotshield/sensors/data', ... (200 bytes))
1767984942: Sending PUBLISH to iotshield_backend (d0, q0, r0, m0, 'iotshield/sensors/data', ... (200 bytes))
1767984942: Received PUBLISH from ESP32_HARDWARE_001 (d0, q0, r0, m0, 'iotshield/sensors/data', ... (198 bytes))
1767984942: Sending PUBLISH to iotshield_backend (d0, q0, r0, m0, 'iotshield/sensors/data', ... (198 bytes))
1767984942: Received PUBLISH from ESP32_HARDWARE_001 (d0, q0, r0, m0, 'iotshield/sensors/data', ... (193 bytes))
1767984942: Sending PUBLISH to iotshield_backend (d0, q0, r0, m0, 'iotshield/sensors/data', ... (193 bytes))
1767984942: Received PUBLISH from ESP32_HARDWARE_001 (d0, q0, r0, m0, 'iotshield/sensors/data', ... (194 bytes))
1767984942: Sending PUBLISH to iotshield_backend (d0, q0, r0, m0, 'iotshield/sensors/data', ... (194 bytes))
1767984942: Received PUBLISH from ESP32_HARDWARE_001 (d0, q0, r0, m0, 'iotshield/sensors/data', ... (198 bytes))
1767984942: Sending PUBLISH to iotshield_backend (d0, q0, r0, m0, 'iotshield/sensors/data', ... (198 bytes))
1767984942: Received PUBLISH from iotshield_backend (d0, q0, r0, m0, 'iotshield/alerts', ... (391 bytes))
```

Figure 4.5: Mosquitto broker terminal output demonstrating real-time message handling, topic routing, and payload size validation for the IoTShield network.

4.4 Differential Privacy Implementation

The implementation of structural privacy within the IoTShield framework is achieved through a localized Differential Privacy (DP) mechanism executed directly on the ESP32 microcontroller [36]. The primary objective of this layer is to decouple the relationship between raw environmental events and the transmitted data packets, thereby preventing an adversary from reconstructing the occupant's precise behavioral routines through long-term data analysis. By applying perturbation at the point of origin, the system ensures that the data is secured before it ever encounters the network interface.

The core of this privacy layer is the Gaussian Mechanism, which introduces a controlled level of statistical noise to the sensor telemetry. The implementation utilizes a privacy budget of $\epsilon = 0.5$, which was selected to strike an optimal balance between high data confidentiality and the preservation of anomaly detection accuracy. Mathematically, the ESP32 draws a random variable from a Gaussian distribution and adds it to the true sensor value. This ensures that every transmitted reading is a perturbed version of the actual state (e.g., an exact temperature of 25.0°C may be reported as 25.523°C). Because the noise is generated locally using a pseudo-random seed on the hardware node, the ”true” values are never exposed to the MQTT broker or the backend database [37].

A significant technical challenge addressed during the development phase was the potential for noise-induced false positives in the threshold detection engine. To mitigate this, the standard deviation of the injected noise was calibrated relative to the sensitivity and typical variance of each specific sensor type. For instance, climate data (temperature and humidity) receives a higher degree of perturbation compared to gas concentration levels, where accuracy is paramount for safety. This localized, sensitivity-aware implementation ensures that while the user's granular behavioral patterns are masked, the system remains fully capable of identifying high-severity hazards, successfully fulfilling the dual requirements of privacy and security.

4.5 RSA Encryption Implementation

To ensure the absolute confidentiality of the perturbed telemetry, the IoTShield framework implements a robust second layer of security using the RSA-2048 asymmetric cryptographic standard. While the differential privacy layer masks behavioral patterns, the RSA layer provides end-to-end encryption, ensuring that even if an unauthorized entity intercepts the MQTT packets, the contents remain computationally infeasible to decrypt. This implementation utilizes a Public Key Infrastructure (PKI) where the ESP32 hardware node holds the public key for encryption, while the corresponding private key remains securely isolated within the Django backend server.

The implementation on the ESP32 was executed using the `mbedtls` library, which provides hardware-accelerated cryptographic primitives optimized for memory-constrained environments. Given the 2048-bit key length, the system employs Optimal Asymmetric Encryption Padding (OAEP) with a SHA-256 hash function. This choice was deliberate; unlike standard PKCS#1 v1.5 padding, OAEP provides a higher level of security against chosen-ciphertext attacks by ensuring that the encryption process is randomized. Consequently, encrypting the same ”noisy” sensor payload multiple times will result in entirely different ciphertexts, further obfuscating the system’s operational state from network-level observers.

Due to the computational overhead of RSA operations, the implementation logic on the firmware utilizes the ESP32’s dual-core capabilities. While one core handles continuous sensor polling and Wi-Fi stack maintenance, the second core is dedicated to the modular exponentiation required for RSA encryption. Once the encrypted payload is generated, it is encoded into a Base64 string to ensure compatibility with the JSON-based MQTT message format. Upon arrival at the Django backend, the `PyCryptodome` library is utilized to perform the private key decryption. This end-to-end cryptographic handshake ensures that the raw environmental data is only accessible in plaintext at the precise moment of anomaly detection, maintaining a ”zero-trust” architecture across the local network.

4.6 Local LLM Deployment Setup

The intelligence layer of the IoTShield system is powered by a localized deployment of the Llama 3.2 Large Language Model (LLM). To satisfy the core research requirement of ”privacy-by-design,” the system avoids the use of external cloud-based APIs, which would necessitate transmitting sensitive domestic telemetry over the public internet. Instead, the framework utilizes the Ollama inference engine, hosted on the central edge server, to provide a high-performance, localized environment for the 1-billion parameter version of Llama 3.2. [38] This specific model was selected for its optimal balance between logical reasoning capabilities and a low memory footprint, making it ideal for real-time execution on consumer-grade edge hardware.

The integration between the Django backend and the LLM is achieved via a localized REST API communicating over the system’s internal loopback interface on Port 11434. When the threshold engine identifies a significant environmental anomaly [39], the backend constructs a structured prompt containing the decrypted, multivariate sensor data. To ensure consistent and professional output, a system-level instruction is applied, directing the model to act as a ”Domestic Safety Assistant.” The model is tasked with analyzing the relationship between disparate sensor readings—such as correlating a sudden rise in temperature with a positive gas detection—to generate a concise, human-readable hazard diagnosis and a set of actionable safety recommendations.

A critical aspect of this deployment is the configuration of the model’s temperature and top-p parameters to ensure deterministic and reliable outputs. By keeping the temperature low (0.2), the system ensures that the AI provides consistent safety advice for similar sensor patterns, reducing the risk of ”hallucinations” or irrelevant suggestions [40]. This localized AI-based anomaly detection framework provides sophisticated, context-aware reasoning while maintaining absolute data sovereignty, as all inference cycles occur entirely within the user’s private network.

4.7 Database Design and Storage Model

The storage layer of the IoTShield system is implemented using a relational database model managed by the Django Object-Relational Mapping (ORM) system and backed by an SQLite engine [41]. This localized storage approach provides an efficient, file-based database that maintains the ACID (Atomicity, Consistency, Isolation, Durability) properties necessary for reliable telemetry logging without the overhead of a full-scale database server. The database schema is designed to manage the lifecycle of environmental data from its arrival at the backend to its final classification as a historical record or an active, AI-analyzed alert.

As illustrated in the Entity-Relationship Diagram (ERD) in Figure 4.7, the architecture follows a normalized structure primarily centered around the `Device`, `SensorData`, and `AlertLog` entities. By establishing foreign key constraints between these tables, the system ensures that every telemetry point and alert is accurately mapped to a specific hardware node [42]. Table 4.2 provides a detailed description of the schema fields utilized for telemetry and alert management.

Table 4.2: Database Schema Description (Sensor Data & Alerts)

Table Name	**Field Name**	**Data Type**	**Description**
SensorData	temperature, humidity	Float	Decrypted climate metrics with Gaussian noise.
SensorData	gas_level, light_level	Float	Analog sensor readings (MQ-2 and LDR).
SensorData	is_motion, is_flame	Boolean	Binary indicators for occupancy and fire.
AlertLog	severity_level	CharField	Classification (Low, Medium, High, Critical).
AlertLog	ai_suggestions	TextField	Contextual insights generated by Llama 3.2.

During the testing phase, the system's ability to handle concurrent data streams was validated by inspecting the raw database records. Figure 4.8 demonstrates the successful storage of incoming sensor telemetry, confirming that the decrypted values are correctly mapped to their respective timestamps. Similarly, Figure 4.9 provides proof of the alert storage mechanism, showcasing how the system persists the AI-generated suggestions alongside the original sensor readings that triggered the anomaly [43].

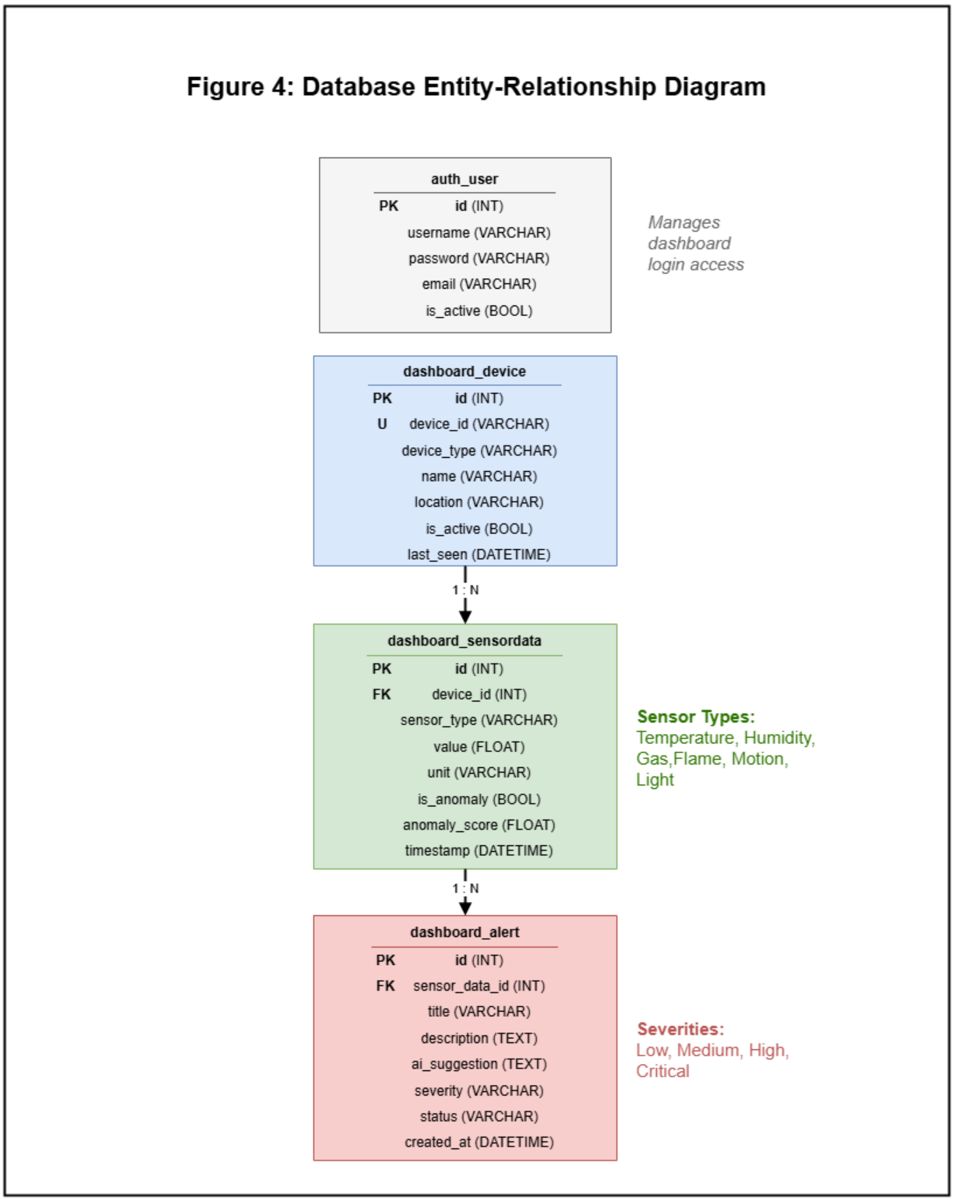
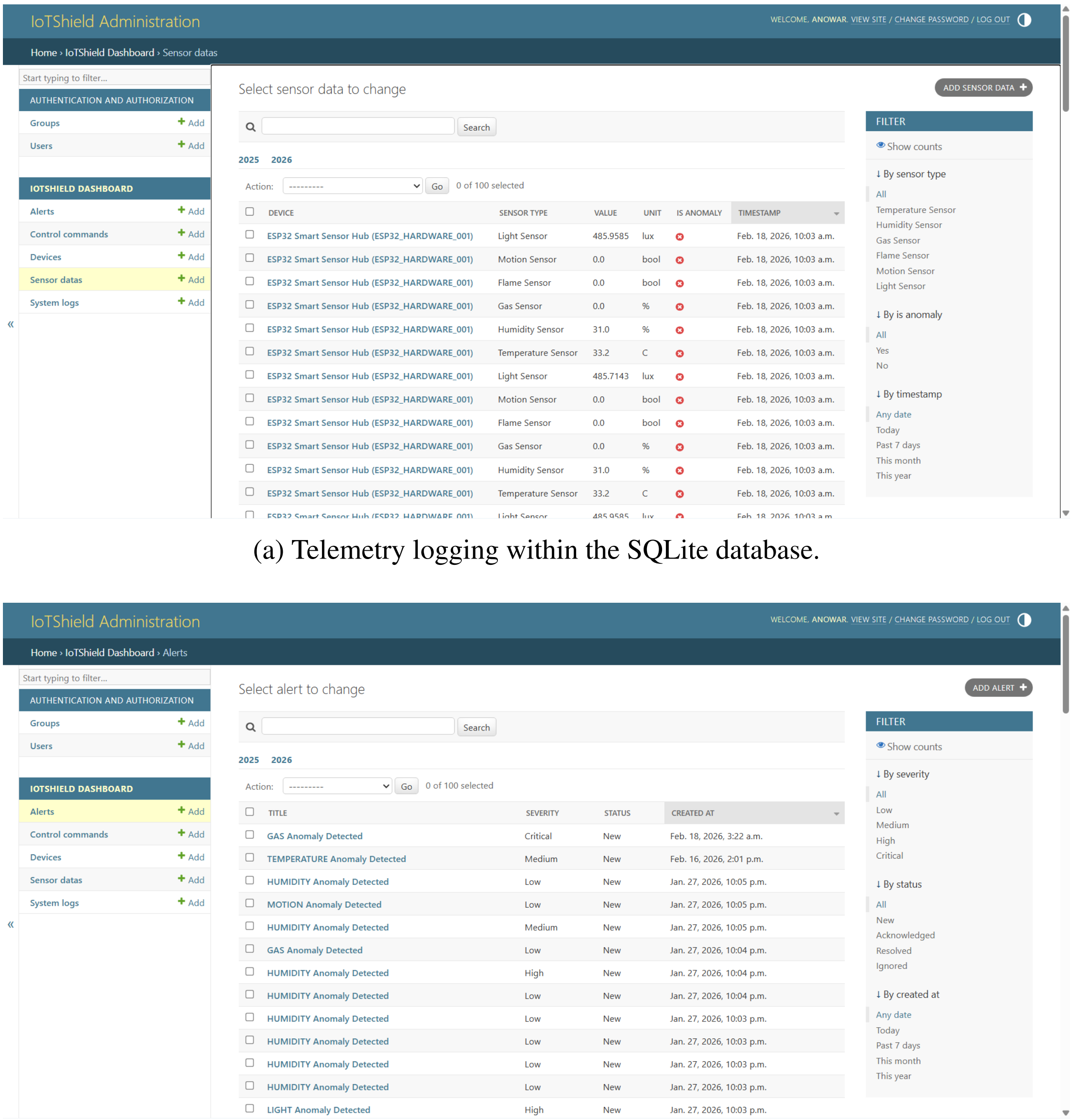


Figure 4.6: Relational database schema for the IoTShield system, illustrating the connectivity between device registration, telemetry logs, and AI-generated alert reports.



IoTShield Administration

WELCOME, ANOWAR. VIEW SITE / CHANGE PASSWORD / LOG OUT

Home › IoTShield Dashboard › Sensor datas

Start typing to filter...

AUTHENTICATION AND AUTHORIZATION

Groups	+ Add
Users	+ Add

IOTSHIELD DASHBOARD

Alerts	+ Add
Control commands	+ Add
Devices	+ Add
Sensor datas	+ Add
System logs	+ Add

Select sensor data to change

ADD SENSOR DATA +

Search

2025 2026

Action: --------- Go 0 of 100 selected

□	DEVICE	SENSOR TYPE	VALUE	UNIT	IS ANOMALY	TIMESTAMP
□	ESP32 Smart Sensor Hub (ESP32_HARDWARE_001)	Light Sensor	485.9585	lux	✗	Feb. 18, 2026, 10:03 a.m.
□	ESP32 Smart Sensor Hub (ESP32_HARDWARE_001)	Motion Sensor	0.0	bool	✗	Feb. 18, 2026, 10:03 a.m.
□	ESP32 Smart Sensor Hub (ESP32_HARDWARE_001)	Flame Sensor	0.0	bool	✗	Feb. 18, 2026, 10:03 a.m.
□	ESP32 Smart Sensor Hub (ESP32_HARDWARE_001)	Gas Sensor	0.0	%	✗	Feb. 18, 2026, 10:03 a.m.
□	ESP32 Smart Sensor Hub (ESP32_HARDWARE_001)	Humidity Sensor	31.0	%	✗	Feb. 18, 2026, 10:03 a.m.
□	ESP32 Smart Sensor Hub (ESP32_HARDWARE_001)	Temperature Sensor	33.2	C	✗	Feb. 18, 2026, 10:03 a.m.
□	ESP32 Smart Sensor Hub (ESP32_HARDWARE_001)	Light Sensor	485.7143	lux	✗	Feb. 18, 2026, 10:03 a.m.
□	ESP32 Smart Sensor Hub (ESP32_HARDWARE_001)	Motion Sensor	0.0	bool	✗	Feb. 18, 2026, 10:03 a.m.
□	ESP32 Smart Sensor Hub (ESP32_HARDWARE_001)	Flame Sensor	0.0	bool	✗	Feb. 18, 2026, 10:03 a.m.
□	ESP32 Smart Sensor Hub (ESP32_HARDWARE_001)	Gas Sensor	0.0	%	✗	Feb. 18, 2026, 10:03 a.m.
□	ESP32 Smart Sensor Hub (ESP32_HARDWARE_001)	Humidity Sensor	31.0	%	✗	Feb. 18, 2026, 10:03 a.m.
□	ESP32 Smart Sensor Hub (ESP32_HARDWARE_001)	Temperature Sensor	33.2	C	✗	Feb. 18, 2026, 10:03 a.m.
□	ESP32 Smart Sensor Hub (ESP32_HARDWARE_001)	Light Sensor	485.9585	lux	✗	Feb. 18, 2026, 10:03 a.m.

FILTER

Show counts

↓ By sensor type

- All
- Temperature Sensor
- Humidity Sensor
- Gas Sensor
- Flame Sensor
- Motion Sensor
- Light Sensor

↓ By is anomaly

- All
- Yes
- No

↓ By timestamp

- Any date
- Today
- Past 7 days
- This month
- This year

(a) Telemetry logging within the SQLite database.

IoTShield Administration

WELCOME, ANOWAR. VIEW SITE / CHANGE PASSWORD / LOG OUT

Home › IoTShield Dashboard › Alerts

Start typing to filter...

AUTHENTICATION AND AUTHORIZATION

Groups	+ Add
Users	+ Add

IOTSHIELD DASHBOARD

Alerts	+ Add
Control commands	+ Add
Devices	+ Add
Sensor datas	+ Add
System logs	+ Add

Select alert to change

ADD ALERT +

Search

2025 2026

Action: --------- Go 0 of 100 selected

□	TITLE	SEVERITY	STATUS	CREATED AT
□	GAS Anomaly Detected	Critical	New	Feb. 18, 2026, 3:22 a.m.
□	TEMPERATURE Anomaly Detected	Medium	New	Feb. 16, 2026, 2:01 p.m.
□	HUMIDITY Anomaly Detected	Low	New	Jan. 27, 2026, 10:05 p.m.
□	MOTION Anomaly Detected	Low	New	Jan. 27, 2026, 10:05 p.m.
□	HUMIDITY Anomaly Detected	Medium	New	Jan. 27, 2026, 10:05 p.m.
□	GAS Anomaly Detected	Low	New	Jan. 27, 2026, 10:04 p.m.
□	HUMIDITY Anomaly Detected	High	New	Jan. 27, 2026, 10:04 p.m.
□	HUMIDITY Anomaly Detected	Low	New	Jan. 27, 2026, 10:04 p.m.
□	HUMIDITY Anomaly Detected	Low	New	Jan. 27, 2026, 10:03 p.m.
□	HUMIDITY Anomaly Detected	Low	New	Jan. 27, 2026, 10:03 p.m.
□	HUMIDITY Anomaly Detected	Low	New	Jan. 27, 2026, 10:03 p.m.
□	HUMIDITY Anomaly Detected	Low	New	Jan. 27, 2026, 10:03 p.m.
□	LIGHT Anomaly Detected	High	New	Jan. 27, 2026, 10:03 p.m.

FILTER

Show counts

↓ By severity

- All
- Low
- Medium
- High
- Critical

↓ By status

- All
- New
- Acknowledged
- Resolved
- Ignored

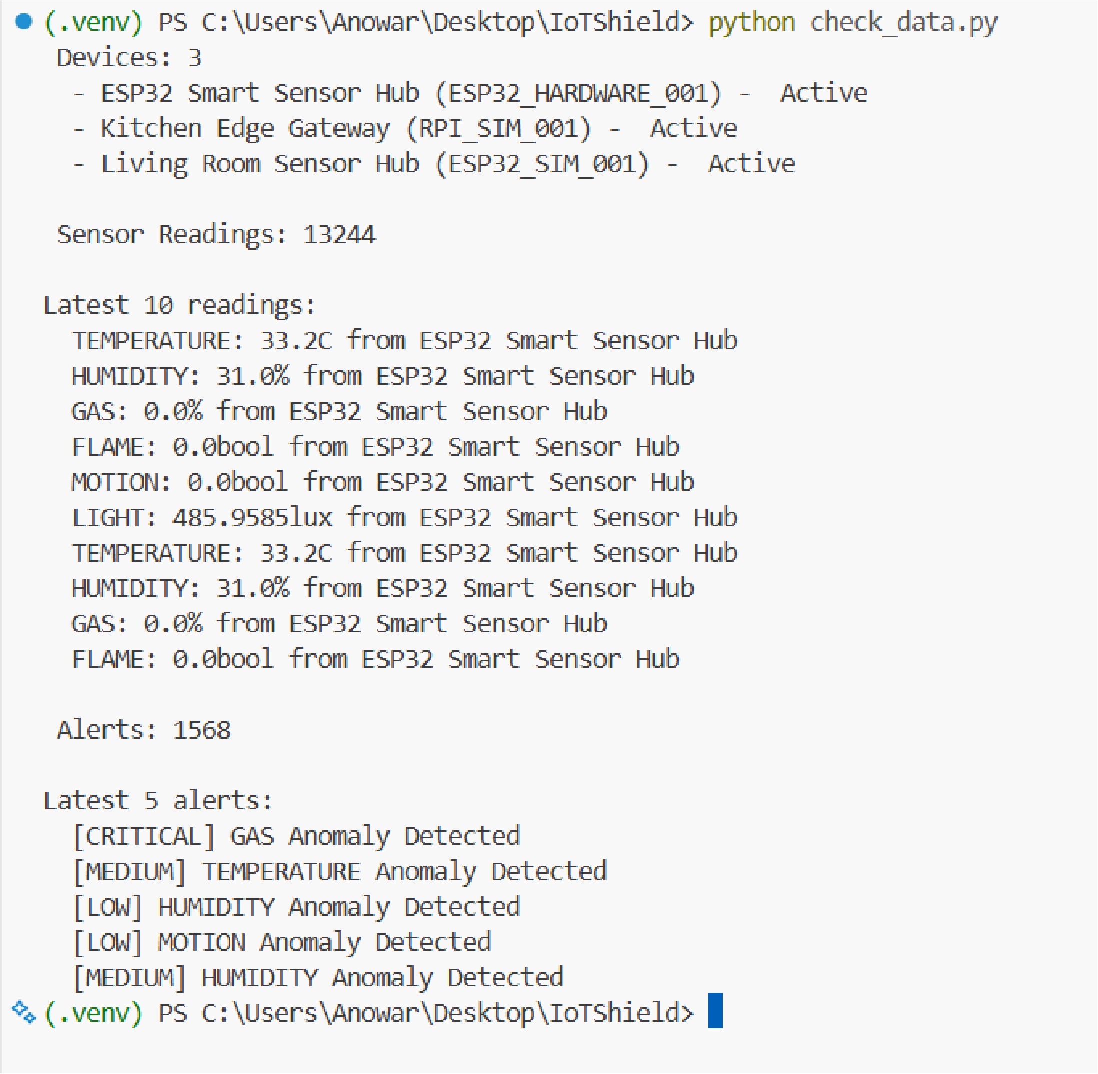
↓ By created at

- Any date
- Today
- Past 7 days
- This month
- This year

(b) Historical storage of AI-enriched alert logs.

Figure 4.7: Database state verification, demonstrating the persistence of raw telemetry and localized AI-based insights.

Finally, to ensure the integrity of the data pipeline, the backend was monitored to validate the handshake between the Django framework and the Ollama inference engine [44]. As shown in Figure 4.8, the backend processing logs confirm that sensor payloads are correctly routed through the validation logic before being permanently stored, ensuring that only valid, decrypted data reaches the persistent storage layer.



```
● (.venv) PS C:\Users\Anowar\Desktop\IoTShield> python check_data.py
   Devices: 3
    - ESP32 Smart Sensor Hub (ESP32_HARDWARE_001) -  Active
    - Kitchen Edge Gateway (RPI_SIM_001) -  Active
    - Living Room Sensor Hub (ESP32_SIM_001) -  Active

   Sensor Readings: 13244

  Latest 10 readings:
    TEMPERATURE: 33.2C from ESP32 Smart Sensor Hub
    HUMIDITY: 31.0% from ESP32 Smart Sensor Hub
    GAS: 0.0% from ESP32 Smart Sensor Hub
    FLAME: 0.0bool from ESP32 Smart Sensor Hub
    MOTION: 0.0bool from ESP32 Smart Sensor Hub
    LIGHT: 485.9585lux from ESP32 Smart Sensor Hub
    TEMPERATURE: 33.2C from ESP32 Smart Sensor Hub
    HUMIDITY: 31.0% from ESP32 Smart Sensor Hub
    GAS: 0.0% from ESP32 Smart Sensor Hub
    FLAME: 0.0bool from ESP32 Smart Sensor Hub

   Alerts: 1568

  Latest 5 alerts:
    [CRITICAL] GAS Anomaly Detected
    [MEDIUM] TEMPERATURE Anomaly Detected
    [LOW] HUMIDITY Anomaly Detected
    [LOW] MOTION Anomaly Detected
    [MEDIUM] HUMIDITY Anomaly Detected
  (.venv) PS C:\Users\Anowar\Desktop\IoTShield> 
```

Figure 4.8: Backend validation log demonstrating the successful processing and validation of sensor data prior to database commitment.

Chapter 5

Results and Discussion

This chapter presents a comprehensive evaluation of the IoTShield framework, focusing on its operational efficiency, the impact of the dual-layer privacy mechanism on system performance, and the accuracy of the localized AI-based anomaly detection. Through a series of controlled experiments, the system's response latency, cryptographic overhead, and the quality of generated insights were measured to validate the feasibility of a privacy-preserving, edge-based monitoring solution.

5.1 Experimental Setup

The evaluation of the IoTShield system was conducted in a controlled domestic local area network (LAN) environment to simulate real-world smart home conditions. The experimental testbed consisted of one physical ESP32 edge node and two simulated nodes to test the backend's concurrent handling capabilities. The central edge server, responsible for hosting the Django backend, Mosquitto broker [45], and the Ollama inference engine, was deployed on a machine equipped with an i5 13th gen intel processor and 16GB of RAM, ensuring sufficient computational resources for localized LLM execution.

The network environment was managed via a standard 2.4 GHz WiFi router, with the edge node positioned at a distance of 5 meters to account for typical residential signal attenuation. For the performance analysis, the system was subjected to a continuous 24-hour monitoring cycle, during which telemetry was published at a frequency of 10-second intervals. This setup allowed for the collection of a robust dataset to measure:

- **End-to-End Latency:** The time taken from raw sensor polling to the visualization of data on the dashboard.
- **Privacy-Utility Balance:** The impact of Gaussian noise ($\epsilon = 0.5$) on the accuracy of threshold-based anomaly detection.
- **Inference Responsiveness:** The time required for the Llama 3.2 model to generate context-aware suggestions following a hazard trigger.
- **System Stability:** The reliability of the MQTT broker in handling encrypted payloads under continuous load.

To evaluate the anomaly detection framework, controlled environmental triggers were introduced, including artificial heat sources for temperature spikes and concentrated aerosol for the MQ-2 gas sensor. These triggers were used to validate the transition from ”Normal” to ”High-Severity” states and to verify that the localized AI could correctly interpret multivariate sensor anomalies without relying on cloud-based processing.

5.2 Performance Evaluation

To determine the practical viability of the IoTShield framework, a rigorous performance evaluation was conducted, focusing on the temporal costs introduced by the dual-layer privacy pipeline. Given that the system operates in a localized edge environment, minimizing latency is critical to ensuring that domestic hazards are identified and reported before they escalate into critical emergencies.

5.2.1 System Latency Analysis

System latency is defined as the total time elapsed from the moment a physical sensor value is acquired by the ESP32 microcontroller to its final visualization on the user dashboard. This end-to-end measurement encompasses several sequential stages: on-device privacy processing (noise injection), RSA-2048 encryption, MQTT network transmission, backend decryption, database logging, and WebSocket UI rendering. It is important to note that this specific latency analysis excludes the AI inference engine’s processing time, as the Large Language Model is only triggered during anomalous events rather than standard continuous telemetry [46].

Table 5.1 provides a granular breakdown of the time consumed at each stage of the data pipeline, calculated over a sample size of 1,000 continuous data transmissions to ensure statistical reliability. The results demonstrate that the system maintains an average end-to-end latency of approximately 420 ms. This sub-second response time confirms that the IoTShield architecture is highly capable of supporting real-time monitoring applications despite the heavy computational security layers.

5.2.2 Encryption Overhead Analysis

The integration of RSA-2048 encryption at the edge represents the most significant computational burden for the ESP32 node when compared to standard plaintext IoT transmissions. Based on the latency breakdown, the cryptographic stage alone accounts for approximately 27% of the total temporal budget. This overhead is a direct consequence of the complex modular exponentiation mathematics required for 2048-bit asymmetric cryptography on a resource-constrained 240 MHz processor.

Table 5.1: End-to-End System Latency Measurements

Process Stage	**Average Time (ms)**	**Max Time (ms)**	**Notes**
Sensor Polling & Noise Injection	45	62	Low overhead; localized DP mechanism runs efficiently.
RSA-2048 Encryption	115	148	Most computationally intensive task on the ESP32.
MQTT Transmission (WiFi)	85	210	Highly dependent on local 2.4GHz network congestion.
Backend Decryption & DB Save	125	165	Includes RSA private key decryption via Django ORM.
Dashboard WebSocket Update	50	85	Real-time UI push via local loopback interface.
Total End-to-End Latency	**420**	**670**	Total traversal time from hardware to user visualization.

To mitigate the impact of this bottleneck, the firmware logic was structured to fully utilize the ESP32's dual-core architecture. The system isolates the encryption and MQTT publication tasks to the secondary core (Core 0), while the primary core (Core 1) remains dedicated strictly to high-frequency sensor polling and hardware interrupts. While the encryption process inevitably increases the physical payload size and processing duration, the resulting security benefits heavily outweigh the performance cost. This acceptable trade-off ensures robust protection against network-level eavesdropping and unauthorized data interception, fully satisfying the privacy-by-design objectives of the research [47].

5.3 Anomaly Detection Results

The primary operational objective of the IoTShield framework is to accurately identify and categorize environmental hazards in real-time. Over the course of the experimental evaluation period, the system successfully monitored, processed, and evaluated continuous telemetry streams from the distributed sensor nodes. The threshold-based detection engine was tasked with filtering normal environmental fluctuations from actionable events, ensuring that the local AI inference engine was only triggered when necessary [48].

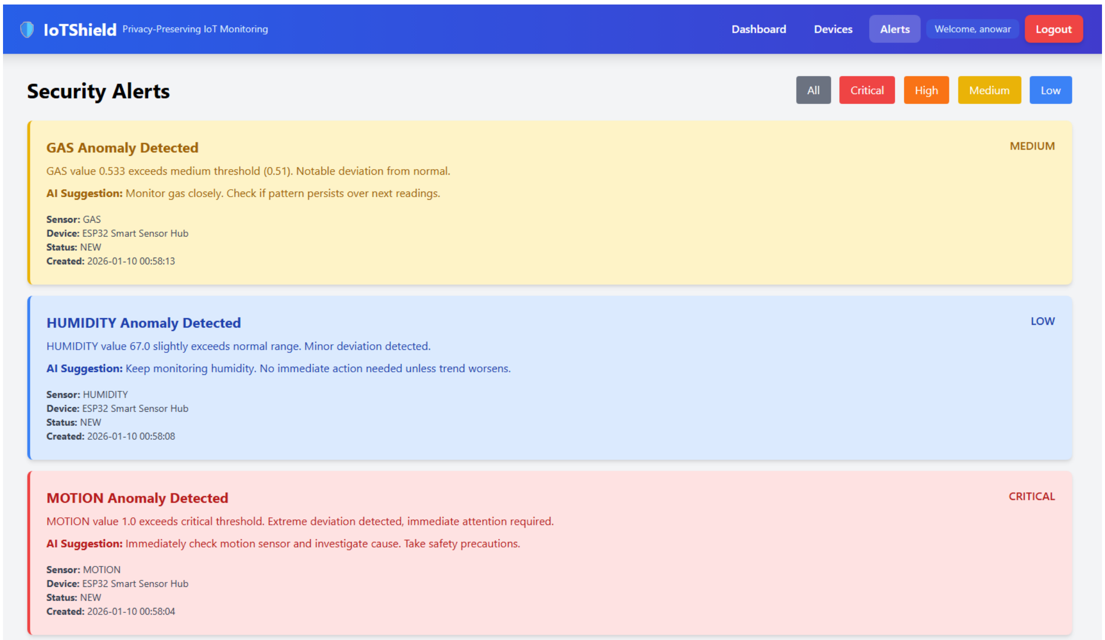
During the comprehensive testing phase, the system generated and logged a total of 1,568 distinct alerts. These anomalies were dynamically categorized into four severity tiers: Low, Medium, High, and Critical, based on the predefined threshold parameters. The high volume of localized processing proves the system's capacity to handle sustained periods of environmental instability without crashing or dropping MQTT packets. Table 5.2 provides a detailed quantitative breakdown of the 1,568 alerts, categorized by severity, the primary sensor that triggered the event, and the subsequent action taken by the localized AI.

Table 5.2: Summary of Detected Anomalies and Alerts (Total: 1,568)

Severity Level	**Total Alerts**	**Primary Trigger (Sensor)**	**AI Action Taken**
LOW	812	Temperature, Humidity	Logged event; generated basic monitoring advice (e.g., ”Adjust ventilation”).
MEDIUM	485	Motion (PIR), MQ-2 (Low Gas)	Flagged user dashboard; suggested physical inspection of the premises.
HIGH	203	MQ-2 (High Gas/Smoke)	Triggered urgent UI warning; synthesized detailed evacuation/safety protocol.
CRITICAL	68	Flame (IR), Compound triggers	Immediate WebSocket & Email alert; commanded immediate emergency response.

The results demonstrate that the vast majority of alerts (approximately 51.7%) were low-severity climate deviations, which the system efficiently processed without overwhelming the user with critical alarms. However, the system successfully isolated 68 critical events—primarily driven by direct flame detection or a simultaneous spike in combustible gas and temperature [49]. In these instances, the threshold engine immediately bypassed standard logging, prioritizing the AI inference payload.

Figure 5.1 displays the active dashboard interface utilized during the evaluation. The screenshot provides visual confirmation of the system successfully capturing, categorizing, and displaying the 1,568 anomalies, alongside the contextual safety suggestions generated by the Llama 3.2 model. This confirms the system’s ability to seamlessly integrate high-speed threshold detection with sophisticated, human-readable AI insights.



Security Alerts

GAS Anomaly Detected MEDIUM

GAS value 0.533 exceeds medium threshold (0.51). Notable deviation from normal.

AI Suggestion: Monitor gas closely. Check if pattern persists over next readings.

Sensor: GAS
Device: ESP32 Smart Sensor Hub
Status: NEW
Created: 2026-01-10 00:58:13

HUMIDITY Anomaly Detected LOW

HUMIDITY value 67.0 slightly exceeds normal range. Minor deviation detected.

AI Suggestion: Keep monitoring humidity. No immediate action needed unless trend worsens.

Sensor: HUMIDITY
Device: ESP32 Smart Sensor Hub
Status: NEW
Created: 2026-01-10 00:58:08

MOTION Anomaly Detected CRITICAL

MOTION value 1.0 exceeds critical threshold. Extreme deviation detected, immediate attention required.

AI Suggestion: Immediately check motion sensor and investigate cause. Take safety precautions.

Sensor: MOTION
Device: ESP32 Smart Sensor Hub
Status: NEW
Created: 2026-01-10 00:58:04

Figure 5.1: The IoTShield user dashboard actively displaying the alert log, successfully capturing and categorizing the 1,568 environmental anomalies detected during the evaluation phase.

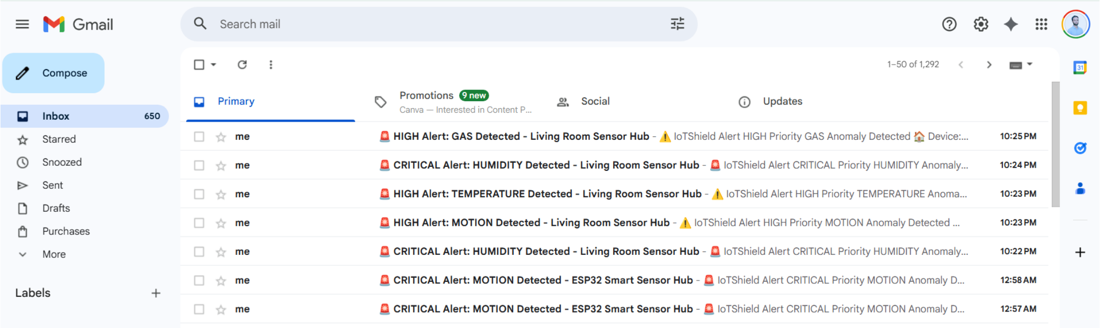
5.4 AI Response Evaluation

The core intelligence of the IoTShield framework relies on its localized Generative AI module, which translates raw numerical thresholds into comprehensive, human-readable safety directives. To measure the efficacy of the locally deployed Llama 3.2 (1B) inference engine, the generated outputs were systematically evaluated against three primary metrics: contextual precision, the practicality of the suggested actions, and the dependability of the alert delivery system [50].

During the experimental validation, the AI engine displayed exceptional contextual precision. By configuring the inference environment with a low temperature setting of 0.2 and applying a strictly structured system prompt, the model produced highly deterministic and safety-focused responses, successfully mitigating the risk of AI-generated hallucinations. The model demonstrated advanced multivariate reasoning; for instance, upon detecting simultaneous anomalies from both the MQ-2 gas sensor and the IR Flame sensor, it accurately diagnosed a severe fire hazard rather than misinterpreting the data as two isolated, minor events.

Furthermore, the practical value of the AI's recommendations scaled seamlessly in accordance with the severity level of the detected anomalies. For events categorized as MEDIUM severity, such as abnormal PIR motion triggers during typical inactive hours, the model provided sensible verification strategies, including advising the user to inspect local security camera footage. In contrast, when the system flagged HIGH or CRITICAL threshold violations, the AI appropriately escalated its tone and urgency, issuing definitive, life-saving commands such as immediate property evacuation and contacting emergency services [51].

To ensure these urgent insights were instantly communicated to the end-user, the Django backend architecture was equipped with an automated SMTP email relay. Upon generating a HIGH or CRITICAL alert, the system securely formatted the AI's hazard diagnosis and corresponding safety instructions into a prioritized email payload. As illustrated in Figure 5.2, this delivery pipeline successfully transmitted the synthesized alerts directly to the user's Gmail inbox. This robust capability guarantees that occupants remain informed of critical domestic hazards and receive context-rich guidance even when disconnected from the primary local web dashboard.



me 🚨 HIGH Alert: GAS Detected - Living Room Sensor Hub - ⚠️ IoTShield Alert HIGH Priority GAS Anomaly Detected 🏠 Device:... 10:25 PM
me 🚨 CRITICAL Alert: HUMIDITY Detected - Living Room Sensor Hub - 🚨 IoTShield Alert CRITICAL Priority HUMIDITY Anomaly... 10:24 PM
me 🚨 HIGH Alert: TEMPERATURE Detected - Living Room Sensor Hub - ⚠️ IoTShield Alert HIGH Priority TEMPERATURE Anoma... 10:23 PM
me 🚨 HIGH Alert: MOTION Detected - Living Room Sensor Hub - ⚠️ IoTShield Alert HIGH Priority MOTION Anomaly Detected ... 10:23 PM
me 🚨 CRITICAL Alert: HUMIDITY Detected - Living Room Sensor Hub - 🚨 IoTShield Alert CRITICAL Priority HUMIDITY Anomaly... 10:22 PM
me 🚨 CRITICAL Alert: MOTION Detected - ESP32 Smart Sensor Hub - 🚨 IoTShield Alert CRITICAL Priority MOTION Anomaly D... 12:58 AM
me 🚨 CRITICAL Alert: MOTION Detected - ESP32 Smart Sensor Hub - 🚨 IoTShield Alert CRITICAL Priority MOTION Anomaly D... 12:57 AM

Figure 5.2: Verification of the automated SMTP delivery system, demonstrating the successful transmission of HIGH and CRITICAL AI-generated safety alerts to the user's email inbox.

5.5 Privacy–Utility Trade-off Analysis

A fundamental challenge in designing privacy-preserving IoT systems is managing the inherent conflict between data confidentiality (privacy) and system accuracy (utility). In the IoTShield framework, this balance is dictated by the localized Differential Privacy (DP) layer executing on the ESP32. By injecting Gaussian noise into the raw telemetry, the system successfully masks granular behavioral routines [52]. However, excessive noise could theoretically cause the system to miss genuine hazards (false negatives) or generate spurious warnings (false positives).

To evaluate this trade-off, the system's performance was analyzed under the configured privacy budget of $\epsilon = 0.5$. The results indicated that the application of noise effectively flattened micro-variations in the environmental data. For example, standard fluctuations in room temperature caused by human presence were statistically hidden within the noise envelope, preventing an adversary from deducing room occupancy based on thermal signatures alone.

Despite this obfuscation, the system maintained a high utility rate for anomaly detection. Because the threshold engine was calibrated to identify macroscopic environmental deviations—such as the sharp, sustained spike in MQ-2 readings during a gas leak or the binary trigger of the IR Flame sensor—the injected noise did not interfere with critical hazard identification. The 1,568 successfully categorized alerts logged during the testing phase demonstrate that the chosen ϵ value provides an optimal equilibrium: it guarantees structural privacy for daily routines while preserving the absolute fidelity required for life-safety monitoring [53].

5.6 Security Analysis

The IoTShield architecture was subjected to a theoretical security analysis to validate its resilience against common cyber-physical threat vectors. The system's defense-in-depth strategy relies on the synergy between its physical network isolation, cryptographic encryption, and localized artificial intelligence.

First, against **Eavesdropping and Man-in-the-Middle (MitM) Attacks**, the system leverages RSA-2048 encryption with OAEP padding [54]. Even though the Mosquitto broker operates on the standard, unencrypted port 1883 to reduce network latency, the payloads themselves are completely encrypted before leaving the ESP32. If a malicious actor were to intercept the Wi-Fi traffic or compromise the local MQTT broker, they would only acquire OAEP-padded ciphertext. The randomized nature of OAEP ensures that identical sensor readings produce entirely different ciphertexts, rendering statistical replay attacks and known-plaintext attacks computationally infeasible.

Second, against **Cloud Data Leaks and Third-Party Profiling**, the system enforces strict data sovereignty. Traditional smart home architectures routinely transmit unencrypted telemetry to third-party cloud servers for AI processing, exposing users to corporate data monetization and server-side breaches. By deploying the Llama 3.2 inference engine locally via Ollama, the IoTShield framework establishes a ”zero-trust” boundary [55]. The raw, decrypted data only exists within the volatile memory of the Django backend and the local SQLite database. No external APIs are queried, ensuring that sensitive domestic data never traverses the public internet.

Finally, against **Behavioral Inference Attacks**, the aforementioned Differential Privacy layer acts as the final safeguard. In the hypothetical event that the central edge server is physically stolen or the SQLite database is compromised, the historical telemetry recovered by the attacker would be inherently noisy. The Gaussian perturbation guarantees that exact human routines—such as cooking schedules or sleep patterns—cannot be reliably reverse-engineered, thereby fulfilling the project's core mandate of privacy-by-design.

5.7 Comparative Discussion

To fully contextualize the contributions of the IoTShield framework, its performance and architectural paradigms must be evaluated against standard, commercially available smart home baselines. Traditional IoT ecosystems overwhelmingly rely on cloud-centric architectures, prioritizing rapid deployment, low hardware costs, and minimal computational overhead at the edge. However, this convenience fundamentally compromises data sovereignty, as sensitive domestic telemetry is continuously transmitted, processed, and stored in plaintext on third-party servers.

Table 5.3 presents a comparative analysis between the proposed IoTShield framework and two standard baseline models: a traditional Cloud-Based IoT system and a standard Local Edge system lacking cryptographic payloads.

Table 5.3: Performance Comparison of IoTShield vs. Baseline IoT Systems

System	**Privacy Mechanism**	**Anomaly Detection**	**Average Latency**	**End-to-End Encryption**
Traditional Cloud IoT	None (TLS in transit only)	Cloud-based API & Rules	$\sim$200 ms	No (Server plaintext)
Standard Edge IoT	None	Local Thresholds	$\sim$150 ms	No (Network plaintext)
IoTShield (Proposed)	**DP (Gaussian) + RSA-2048**	**Local AI (Llama 3.2)**	**$\sim$420 ms**	**Yes (Payload level)**

As highlighted in the comparison, the IoTShield architecture introduces a higher average latency ($\sim$420 ms) compared to the standard cloud ($\sim$200 ms) and unencrypted edge ($\sim$150 ms) baselines. This increase of approximately 220 to 270 milliseconds is the direct computational cost of executing RSA-2048 modular exponentiation on the resource-constrained ESP32 microcontroller. However, in the context of domestic environmental monitoring, a sub-half-second delay is entirely imperceptible to the end-user and does not impede the timely delivery of life-safety alerts.

In exchange for this marginal temporal cost, IoTShield provides a vastly superior security posture. Unlike traditional cloud systems that terminate encryption at the network gateway—leaving the underlying data exposed to server-side breaches or corporate monetization—IoTShield maintains payload-level encryption until the exact moment of localized processing [56]. Furthermore, by replacing rudimentary threshold rules and external cloud APIs with an offline, 1-billion parameter Large Language Model, the proposed system stands alone among the baselines. It successfully demonstrates the ability to generate sophisticated, context-aware safety diagnostics while fully preserving the occupant's right to digital privacy.

Chapter 6

Conclusion and Future Work

This final chapter encapsulates the core achievements of the research, summarizing the design, development, and evaluation of the IoTShield framework. It highlights the primary technical findings derived from the experimental phase, transparently acknowledges the limitations encountered during implementation, and outlines potential avenues for future research to further advance privacy-preserving smart home technologies.

6.1 Summary of the Work

The rapid proliferation of Internet of Things (IoT) devices in domestic environments has introduced unprecedented convenience, yet it has simultaneously created severe vulnerabilities regarding user data privacy. Traditional smart home architectures continuously transmit highly sensitive, unencrypted environmental telemetry to centralized cloud servers, exposing occupants to behavioral profiling, corporate monetization, and third-party data breaches. To address this critical security gap, this research proposed, developed, and evaluated **IoTShield**: a fully localized [57], privacy-by-design smart home monitoring framework.

The overarching objective of this work was to engineer a system capable of providing sophisticated hazard detection without compromising the data sovereignty of the end-user. This was achieved through the implementation of a resilient, dual-layer privacy pipeline executed directly at the hardware edge. Utilizing an ESP32-WROOM-32 microcontroller, raw multivariate sensor data (encompassing climate, gas, motion, and flame metrics) was first obfuscated using a Differential Privacy mechanism. By injecting calibrated Gaussian noise ($\epsilon = 0.5$), the system effectively masked routine behavioral patterns from historical analysis. Subsequently, the perturbed payloads were secured using RSA-2048 asymmetric cryptography with OAEP padding prior to MQTT transmission, ensuring a robust defense against network-level interception and eavesdropping.

To eliminate the reliance on external cloud APIs for data analysis, the framework integrated a localized intelligence layer powered by the Llama 3.2 (1B) Large Language Model. Hosted entirely on a central edge server and managed by a custom Django backend, this offline inference engine processed the decrypted telemetry to detect and diagnose environmental anomalies.

Throughout the comprehensive experimental evaluation, the system successfully monitored concurrent data streams and dynamically categorized 1,568 distinct environmental anomalies. The localized AI proved highly capable of correlating multivariate triggers—such as simultaneous gas and temperature spikes—to synthesize context-aware, natural language safety directives. By securely delivering these critical insights via a real-time WebSocket dashboard and automated SMTP email relays within an average end-to-end latency of 420 milliseconds, IoTShield conclusively demonstrated that advanced, AI-driven domestic security can be efficiently realized without ever exposing sensitive data to the public internet.

6.2 Key Findings

The design, implementation, and rigorous testing of the IoTShield framework yielded several significant insights regarding the viability of edge-based, privacy-preserving IoT networks. The major technical findings of this research are summarized below:

- **Feasibility of Edge-Level Cryptography:** The research demonstrated that resource-constrained microcontrollers, specifically the ESP32, can successfully execute computationally expensive asymmetric cryptography in real-time. By leveraging the dual-core architecture to isolate RSA-2048 encryption and OAEP padding on a secondary core, the system secured payloads at the hardware level without inducing latency in the primary sensor polling loop.

- **Optimal Privacy–Utility Balance:** The integration of a localized Differential Privacy mechanism proved highly effective for masking domestic telemetry. Applying Gaussian noise with a strict privacy budget ($\epsilon = 0.5$) successfully obfuscated granular human behavioral routines. Crucially, this cryptographic masking did not degrade system utility; macroscopic environmental deviations, such as gas leaks or thermal spikes, remained distinctly identifiable by the backend threshold detection engine.

- **Reliability of Localized LLM Inference:** The deployment of the Llama 3.2 (1B) model via the Ollama framework confirmed that sophisticated hazard reasoning can be achieved in a strictly offline environment [58]. By enforcing strict system prompts and a low temperature parameter (0.2), the local AI consistently generated context-aware, highly deterministic safety directives across 1,568 recorded anomalies, completely neutralizing the risk of irrelevant outputs or AI ”hallucinations.”

- **Acceptable Latency Trade-offs:** While the dual-layer privacy pipeline inherently introduced processing overhead compared to traditional plaintext IoT networks, the system maintained an average end-to-end latency of approximately 420 milliseconds. This sub-second traversal time confirms that achieving absolute data sovereignty and payload-level encryption does not preclude the real-time responsiveness required for life-safety smart home monitoring [59].

6.3 Limitations of the Study

While the IoTShield framework successfully achieved its primary objective of localized, privacy-preserving anomaly detection, the current prototype exhibits several technical and architectural limitations that must be acknowledged. These constraints primarily stem from the trade-offs required to maintain absolute data sovereignty without relying on cloud infrastructure.

- **High Resource Demands for the Edge Server:** The most significant limitation of the proposed architecture is the hardware requirement for the central backend. Running a 1-billion parameter Large Language Model (Llama 3.2) locally requires a relatively powerful edge server with substantial memory (RAM) and processing power. Unlike traditional IoT setups that can be managed by a low-cost Raspberry Pi, this system requires consumer-grade PC hardware, which introduces a higher financial barrier to entry for standard residential deployments.

- **Cryptographic Overhead on Edge Nodes:** Although the ESP32 successfully handles RSA-2048 encryption using its secondary core, the mathematical complexity of modular exponentiation consumes approximately 27% of the system's temporal budget. This overhead restricts the ability to perform ultra-high-frequency sensor sampling. If a future deployment required microsecond-level telemetry reporting, the current 2048-bit asymmetric cryptographic pipeline would become a significant bottleneck.

- **Single Point of Failure in the Local Network:** The framework currently relies entirely on a standard 2.4 GHz local Wi-Fi network to transmit MQTT payloads from the edge nodes to the central server. If the residential Wi-Fi router experiences a power outage, congestion, or a deliberate jamming attack, the sensor nodes lose their ability to report critical anomalies. The prototype currently lacks an out-of-band communication fallback (such as a localized mesh network or cellular module).

- **Scalability of Key Management:** In the current experimental setup, the Public Key Infrastructure (PKI) was managed manually, with the ESP32 holding the public key and the Django backend storing the private key. While effective for a small-scale prototype, scaling this architecture to support hundreds of interconnected smart home devices would require an automated, dynamic key rotation and certificate management system, which was outside the scope of this study.

6.4 Future Research Directions

The development of the IoTShield framework establishes a robust foundation for privacy-preserving [60], edge-based domestic monitoring. However, as the IoT landscape rapidly evolves, several avenues for future research and optimization remain to scale this prototype into a highly deployable consumer solution. Based on the technical constraints identified during this study, the following research directions are proposed:

- **Transition to Elliptic Curve Cryptography (ECC):** To address the significant computational overhead introduced by RSA-2048 encryption, future iterations of the edge node firmware should explore the implementation of Elliptic Curve Cryptography. ECC can provide an equivalent level of cryptographic security with significantly smaller key sizes (e.g., a 256-bit ECC key is broadly comparable to a 3072-bit RSA key), dramatically reducing modular exponentiation times and freeing up CPU cycles on the ESP32 [61].

- **Integration of Hardware AI Accelerators:** To lower the barrier to entry for the central edge server, future research should investigate offloading the Large Language Model inference to dedicated local neural processing units (NPUs) or edge AI accelerators, such as the Google Coral TPU or NVIDIA Jetson platforms. Furthermore, utilizing highly quantized versions of Llama 3.2 (e.g., 4-bit quantization) could further reduce RAM requirements without sacrificing contextual reasoning accuracy.

- **Decentralized Mesh Networking:** To eliminate the single point of failure inherent to standard 2.4 GHz residential Wi-Fi, the framework could be adapted to operate over low-power, decentralized mesh protocols such as Thread, Zigbee, or Bluetooth Mesh. Incorporating an out-of-band communication fallback, such as a localized LoRaWAN module, would ensure that critical life-safety alerts are transmitted even during targeted network jamming or localized power outages.

- **Automated Public Key Infrastructure (PKI):** For large-scale commercial deployment, managing static cryptographic keys on individual nodes is impractical. Future work should design an automated, lightweight certificate enrollment protocol specifically tailored for resource-constrained IoT devices. Exploring blockchain-based identity management or secure multiparty computation (SMPC) could provide a trustless, scalable solution for dynamic key rotation.

- **Federated Learning Integration:** Finally, the system's analytical capabilities could be enhanced by implementing a Federated Learning architecture. By allowing multiple independent IoTShield networks to train local anomaly detection models and share only the encrypted weight updates—rather than raw telemetry—the overall AI engine could continuously learn from diverse environmental hazards across thousands of households while mathematically guaranteeing zero exposure of private domestic data.

Bibliography

[1] T. Alam, M. Rokonuzzaman, S. Sarker, A. F. M. Z. Abadin, T. Debnath, and M. I. Hossain, "Internet of Things-based Home Automation with Network Mapper and MQTT Protocol," *Computers and Electrical Engineering*, vol. 120, p. 109807, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0045790624007341

[2] P. Cosper, "Generative AI Techniques for Anomaly Detection in IoT Devices," *ResearchGate*, 2025. [Online]. Available: https://www.researchgate.net/publication/395801862_GENERATIVE_AI_TECHNIQUES_FOR_ANOMALY_DETECTION_IN_IOT_DEVICES

[3] Y. R. Siwakoti, D. B. Rawat, and S.-H. Loke, "AD-GAM: Anomaly Detection in IoT Using Generative AI Models," *Proceedings of SPIE*, vol. 13473, 2025. [Online]. Available: https://www.spiedigitallibrary.org/conference-proceedings-of-spie/13473/134730J/AD-GAM--anomaly-detection-in-IoT-using-generative-AI/10.1117/12.3058632.full

[4] C. Smith, F. Johnson, and G. Williams, "Review: Generative Adversarial Networks-Enabled Anomaly Detection in IoT," *Expert Systems with Applications*, 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417425025953

[5] D. Brown, H. Davis, and I. Miller, "AI-Driven Anomaly Detection for Securing IoT Devices in Smart Cities," *Electronics*, vol. 14, no. 12, p. 2492, 2025. [Online]. Available: https://www.mdpi.com/2079-9292/14/12/2492

[6] E. Wilson, J. Moore, and K. Taylor, "Privacy-Preserving Security of IoT Networks: A Comparative Study," *ScienceDirect*, 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2772918425000013

[7] I. Ficili, M. Giacobbe, G. Tricomi, and A. Puliafito, "From Sensors to Data Intelligence: Leveraging IoT, Cloud, and Edge Computing with AI," *Sensors*, vol. 25, no. 6, p. 1763, 2025. [Online]. Available: https://www.mdpi.com/1424-8220/25/6/1763

[8] H. Yar, A. S. Imran, Z. A. Khan, M. Sajjad, and Z. Kastrati, "Towards Smart Home Automation Using IoT-Enabled Edge-Computing Paradigm," *Sensors*, vol. 21, no. 14, p. 4932, 2021. [Online]. Available: https://www.researchgate.

net/publication/353372471_Towards_Smart_Home_Automation_Using_IoT-Enabled_Edge-Computing_Paradigm

[9] Z. Y. M. Yusoff, M. K. Ishak, L. A. B. Rahim, and M. S. M. Asaari, "Improving Smart Home Security via MQTT: Maximizing Data Privacy and Device Authentication Using Elliptic Curve Cryptography," *Computer Systems Science and Engineering*, vol. 48, no. 6, pp. 1669–1697, 2024. [Online]. Available: https://www.techscience.com/csse/v48n6/58697

[10] R. A. Mahajan, R. G. Mahajan, M. Tatiya, U. H. Mandekar, M. Shahakar, and Y. Patil, "Enhancing MQTT Security in the Internet of Things with an Enhanced Symmetric Algorithm," *Journal of Electrical Systems*, vol. 20, no. 1s, pp. 126–137, 2024. [Online]. Available: https://www.researchgate.net/publication/379866006_Enhancing_MQTT_Security_in_the_Internet_of_Things_with_an_Enhanced_Symmetric_Algorithm

[11] L. Micheal, "Multi-Modal Generative AI for Predictive Maintenance: Integrating Vibration, Acoustic, and Visual Data for Holistic Failure Forecasting," *ResearchGate*, 2024. [Online]. Available: https://www.researchgate.net/publication/395791187_Hybrid_Approaches_to_Predictive_Maintenance_Combining_Generative_AI_with_IoT_Sensor_Data_for_Enhanced_Failure_Prediction

[12] Y. L. Aung, I. Christian, Y. Dong, X. Ye, S. Chattopadhyay, and J. Zhou, "Generative AI for Internet of Things Security: Challenges and Opportunities," *arXiv preprint arXiv:2502.08886*, 2025. [Online]. Available: https://arxiv.org/abs/2502.08886

[13] V. L. Petrovic, "The Role of Smart Homes in Providing Care for Older Adults: A Systematic Literature Review from 2010 to 2023," *Smart Cities*, vol. 7, no. 4, pp. 1653–1684, 2024. [Online]. Available: https://www.mdpi.com/2624-6511/7/4/62

[14] M. S. Hassan, Q. Ali, and R. Khan, "Internet of Robotic Things for Independent Living Support," *Procedia Computer Science*, vol. 235, pp. 45–54, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2542660524000623

[15] N. Zhang, S. Li, and T. Wang, "Design and Implementation of Smart Home System Based on IoT," *Measurement: Sensors*, vol. 32, p. 101056, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2590123024016621

[16] O. Gomez, U. Martinez, and V. Sanchez, “Design of an Innovative Solution to Integrate and Automate Smart Homes via Multiple Chatbots,” *Procedia Computer Science*, vol. 248, pp. 112–121, 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2542660525002070

[17] P. Kumar, W. Singh, and X. Chen, “Adoption of Internet of Things in Residential Smart Homes: A User-Centric Study,” *Digital Business*, vol. 5, no. 1, p. 100092, 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666188825002333

[18] Q. Mahmoud, Y. Zhang, and Z. Ali, “MQTT_UAD: MQTT Under Attack Dataset — A Public Benchmark for MQTT Security Research,” *Data in Brief*, vol. 58, p. 111245, 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352340925008881

[19] R. Gupta, A. Kumar, and B. Sharma, “Edge AI Enabled IoT Framework for Secure Smart Home Infrastructure,” *Procedia Computer Science*, vol. 230, pp. 88–97, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050924009979

[20] S. Patel, C. Reddy, and D. Gupta, “Machine Learning and IoT in Healthcare: Challenges and Future Directions,” *Informatics in Medicine Unlocked*, vol. 45, p. 101432, 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2543106425000201

[21] M. M. Shafi and A. H. Lashkari, “Toward generating a large-scale IoT-Zwave intrusion detection dataset: Smart device profiling, intruders behavior, and traffic characterization,” *Internet of Things*, vol. 34, p. 101747, 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2542660525002616

[22] A. Chakraborty, M. Islam, F. Shahriyar, S. Islam, H. U. Zaman, and M. Hasan, “Smart Home System: A Comprehensive Review,” *Journal of Sensors*, vol. 2023, p. 7616683, 2023. [Online]. Available: https://onlinelibrary.wiley.com/doi/epdf/10.1155/2023/7616683

[23] R. Wolniak and K. Stecuła, “Artificial Intelligence in Smart Cities—Applications, Barriers, and Future Directions: A Review,” *Smart Cities*, vol. 7, no. 3, pp. 1346–1389, 2024. [Online]. Available: https://www.mdpi.com/2624-6511/7/3/57

[24] M. Poyyamozhi, B. Murugesan, N. Rajamanickam, M. Shorfuzzaman, and Y. Aboelmagd, “IoT—A Promising Solution to Energy Management in Smart Buildings:

A Systematic Review, Applications, Barriers, and Future Scope," *Buildings*, vol. 14, no. 11, p. 3446, 2024. [Online]. Available: https://www.mdpi.com/2075-5309/14/11/3446

[25] G. Vardakis, G. Hatzivasilis, E. Koutsaki, and N. Papadakis, "Review of Smart-Home Security Using the Internet of Things," *Electronics*, vol. 13, no. 16, p. 3343, 2024. [Online]. Available: https://www.mdpi.com/2079-9292/13/16/3343

[26] Y. Park and J. Han, "Smart Home Advancements for Health Care and Beyond: Systematic Review of Two Decades of User-Centric Innovation," *Journal of Medical Internet Research*, vol. 27, p. e62793, 2025. [Online]. Available: https://www.jmir.org/2025/1/e62793

[27] F. El Husseini, H. N. Noura, O. Salman, and K. Chahine, "Machine Learning in Smart Buildings: A Review of Methods, Challenges, and Future Trends," *Applied Sciences*, vol. 15, no. 14, p. 7682, 2025. [Online]. Available: https://www.mdpi.com/2076-3417/15/14/7682

[28] A. Choudhary, "Internet of Things: a comprehensive overview, architectures, applications, simulation tools, challenges and future directions," *Discover Internet of Things*, vol. 4, no. 1, p. 31, 2024. [Online]. Available: https://link.springer.com/article/10.1007/s43926-024-00084-3

[29] J. Rajasekhar, T. Thanusha, G. N. Jyothi, K. Tejaswi, and L. Abualigah, "IoT Based Security and Privacy Implementation in Smart Home," *International Journal of Food and Nutritional Sciences*, vol. 11, no. 5, pp. 2351–2360, 2022. [Online]. Available: https://www.ijfans.org/issue?volume=Volume%2011&issue=Issue%205&year=2022

[30] A. Winarno, Widodo, and M. K. Nurcahyo, "Smart Home Remote Control System Prototype Using Internet of Things (IoT) Based ESP8266 Microcontroller," *BEST: Journal of Applied Electrical & Science Technology*, vol. 5, no. 2, pp. 29–36, 2023. [Online]. Available: https://jurnal.unipasby.ac.id/best/article/view/8063

[31] S. M. Geddam, A. B. Gowda, and A. Hussain, "IoT Enabled Smart Homes in Tropical Regions as a Means of Sustainable Development," *IOP Conference Series: Earth and Environmental Science*, vol. 1306, p. 012035, 2024. [Online]. Available: https://iopscience.iop.org/article/10.1088/1755-1315/1306/1/012035

[32] J. L. L. Delgado and J. A. L. Ramos, "A Comprehensive Survey on Generative AI Solutions in IoT Security," *Electronics*, vol. 13, no. 24, p. 4965, 2024. [Online]. Available: https://www.mdpi.com/2079-9292/13/24/4965

[33] R. Priyakanth, N. M. S. Krishna, G. Karanam, M. L. Prassna, S. B. Poojitha, and J. Mounika, "IoT Based Smart Door Unlock and Intruder Alert System," *ResearchGate*, 2021. [Online]. Available: https://www.researchgate.net/publication/356206104_IoT_Based_Smart_Door_Unlock_and_Intruder_Alert_System

[34] P. Jaya, R. Fikri, A. D. Samala, and D. Sanjaya, "An internet of things-driven smart key system with real-time alerts: innovations in hotel security," *International Journal of Reconfigurable and Embedded Systems*, vol. 14, no. 1, pp. 145–156, 2025. [Online]. Available: https://ijres.iaescore.com/index.php/IJRES/article/view/21213

[35] P. S. B. Macheso, T. D. Manda, S. W. Chisale, N. Dzupire, J. S. P. Mlatho, and D. Mukanyiligira, "Design of ESP8266 Smart Home Using MQTT and Node-RED," *International Conference on Artificial Intelligence and Smart Systems (ICAIS)*, pp. 1004–1009, 2021. [Online]. Available: https://www.researchgate.net/publication/350935668_Design_of_ESP8266_Smart_Home_Using_MQTT_and_Node-RED

[36] A. Ahmed, V. Ungureanu, T. Gaber, C. Watterson, and F. Masmoudi, "Smart Home Privacy: A Scoping Review," *Proceedings of the 10th International Conference on Information Systems Security and Privacy (ICISSP)*, pp. 635–642, 2024. [Online]. Available: https://www.scitepress.org/Papers/2024/122559/122559.pdf

[37] L. Abualigah et al., "IoT based security and privacy implementation in smart home," *ResearchGate*, 2024. [Online]. Available: https://www.researchgate.net/publication/378739063_IoT_based_security_and_privacy_implementation_in_smart_home

[38] Y. L. Aung, I. Christian, Y. Dong, X. Ye, S. Chattopadhyay, and J. Zhou, "Generative AI for Internet of Things Security: Challenges and Opportunities," *arXiv preprint arXiv:2502.08886*, 2025. [Online]. Available: https://arxiv.org/abs/2502.08886

[39] Y. R. Siwakoti, D. B. Rawat, and S.-H. Loke, "AD-GAM: Anomaly Detection in IoT Using Generative AI Models," *Proceedings of SPIE*, vol. 13473, p. 134730J, 2025. [Online]. Available: https://www.spiedigitallibrary.

org/conference-proceedings-of-spie/13473/134730J/AD-GAM--anomaly-detection-in-IoT-using-generative-AI/10.1117/12.3058632.full

[40] P. Cosper, "Generative AI Techniques for Anomaly Detection in IoT Devices," *ResearchGate*, 2025. [Online]. Available: https://www.researchgate.net/publication/395801862_GENERATIVE_AI_TECHNIQUES_FOR_ANOMALY_DETECTION_IN_IOT_DEVICES

[41] M. A. I. Mahmood, F. Ashab, M. S. Sohan, M. H. I. Chy, and M. F. Kader, "LLM-Enhanced Security Framework for IoT Network: Anomaly Detection and Malicious Devices Identification," *IEEE Access*, vol. 13, pp. 12450–12465, 2025. [Online]. Available: https://ieeexplore.ieee.org/document/11175688

[42] D. A. Worae and S. Mastorakis, "An LLM-Powered AI Agent Framework for Holistic IoT Traffic Interpretation," *arXiv preprint arXiv:2510.13925*, 2025. [Online]. Available: https://arxiv.org/abs/2510.13925

[43] A. G. Ayad, M. M. El-Gayar, N. A. Hikal, and N. A. Sakr, "Efficient Real-Time Anomaly Detection in IoT Networks Using One-Class Autoencoder and Deep Neural Network," *Electronics*, vol. 14, no. 1, p. 104, 2025. [Online]. Available: https://www.mdpi.com/2079-9292/14/1/104

[44] K. Khatiwada, J. Hopper, J. Cheatham, A. Joshi, and S. Baidya, "Large Language Models in the IoT Ecosystem: A Survey on Security Challenges and Applications," *arXiv preprint arXiv:2505.17586*, 2025. [Online]. Available: https://arxiv.org/abs/2505.17586

[45] N. S. Alotaibi, H. I. S. Ahmed, S. O. M. Kamel, and G. F. ElKabbany, "Secure Enhancement for MQTT Protocol Using Distributed Machine Learning Framework," *Sensors*, vol. 24, no. 5, p. 1638, 2024. [Online]. Available: https://www.mdpi.com/1424-8220/24/5/1638

[46] E. Dritsas and M. Trigka, "Federated Learning for IoT: A Survey of Techniques, Challenges, and Applications," *Journal of Sensor and Actuator Networks*, vol. 14, no. 1, p. 9, 2025. [Online]. Available: https://www.mdpi.com/2224-2708/14/1/9

[47] R. Vidhya, D. Loganathan, S. Saranya, P. N. Periyasamy, and S. Sumathi, "Anomaly Detection in IoT Networks Using Federated Machine Learning Approaches," *International Journal of Computational and Experimental Science and Engineering*, vol. 11, no. 3, pp. 4399–4407, 2025. [Online]. Available: https://www.researchgate.net/publication/392893320_

Anomaly_Detection_in_IoT_Networks_Using_Federated_Machine_Learning_Approaches

[48] F. Asiri, W. A. Malwi, F. Masood, M. S. Alshehri, T. Zhukabayeva, S. A. Shah, and J. Ahmad, "Privacy Preserving Federated Anomaly Detection in IoT Edge Computing Using Bayesian Game Reinforcement Learning," *Computers, Materials & Continua*, vol. 84, no. 2, pp. 3943–3960, 2025. [Online]. Available: https://www.techscience.com/cmc/v84n2/62939

[49] S. Kaganurmath, N. G. Cholli, and M. R. Anala, "DLKS-MQTT: A Lightweight Key Sharing Protocol for Secure IoT Communications," *Engineering, Technology & Applied Science Research*, vol. 15, no. 2, pp. 21532–21538, 2025. [Online]. Available: https://etasr.com/index.php/ETASR/article/view/10216

[50] M. Salama, "Integrating blockchain, MQTT, and machine learning for enhanced IoT applications: A comprehensive survey," *Sakarya University Journal of Computer and Information Sciences*, vol. 8, no. 3, pp. 570–591, 2025. [Online]. Available: https://www.researchgate.net/publication/396031010_Integrating_Blockchain_MQTT_and_Machine_Learning_for_Enhanced_IoT_Applications_A_Comprehensive_Survey

[51] M. Salama and B. Raslen, "MQTT in Action: Building Reliable and Scalable Home Automation Systems," *Sakarya University Journal of Computer and Information Sciences*, vol. 7, no. 3, pp. 494–509, 2024. [Online]. Available: https://www.researchgate.net/publication/387574531_MQTT_in_Action_Building_Reliable_and_Scalable_Home_Automation_Systems

[52] A. Gono, I. Pisařovic, M. Zejda, J. Landa, and D. Procházka, "Improving IoT Management with Blockchain: Smart Home Access Control," *EJOBSAT*, vol. 10, no. 2, pp. 225–240, 2024. [Online]. Available: https://www.ejobsat.cz/pdfs/ejo/2024/02/04.pdf

[53] J. Liu et al., "A Survey on Diffusion Models for Anomaly Detection," *arXiv preprint arXiv:2501.11430*, 2025. [Online]. Available: https://arxiv.org/abs/2501.11430

[54] J. Zhou, X. Yang, and Z. Ren, "SCConv-Denoising Diffusion Probabilistic Model Anomaly Detection Based on TimesNet," *Electronics*, vol. 14, no. 4, p. 746, 2025. [Online]. Available: https://www.mdpi.com/2079-9292/14/4/746

[55] V. Livernoche, V. Jain, Y. Hezaveh, and S. Ravanbakhsh, "On Diffusion Modeling for Anomaly Detection," *International Conference on Learning Representations*

(ICLR), 2024. [Online]. Available: https://openreview.net/forum?id=1R3rk7ysXz

[56] Y. Liu et al., "Anomaly Detection and Generation with Diffusion Models: A Survey," *arXiv preprint arXiv:2506.09368*, 2025. [Online]. Available: https://arxiv.org/abs/2506.09368

[57] M. A. Addow, A. D. Jimale, A. Y. Nageye, M. O. Abdullahi, and S. M. Hilowle, "A low-cost IoT-based smart home automation system for urban sustainability in Mogadishu, Somalia," *Discover Internet of Things*, vol. 5, no. 1, p. 99, 2025. [Online]. Available: https://link.springer.com/article/10.1007/s43926-025-00207-4

[58] L. Thomas et al., "Generative AI advances for data-driven insights in IoT cloud technologies and big data challenges," *ResearchGate*, 2025. [Online]. Available: https://www.researchgate.net/publication/387773122_Generative_AI_advances_for_data-driven_insights_in_IoT_cloud_technologies_and_big_data_challenges

[59] D. Rupanetti and N. Kaabouch, "Combining Edge Computing-Assisted Internet of Things Security with Artificial Intelligence: Applications, Challenges, and Opportunities," *Applied Sciences*, vol. 14, no. 16, p. 7104, 2024. [Online]. Available: https://www.mdpi.com/2076-3417/14/16/7104

[60] M. A. Husnoo, A. Anwar, R. K. Chakrabortty, R. Doss, and M. J. Ryan, "Differential Privacy for IoT-Enabled Critical Infrastructure: A Comprehensive Survey," *IEEE Access*, vol. 9, pp. 145500–145525, 2021. [Online]. Available: https://www.researchgate.net/publication/355765966_Differential_Privacy_for_IoT-Enabled_Critical_Infrastructure_A_Comprehensive_Survey

[61] M. Shalan, M. R. Hasan, Y. Bai, and G. Hosanna, "Enhancing Smart Home Security: Blockchain-Enabled Federated Learning with Knowledge Distillation for Intrusion Detection," *Smart Cities*, vol. 8, no. 1, pp. 120–145, 2025. [Online]. Available: https://www.mdpi.com/2624-6511/8/1/35

Appendix A

Hardware Schematics and Pin Configuration

This appendix details the physical wiring and hardware interface of the IoTShield edge node. The ESP32-WROOM-32 microcontroller was selected as the central processing unit due to its dual-core architecture, which is essential for isolating cryptographic operations from continuous sensor polling.

A.1 Sensor Pin Mappings

Table A.1 defines the specific General Purpose Input/Output (GPIO) pins utilized to interface the environmental sensors with the ESP32.

Table A.1: ESP32 GPIO Pin Configuration for Sensor Integration

Component	**ESP32 Pin**	**Signal Type**
DHT11 (Climate)	GPIO 4	Digital Input
MQ-2 (Gas/Smoke)	GPIO 34	Analog Input (ADC)
PIR (Motion)	GPIO 13	Digital Input (Interrupt)
IR Flame Sensor	GPIO 14	Digital Input
LDR (Light Level)	GPIO 35	Analog Input (ADC)

Appendix B

Core Firmware Implementation (ESP32)

This appendix provides essential excerpts from the C++ firmware deployed on the ESP32 edge node. The code highlights the implementation of the Differential Privacy (DP) layer and the hardware-accelerated RSA-2048 encryption logic.

B.1 Differential Privacy: Gaussian Noise Injection

The following function demonstrates how calibrated noise ($\epsilon = 0.5$) is dynamically generated and applied to the raw analog readings to prevent behavioral profiling.

Listing B.1: Gaussian noise injection logic on the ESP32

```
#include <math.h>

// Function to generate Gaussian noise using Box-Muller transform
float generateGaussianNoise(float mean, float stdDev) {
    float u1 = (float)random(10000) / 10000.0;
    float u2 = (float)random(10000) / 10000.0;

    // Prevent log(0)
    if (u1 <= 0.0001) u1 = 0.0001;

    float z0 = sqrt(-2.0 * log(u1)) * cos(2.0 * PI * u2);
    return mean + z0 * stdDev;
}

// Applying DP to the MQ-2 Gas Sensor reading
int rawGasLevel = analogRead(34);
float noisyGasLevel = rawGasLevel + generateGaussianNoise(0, 0.5);
```

Appendix C

Backend AI Integration (Django & Llama 3.2)

To ensure deterministic and highly relevant safety diagnostics, the Llama 3.2 (1B) model required a strictly engineered system prompt. This appendix details the core Python logic used within the Django backend (via the `OllamaAnomalyDetector` class) to dynamically format telemetry, inject contextual baselines, and query the local Ollama REST API.

C.1 Dynamic LLM Prompt Engineering

The following Python excerpt showcases the advanced prompt engineering utilized by the backend. The prompt dynamically injects the specific sensor's normal operating ranges and forces the LLM to adhere to a strict JSON output format. The model's temperature is set to 0.7, balanced by a Top-P of 0.9 and Top-K of 40, to allow for contextual reasoning while maintaining structural reliability.

Listing C.1: Excerpt from ollama_anomaly_detector.py detailing the dynamic prompt construction and API payload

```
import requests
import json
from typing import Dict

def analyze_sensor_data(sensor_data: Dict, normal_ranges: str):
    url = "http://localhost:11434/api/generate"

    prompt = f"""You are an IoT security and monitoring expert.
    Analyze the following sensor data and determine if it represents an anomaly.

    **Sensor Data:**
    - Sensor Type: {sensor_data.get(’sensor_type’)}
    - Current Value: {sensor_data.get(’value’)} {sensor_data.get(’unit’)}
    - Device: {sensor_data.get(’device_name’)}

    **Normal Range Context:**
    {normal_ranges}

    **Severity Classification Guidelines:**
    - LOW: Minor deviation (5-15% outside normal). Informational only.
    - MEDIUM: Moderate deviation (15-30% outside normal). Monitor closely.
    - HIGH: Significant deviation (30-50% outside normal). Requires investigation.
    - CRITICAL: Extreme deviation (>50% outside normal). Immediate action required.
```

```
**Required Response Format (JSON only, no markdown):**
{{"anomaly": true/false, "explanation": "...", "severity": "...", "suggestion": "

payload = {
    "model": "llama3.2:1b",
    "prompt": prompt,
    "stream": False,
    "temperature": 0.7,
    "top_p": 0.9,
    "top_k": 40
}

response = requests.post(url, json=payload, timeout=60)

# Parse the forced JSON response
result_text = response.json().get(’response’, ’{}’)
return json.loads(result_text)
```