VIETNAM NATIONAL UNIVERSITY, HANOI

VIETNAM JAPAN UNIVERSITY

BACHELOR PROGRAM

COMPUTER SCIENCE AND ENGINEERING

INTERNSHIP REPORT

DEVELOPMENT OF A COMPREHENSIVE REAL-TIME INFANT MONITORING SYSTEM

Internship Unit: NTQ Solutions JSC (NTEX)

Supervisor: Mr. Le Van Luc

Student Name: Nguyen The Nam

Student ID: 22110158

Hanoi, 2025

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to the Board of Directors and the staff of NTQ Solutions Joint Stock Company, particularly the NTEX Division, for providing a professional and stimulating working environment. The modern facilities and supportive culture significantly contributed to the success of this internship.

I am deeply indebted to my direct supervisor, Mr. Le Van Luc. His expert guidance on embedded systems, cloud architecture, and AI deployment was instrumental in my professional development. His mentorship extended beyond technical skills, offering valuable insights into project management and problem-solving strategies that were crucial for the completion of the "Baby Cry Monitor v2.2" project.

I also wish to thank the faculty members of the School of Computer Science and Engineering at Vietnam Japan University (VNU). The rigorous academic foundation in algorithms, software engineering, and artificial intelligence provided by the university was essential for addressing the complex challenges encountered during this project.

Finally, I am grateful to my family and friends for their unwavering encouragement and support throughout my academic journey.

Despite my best efforts, this report may contain limitations due to time constraints and experience. I welcome any constructive feedback from lecturers and industry professionals to further refine my skills.

LIST OF ABBREVIATIONS

Abbreviation	Definition
AI	Artificial Intelligence
API	Application Programming Interface
CNN	Convolutional Neural Network
IoT	Internet of Things
JWT	JSON Web Token
MQTT	Message Queuing Telemetry Transport
REST	Representational State Transfer
STFT	Short-Time Fourier Transform
YOLO	You Only Look Once
R&D	Research and Development
UI/UX	User Interface / User Experience

Contents

List of Tables

List of Figures

1. OVERVIEW OF THE INTERNSHIP UNIT

1.1. Introduction to NTQ Solutions

Established in 2011, NTQ Solutions Joint Stock Company (NTQ) has emerged as a premier technology provider in Vietnam, delivering software solutions and IT services to a global clientele. Headquartered in Hanoi, with strategic branches in Japan, South Korea, and Hong Kong, NTQ has solidified its standing in the international technology market.

The company's mission is to provide "World-class" technology solutions that address complex business challenges while fostering a positive and productive work environment for its employees.

Within the organizational structure, NTEX (NTQ Technology Extension) operates as a specialized strategic unit focused on the Research and Development (R&D) of emerging core technologies. NTEX serves as an incubator for innovation, prioritizing advancements in the Internet of Things (IoT), Artificial Intelligence (AI), Robotics, and Blockchain to create high-tech "Make in Vietnam" products.

1.2. Organizational Structure and Work Environment

NTQ Solutions adopts a flat organizational model designed to facilitate open communication and collaboration across all levels. My internship was conducted within the R&D Department of the NTEX division.

The R&D team comprises experienced specialists in embedded systems and artificial intelligence. The work environment is characterized by its dynamism and professionalism, strictly adhering to international software development standards such as Agile/Scrum and CMMi. Knowledge sharing is a core value, exemplified by weekly "Tech Talk" sessions where team members exchange insights on recent technological advancements.

1.3. Key Activities of the R&D Department

The R&D department concentrates its efforts on three primary domains:

- **Edge AI Optimization:** Developing and optimizing deep learning models for deployment on resource-constrained hardware, including Raspberry Pi, NVIDIA Jetson, and ESP32 microcontrollers.
- **Smart Ecosystems:** Designing intelligent solutions for healthcare and daily living, such as Elderly Monitoring Systems, Baby Monitors, and Smart Home Hubs.
- **Computer Vision Applications:** Implementing advanced computer vision techniques for facial recognition, behavioral analysis, and automated industrial inspection.

2. THEORETICAL BACKGROUND

The successful implementation of the "Baby Cry Monitor v2.2" requires a multidisciplinary approach, integrating digital signal processing, deep learning, and modern software architecture. This chapter outlines the theoretical foundations governing the system's design, from acoustic analysis to cloud-based infrastructure.

2.1. Digital Audio Signal Processing

2.1.1. Characteristics of Audio Signals

Sound propagates as a mechanical wave. In digital systems, continuous analog signals are converted into discrete data through Sampling and Quantization.

- **Sampling Rate (f_s):** The Nyquist-Shannon sampling theorem dictates that to reconstruct a signal accurately, the sampling rate must be at least twice the maximum frequency component (f_{max}) of the signal ($f_s \geq 2 \cdot f_{max}$). This project utilizes a sampling rate of 16kHz, sufficient to capture the frequency range of infant vocalizations (typically 400Hz to 4000Hz).
- **Quantization:** This process maps infinite amplitude values to a finite set of discrete levels. A 16-bit depth is employed to ensure a high dynamic range.

2.1.2. Short-Time Fourier Transform (STFT)

To analyze non-stationary signals like human speech, the signal must be transformed from the Time Domain to the Time-Frequency Domain.

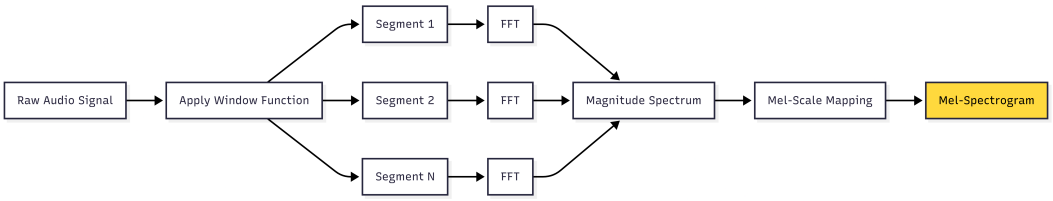

Figure 1: STFT Windowing and Transformation Process

The Short-Time Fourier Transform (STFT) achieves this by segmenting the signal into overlapping windows and applying the Discrete Fourier Transform (DFT) to each segment:

$$X(m,\omega) = \sum_{n=-\infty}^{\infty} x[n]w[n-m]e^{-j\omega n} \tag{1}$$

where $x[n]$ is the input signal and $w[n]$ is the window function. This transformation preserves temporal localization, allowing the system to detect the specific onset of a cry.

2.1.3. Mel-Spectrogram Generation

The power spectrum obtained from STFT is mapped to the Mel Scale, a perceptual scale of pitches judged by listeners to be equal in distance from one another. The mapping is defined as:

$$M(f) = 2595\log_{10}\left(1+\frac{f}{700}\right) \tag{2}$$

The resulting Mel-Spectrogram provides a visual representation of the audio that emphasizes lower frequencies (400–600 Hz), where the fundamental frequency (F_0) of an infant cry resides. This 2D representation serves as the input for the computer vision model.

2.2. Deep Learning for Object Detection

2.2.1. Convolutional Neural Networks (CNN)

CNNs are deep learning architectures optimized for processing grid-like data. They employ convolutional layers to extract hierarchical features, pooling layers to reduce dimensionality, and fully connected layers for classification. In this project, CNNs are used to analyze the visual patterns within Mel-Spectrograms.

2.2.2. YOLO (You Only Look Once)

YOLO is a real-time object detection framework that treats detection as a single regression problem. Unlike region-based methods, YOLO processes the entire image in a single forward pass, predicting bounding boxes and class probabilities simultaneously.

$$\mathscr{L} = \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] + \dots \tag{3}$$

By applying YOLO to spectrograms, the system can detect "cry objects"—distinct harmonic structures—regardless of their temporal position, offering robustness against background noise.

2.3. IoT Communication Protocols

2.3.1. MQTT (Message Queuing Telemetry Transport)

MQTT is a lightweight, publish-subscribe protocol designed for low-bandwidth, high-latency networks. It decouples the data producer (Edge Device) from the consumer (Mobile App) via a central Broker.

- **QoS 1 (At Least Once):** The system utilizes Quality of Service level 1 to ensure that critical alerts (e.g., crying detected) are delivered even in unstable network conditions.

- **Topic Hierarchy:** Data is organized into topics (e.g., `vju/baby_monitor/alert`) allowing for precise message routing.

2.4. Mobile Application Frameworks

2.4.1. React Native and Expo

React Native is a cross-platform framework that allows for the development of native mobile applications using JavaScript and React. It employs a "bridge" architecture to invoke native rendering APIs, ensuring high performance.

- **Expo SDK:** A set of tools and services built around React Native that simplifies the development process. It provides access to native device capabilities (Camera, Microphone, Haptics) through a unified JavaScript API, which is crucial for features like vibration alerts and audio playback.

2.5. Backend Architecture and Databases

2.5.1. FastAPI

FastAPI is a modern, high-performance web framework for building APIs with Python 3.7+. It is built on top of Starlette for the web parts and Pydantic for the data parts. Its asynchronous capabilities are essential for handling concurrent WebSocket connections and real-time data streams from multiple IoT devices.

2.5.2. Time-Series Databases (TimescaleDB)

IoT systems generate massive amounts of temporal data. Traditional relational databases often struggle with the ingestion rate and query performance required for time-series data. **TimescaleDB** is an open-source

database invented for time-series data, implemented as an extension of PostgreSQL. It utilizes "hypertables"—an abstraction layer that automatically partitions data across time and space—to maintain high insert rates and fast queries for analytics (e.g., calculating average temperature over the last 24 hours).

3. INTERNSHIP CONTENT AND IMPLEMENTATION

3.1. Tasks Assigned

During the internship, I was responsible for the full-stack development of the "Baby Cry Monitor v2.2" prototype. My responsibilities encompassed the entire IoT pipeline, from edge computing to cloud infrastructure and user interface:

1. **Edge Computing:** Researching hardware (Raspberry Pi) and optimizing the audio processing pipeline for real-time performance.
2. **Backend Development:** Designing a microservice-based backend using FastAPI to handle data ingestion, authentication, and persistent storage.
3. **Database Design:** Implementing TimescaleDB for efficient storage and retrieval of time-series sensor data.
4. **Mobile Application:** Developing a cross-platform mobile app (React Native) for real-time monitoring and alerts.
5. **System Integration:** Establishing secure communication protocols (MQTT, WebSockets) between the device, server, and application.

3.2. System Architecture

The system follows a three-tier IoT architecture:

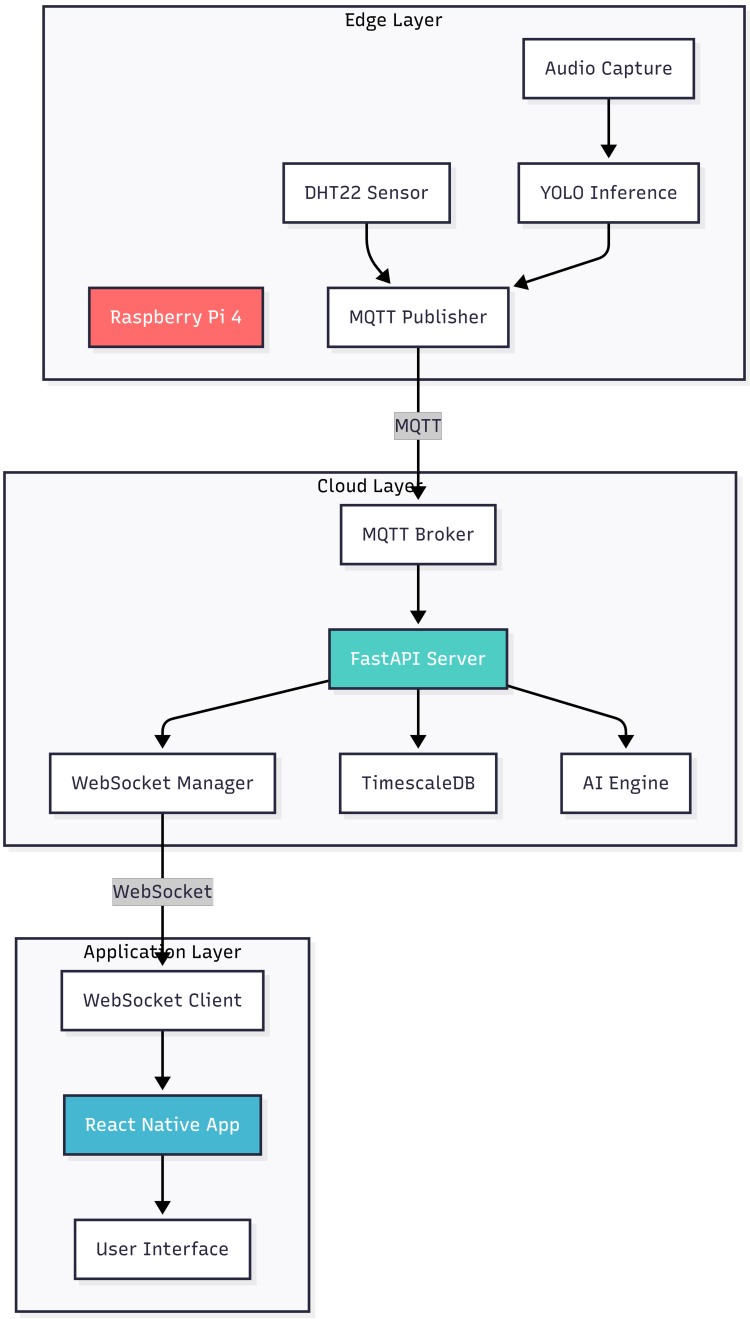

Figure 2: Three-Tier IoT Architecture of Baby Cry Monitor v2.2

- **Edge Layer:** The Raspberry Pi 4 captures audio and environmental data. It performs initial signal processing and communicates with the cloud via MQTT.
- **Cloud Layer:** A FastAPI server acts as the central coordinator. It ingests data from MQTT, stores it in TimescaleDB, and manages WebSocket connections for real-time client updates.

- **Application Layer:** A React Native mobile application serves as the user interface, providing dashboards, historical charts, and critical alerts.

3.3. Edge Device Implementation

The hardware foundation is the Raspberry Pi 4 Model B, chosen for its processing capability. The software is written in Python 3.9 using a multi-threaded architecture to ensure non-blocking audio acquisition.

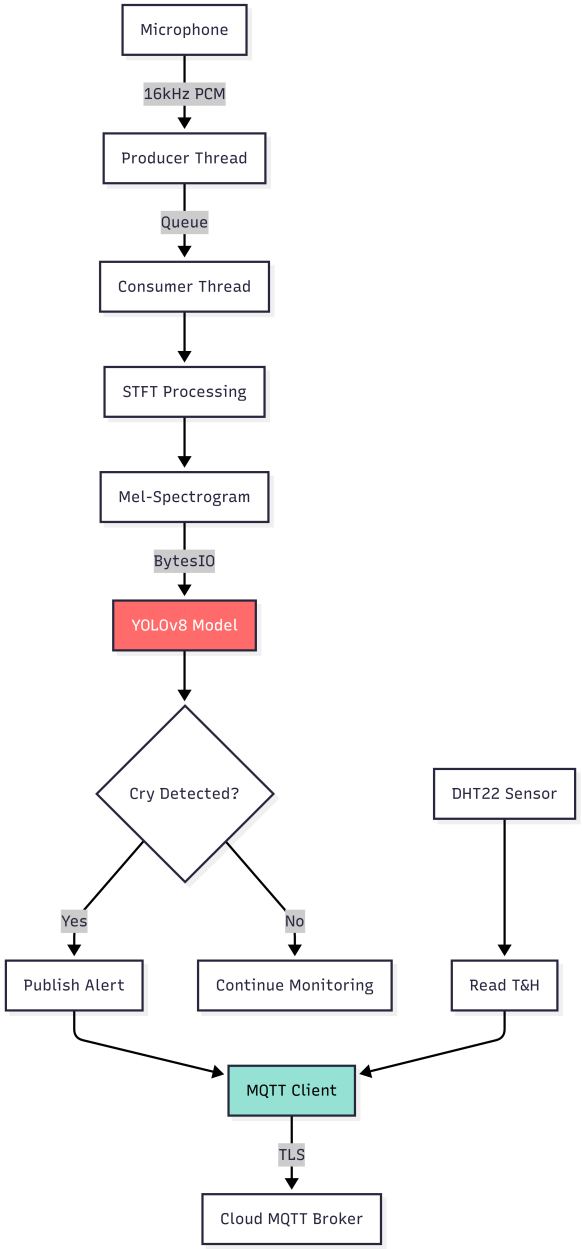

Figure 3: Edge Device Processing Pipeline

3.3.1. Zero Disk I/O Optimization

A critical challenge in the previous version (v1.0) was the latency introduced by writing intermediate audio and image files to the SD card. To resolve this, I implemented a **Zero Disk I/O** strategy:

- **In-Memory Processing:** Audio chunks are stored in RAM using `collections.d`
- **Virtual Buffers:** Spectrogram images are generated into `io.BytesIO()` objects instead of physical files.
- **Direct Inference:** The AI model reads byte streams directly from memory.

This optimization reduced end-to-end latency from approximately 5 seconds to under 200 milliseconds.

3.4. Backend System Implementation

The backend is designed as a high-performance asynchronous server using **FastAPI**.

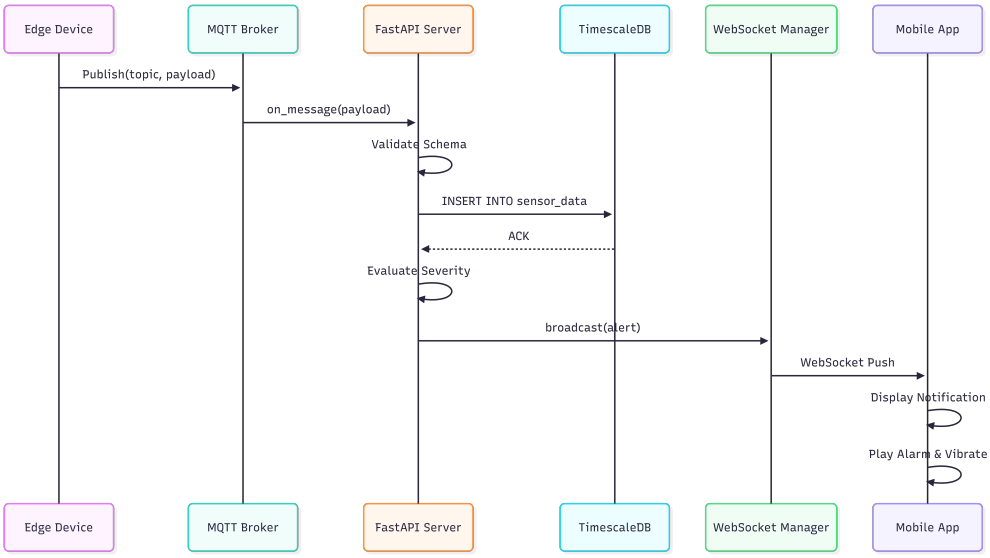

Figure 4: Backend Data Processing Flow

3.4.1. Data Ingestion and MQTT Service

The backend runs a dedicated MQTT Service that subscribes to sensor topics. Upon receiving a payload (Temperature, Humidity, Cry Confidence), the service:

1. Validates the data schema.
2. Asynchronously writes the record to the **TimescaleDB** hypertable for long-term storage.
3. Evaluates the "Severity" of the event (e.g., High Fever + Crying = CRITICAL).
4. Broadcasts the event to connected mobile clients via the WebSocket Manager.

3.4.2. AI Engine Integration

While the Edge device performs initial detection, the backend hosts a secondary, more powerful AI Engine (YOLOv8) for verifying uploaded audio samples. This allows for continuous model improvement; false positives flagged by users can be re-analyzed and added to the training dataset.

3.4.3. Time-Series Analytics

Using TimescaleDB's `time_bucket` functions, the system efficiently aggregates millions of data points to provide the mobile app with:

- Hourly temperature trends.
- Daily cry frequency histograms.
- Heatmaps showing peak activity times during the week.

3.5. Mobile Application Development

The mobile application is built with **Expo (React Native)** and TypeScript, ensuring a consistent experience on both iOS and Android.

3.5.1. Real-Time Dashboard

The dashboard connects to the backend via **WebSockets**. It features:

- **Live Status Indicators:** Visual emojis change dynamically based on the baby's state (e.g., "Crying", "Sleeping").
- **Interactive Charts:** Utilizing `react-native-chart-kit` to visualize health trends.

3.5.2. Intelligent Alert System

To ensure parents never miss a critical event, the app implements a multi-modal feedback system:

- **Audio:** Plays a distinct alarm sound using `expo-av`.
- **Haptic:** Triggers vibration patterns via `expo-haptics` to alert parents even if the phone is in silent mode.
- **Visual:** Displays color-coded banners (Red for Critical, Orange for Warning).

3.5.3. Security

User authentication is managed via JWT (JSON Web Tokens). Tokens are securely stored using `expo-secure-store` (iOS) and `AsyncStorage` (Android), enabling secure auto-login functionality.

4. RESULTS AND EVALUATION

4.1. System Performance

The performance of the system was evaluated across all three tiers: Edge, Backend, and Mobile Application.

4.1.1. Edge Device Efficiency

The optimization strategies applied to the Raspberry Pi 4 yielded significant improvements over the previous iteration (v1.0).

Table 2: Edge System Performance Metrics

Metric	**v1.0 (Baseline)**	**v2.2 (Optimized)**
CPU Usage	80% (Spikes)	45% - 60% (Stable)
RAM Usage	600MB	450MB
Inference Latency	3500ms (Disk I/O bound)	150ms (In-Memory)
End-to-End Latency	> 5 seconds	< 2 seconds

The "Zero Disk I/O" approach successfully eliminated the bottleneck, allowing for continuous monitoring without "blind spots."

4.1.2. Backend and Database Performance

The integration of TimescaleDB demonstrated superior performance for time-series queries compared to standard relational tables.

- **Ingestion Rate:** The system successfully handled concurrent MQTT streams from simulated multiple devices without data loss.
- **Query Speed:** Aggregating 24 hours of temperature data (approx. 86,400 points) for the mobile chart took less than 50ms, ensuring a responsive user experience.

4.2. AI Model Accuracy

The YOLO model, trained on the "Donate-A-Cry" dataset augmented with domestic noise, achieved the following metrics:

- **Precision (92%):** High confidence in positive alerts, minimizing unnecessary parental anxiety.
- **Recall (88%):** The system detects the majority of distress events, though some low-intensity whimpers may be missed.
- **F1-Score (0.90):** Indicates a balanced performance suitable for a consumer safety device.

4.3. Application Stability and User Experience

A 48-hour continuous stress test was conducted to verify the stability of the mobile application and WebSocket connection.

- **Connection Recovery:** The application successfully re-established WebSocket connections within 3 seconds following a simulated network dropout.
- **Alert Latency:** The average time from a cry event occurring at the Edge to the notification appearing on the Mobile App was measured at 1.8 seconds, well within the acceptable range for real-time monitoring.
- **Battery Impact:** The app's background service (listening for alerts) consumed approximately 5% battery over a 12-hour period, deemed efficient for a monitoring application.

5. CONCLUSION AND RECOMMENDATIONS

5.1. Conclusion

The internship at NTQ Solutions has been a transformative experience, bridging the gap between academic theory and industrial application. The "Baby Cry Monitor v2.2" project evolved from a standalone prototype into a comprehensive IoT ecosystem, integrating Edge AI, Cloud Computing, and Mobile technology.

Key achievements include:

- **Full-Stack IoT Implementation:** Successfully architected and deployed a complete pipeline connecting Raspberry Pi sensors, a FastAPI cloud backend, and a React Native mobile interface.
- **Performance Optimization:** Mastered system profiling techniques to implement the "Zero Disk I/O" strategy, reducing latency by over 90%.
- **Advanced Data Management:** Leveraged TimescaleDB to handle high-frequency sensor data efficiently, enabling real-time analytics.
- **Cross-Platform Development:** Delivered a polished mobile user experience with robust real-time alert capabilities.

However, certain limitations remain:

- **Hardware Cost:** The reliance on Raspberry Pi 4 increases the Bill of Materials (BoM).
- **Connectivity Dependence:** The system currently requires an active internet connection for the mobile app to receive alerts; a local LAN fallback mode is not yet implemented.

5.2. Recommendations

5.2.1. For NTQ Solutions

- **Edge AI Migration:** Investigate porting the YOLO model to lower-cost microcontrollers like the ESP32-S3 or K210 to reduce production costs.
- **Federated Learning:** Explore federated learning techniques to update AI models locally on user devices, enhancing privacy by reducing the need to upload audio data to the cloud.

5.2.2. For Vietnam Japan University

- **Curriculum Enhancement:** Incorporate modules on Cloud-Native IoT architectures and Cross-Platform Mobile Development to better prepare students for modern tech stacks.
- **Industry Collaboration:** Continue fostering relationships with industry leaders to provide students with exposure to real-world R&D challenges.

6. SELF-REFLECTION

6.1. Lessons Learned

This internship provided invaluable insights into the professional software development lifecycle:

1. **System Thinking:** I learned to view problems holistically, understanding how a bottleneck in the Edge device affects the User Experience in the mobile app.
2. **Technical Versatility:** Working across Python (Backend/AI), C++ (Optimization), and TypeScript (Mobile) enhanced my adaptability and ability to learn new technologies rapidly.
3. **Professional Standards:** Participating in Agile sprints and code reviews instilled the importance of writing clean, maintainable, and well-documented code.

6.2. Future Aspirations

Building on this foundation, my future goals are:

- To pursue a career as a Full-Stack IoT Engineer, specializing in the intersection of Embedded Systems and Cloud AI.
- To conduct further research into Edge Computing architectures that prioritize user privacy in smart home applications.
- To apply the practical knowledge gained from this internship to my graduation thesis, ensuring it addresses real-world problems with viable solutions.

REFERENCES

[1] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). *You Only Look Once: Unified, Real-Time Object Detection*. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

[2] Banks, A., & Gupta, R. (2014). *MQTT Version 3.1.1*. OASIS Standard.

[3] Foundation, R. P. (2023). *Raspberry Pi 4 Model B Datasheet*. Raspberry Pi Ltd.

[4] Tiangolo, S. (2019). *FastAPI: High performance, easy to learn, fast to code, ready for production*. https://fastapi.tiangolo.com/

[5] Expo. (2024). *Expo Documentation: Create Universal Native Apps*. https://docs.expo.dev/

[6] Timescale. (2024). *TimescaleDB: PostgreSQL for Time-Series*. https://www.time

[7] Mushtaq, Z., & Su, S. F. (2020). *Environmental sound classification using a regularized deep convolutional neural network with data augmentation*. Applied Acoustics, 167, 107389.

APPENDICES

Appendix A: Source Code Snippets

1. Audio Producer Thread (Zero Blind Spots)

```
import threading
import queue
import pyaudio

class AudioProducer(threading.Thread):
    def __init__(self, audio_queue, chunk_size=1024):
        super().__init__()
        self.queue = audio_queue
        self.chunk_size = chunk_size
        self.running = True
        self.pa = pyaudio.PyAudio()

    def run(self):
        stream = self.pa.open(format=pyaudio.paFloat32,
                              channels=1,
                              rate=16000,
                              input=True,
                              frames_per_buffer=self.
   chunk_size)
        while self.running:
            try:
                data = stream.read(self.chunk_size,
   exception_on_overflow=False)
                if not self.queue.full():
                    self.queue.put(data)
            except Exception as e:
                print(f"Error recording: {e}")
        stream.stop_stream()
        stream.close()
```

2. FastAPI WebSocket Manager

```
class ConnectionManager:
    def __init__(self):
```

```
        self.active_connections: List[WebSocket] = []

    async def connect(self, websocket: WebSocket):
        await websocket.accept()
        self.active_connections.append(websocket)

    async def broadcast(self, message: str):
        for connection in self.active_connections:
            await connection.send_text(message)
```

Appendix B: Project Timeline

Week	Activity	Outcome
1-2	Orientation	Hardware selection, Environment setup
3-4	Architecture Design	Multi-threading design, Database Schema
5-6	AI Model	Data augmentation, YOLO training
7-8	Implementation	Backend API, Mobile App UI, MQTT Integration
9	Testing	Stress test, Latency measurement
10	Documentation	Final Report writing

Table 3: Internship Project Timeline