Development of an Automated Condition Controlling and Monitoring System for an Infant Incubator (NeoCare)

An undergraduate project report submitted to the

Department of Electrical and Information Engineering
Faculty of Engineering
University of Ruhuna
Sri Lanka

in partial fulfillment of the requirements for the

Degree of the Bachelor of the Science of Engineering Honours

by

L.G.S.R. Lelwala - EG/2020/4047
G.K.H.P. Madhushani - EG/2020/4054
A.V. Shamali - EG/2020/4214
Y.M.S.K. Yaparathna - EG/2020/4307

.....
Ms. G.C.W. Thilakarathne
(Supervisor)
Mr. Neel Karunasena
(Co-Supervisor)

Abstract

The proposed project, *Development of an Automated Condition Controlling and Monitoring System for an Infant Incubator (NeoCare)*, addresses the need for continuous yet low burden monitoring of infant incubators in resource limited Neonatal Intensive Care Units(NICU), where routine checks are often manual and staff intensive. NeoCare is an edge cloud monitoring and decision support system that combines non invasive sensing and on device machine learning to support timely clinical attention while keeping deployment practical. A Raspberry Pi device operates as the bedside edge controller with two cameras and an audio input. Temperature and humidity are acquired using a camera based incubator LCD reader (YOLOv8 detection followed by EasyOCR), avoiding modifications to commercial incubators and improving adaptability across different models. A Neutral Thermal Environment (NTE) rule engine converts infant age and weight into recommended temperature ranges. In parallel, an image based jaundice detector using MobileNetV3 Small and an audio pipeline with a YAMNet based cry detector followed by an ensemble cry classifier generate alerts for clinical review. Telemetry is published via MQTT to a ThingsBoard Community Edition deployment integrated with Google Cloud Provider(GCP) services, while role based web dashboards and a Flutter mobile app provide real-time views and notifications. Parent access is restricted to authorised live streaming and basic status, controlled by clinical staff. Experimental evaluations showed jaundice detection accuracy of 93.0% (97.6% specificity), cry detection accuracy of 97.8%, cry classification test accuracy of 92.9%, LCD reader performance of 99.3% mAP@50, and 100% correctness for evaluated NTE recommendations. Overall, NeoCare provides continuous monitoring, actionable recommendations, and scalable remote visibility to support NICU workflows.

Acknowledgments

We are pleased to provide our sincere gratitude to our supervisor, Ms. G.C.W. Thilakarathne, for her valuable guidance, inspiration, and dedication towards this undergraduate project. Her expertise and insightful suggestions were instrumental. We also want to thank our co-supervisor, Mr. Neel Karunasena, for his useful guidance and advice. Also, we want to express our appreciation Dr. Iromi Ranaweera, EE7802 Undergraduate Project Coordinator, for the guidance given. We would also like to extend our sincere appreciation to Dr. S.D.U.M. Ranga, Director of National Hospital Galle, for his approval to do NICU observation and test run. We would also like to appreciate the support provided by Dr. Nirmala Waththuhewa of National Hospital Galle to facilitate our project. We would also appreciate the support provided to our project from the NICU team for their knowledge regarding our chosen field, which was helpful for us to do data evaluation. At last, our appreciation would extend to all individuals who provided us support to complete our project.

Contents

List of Figures

List of Tables

Acronyms

ADASYN	-	Adaptive Synthetic Sampling
API	-	Application Programming Interface
ARM	-	Advanced RISC Machine
CE	-	Community Edition
CLAHE	-	Contrast Limited Adaptive Histogram Equalisation
CNN	-	Convolutional Neural Network
ECG	-	Electrocardiogram
EMR	-	Electronic Medical Record
FCM	-	Firebase Cloud Messaging
GCP	-	Google Cloud Provider
HTTP	-	Hypertext Transfer Protocol
IoT	-	Internet of Things
JSON	-	JavaScript Object Notation
JWT	-	JSON Web Token
LCD	-	Liquid Crystal Display
LSTM	-	Long Short-Term Memory
MAH	-	Marketing Authorisation Holder
MFCC	-	Mel Frequency Cepstral Coefficients
MJPEG	-	Motion JPEG
ML	-	Machine Learning
MQTT	-	Message Queuing Telemetry Transport
NICU	-	Neonatal Intensive Care Unit
NMRA	-	National Medicines Regulatory Authority
NTE	-	Neutral Thermal Environment
OCR	-	Optical Character Recognition
ONNX	-	Open Neural Network Exchange
PPG	-	Photoplethysmography
REST	-	Representational State Transfer
RH	-	Relative Humidity
RMS	-	Root Mean Square
ROC	-	Receiver Operating Characteristic
SMOTE	-	Synthetic Minority Oversampling Technique
SPA	-	Single Page Application
SQL	-	Structured Query Language
STFT	-	Short-Time Fourier Transform
UI	-	User Interface
VM	-	Virtual Machine
VPN	-	Virtual Private Network

Chapter 1

Introduction

Premature birth (before 37 completed weeks of gestation) remains a major global health concern, with an estimated 13.4 million preterm births reported in 2020, and complications of prematurity continuing to be a leading cause of mortality in children under five years of age. Children who manage to survive preterm birth usually present health concerns in the years to come. These concerns range from neuro developmental issues to vision and hearing impairments. Outcomes associated with preterm births differ from one healthcare setting to another. While in low income and resource scar environments the lack of appropriate neonatal care in terms of heating or cooling the baby in order to maintain body temperature or protecting the child from infections as well as providing basic respiratory care is the main reason for the large number of preterm related child mortalities, especially in the case of extreme preterm births. These disparities are further compounded by constraints in staffing, infrastructure, clinical guidelines, and availability of essential equipment and consumables in many NICUs, demanding the need for innovative, context appropriate solutions capable of delivering effective care even in resource limited settings [1, 2].

1.1 Background

1.1.1 High risks faced by premature infants in developing countries

NICU settings in developing countries, outcomes for preterm infants are largely revolved by constraints in staffing, availability of necessary infrastructure, and access to necessary care. In low resource environments, the lack of access to necessary care options, including thermal care, breast feeding support, infection control measures, and basic respiratory care options, leads to increased rates of mortality that are often much higher among babies born either at or before 32 weeks gestation [1, 2]. Evidence suggests that in low resource environments, more than 90% of extremely preterm births (before 28 weeks gestation) may result in mortality during the first week of life, whereas the rates in high resource environments remain substantially lower [1].

In addition to mortality, if a newborn is born prematurely, he or she is prone to various complications, both in the short term and long term. Short term complications in newborn babies include, but are not limited to, difficulties in the respiratory sys-

tem (respiratory distress syndrome, apneic episodes), cardiovascular instability (such as patent ductus arteriosus, hypotension), intraventricular hemorrhage, hypothermia, which can occur due to a lack of subcutaneous fat, difficulties in nutrient intake (difficulty in feeding, necrotizing enterocolitis), hematologic disorders (such as anemia, jaundice), infections such as sepsis [3]. Long-term complications include developmental delay, pulmonary disease, vision, and hearing impairments, which continue to pose a burden on families as well as the healthcare system [1–3]. These risks highlights the importance of maintaining stable incubator conditions and enabling timely attention to the babies, especially when resources are limited.

1.1.2 Challenges of traditional infant incubators

In conventional incubators for infants, it is often necessary for continuous human intervention in order to maintain suitable environments for preterm babies. In a practical clinical setting, it would often be necessary for clinical to monitor temperature, humidity, and oxygen levels and make adjustments when deviations occur. However, adjustments may sometimes be delayed if there are other important matters to attend to in busy NICU settings.

Another concern is the performance in thermoregulation. The functionality of some incubators may not be able to sustain a constant temperature in the hood since the heat transfer mechanisms may not be entirely accounted for in resistance incubators, with a variation of around $\pm 0.8\,^{\circ}\mathrm{C}$ in some models [4]. Moreover, the incubator may utilize resistive heating without a cooling system when the environmental temperature is high, constituting a possible reason for the development of hyperthermia, especially in a hot climate [4, 5].

Safety and comfort related factors are also discussed in the literature. Some incubators emit electromagnetic fields, and exposure levels above 200 mG have been suggested as a potential concern for both preterm infants and healthcare staff in close contact; design measures such as increasing separation and shielding are commonly recommended to reduce exposure [4]. Furthermore, incubators may provide limited attenuation of NICU noise and can contribute additional fan noise; sustained exposure to elevated noise is associated with adverse effects on preterm infants, including sleep disturbance and stress related physiological changes [4, 5].

Relative humidity (RH) management is another area where practice varies. Maintaining appropriate RH is clinically important because it influences trans epidermal water loss, thermal stability, electrolyte balance, oxygen consumption, infection risk, and skin integrity, yet clinical approaches to RH control are often inconsistent [4]. Finally, temperature distribution may be slow and non uniform, producing localized hot or cold regions that can cause discomfort or physiological stress, and closed incubator designs can limit rapid access for procedures such as ventilation support, pulse oximetry, or ECG monitoring [4, 6]. These limitations motivate the need for monitoring and decision support solutions that improve consistency, visibility, and responsiveness without increasing bedside workload.

1.1.3 Neonatal Care in Sri Lanka

Neonatal intensive care units in many low and middle income countries are faced with challenges of equipment availability, infrastructure, personnel capacity, and technical medicine. Such challenges may impede consistency and efficiency of care delivery, more so among preterm and low birth weight infants who require special care regarding thermal regulation and prevention of infection.

The prematurity still poses a significant clinical burden within the Sri Lankan context. A local study of 9,130 live born neonates reported a prematurity rate of 10.9%, indicating a substantial proportion of infants who may require specialised neonatal care [7]. Although Sri Lanka has achieved a major long term reduction in infant mortality compared with the mid 20th century, continued improvement in neonatal services remains important. Recent national and international estimates still report infant mortality at several deaths per 1,000 live births [8, 9]. Similar to other developing country settings, practical challenges reported by healthcare workers include limitations in equipment, training, and facility resources, which can affect the delivery of timely interventions for preterm and sick newborns.

Given these realities, this project focuses on a set of parameters that can be monitored continuously and acted upon at the bedside: incubator temperature and humidity, together with infant age and weight for personalised thermal recommendations. Neutral Thermal Environment (NTE) guidance provides age and weight specific incubator temperature ranges; the values used in this work are summarised in Table 1.1. Maintaining appropriate ranges is particularly critical for infants under 1500 g, who typically require higher incubator air temperatures in early life, with set points gradually reduced as thermoregulatory ability improves [10].

Table 1.1: Neutral Thermal Environmental Temperatures

Age and Weight	**Starting Temperature (°C)**	**Range of Temperature (°C)**
0–6 h		
Under 1200 g	35.0	34.0–35.4
1200–1500 g	34.1	33.9–34.4
1501–2500 g	33.4	32.8–33.8
Over 2500 g (and >36 wk)	33.9	32.0–33.8
>6–12 h		
Under 1200 g	35.0	34.0–35.4
1200–1500 g	34.0	33.5–34.4
1501–2500 g	33.1	32.2–33.8
Over 2500 g (and >36 wk)	32.8	31.4–33.8
>12–24 h		
Under 1200 g	34.0	34.0–35.4
1200–1500 g	33.8	33.3–34.3
1501–2500 g	32.8	31.8–33.8
Over 2500 g (and >36 wk)	32.4	31.0–33.7
>24–36 h		

Age and Weight	**Starting Temperature (°C)**	**Range of Temperature (°C)**
Under 1200 g	34.0	34.0–35.0
1200–1500 g	33.6	33.1–34.2
1501–2500 g	32.6	31.6–33.6
Over 2500 g (and >36 wk)	32.1	30.7–33.5
>36–48 h		
Under 1200 g	34.0	34.0–35.0
1200–1500 g	33.5	33.0–34.1
1501–2500 g	32.5	31.4–33.5
Over 2500 g (and >36 wk)	31.9	30.5–33.3
>48–72 h		
Under 1200 g	34.0	34.0–35.0
1200–1500 g	33.5	33.0–34.0
1501–2500 g	32.3	31.2–33.4
Over 2500 g (and >36 wk)	31.7	30.1–33.2
>72–96 h		
Under 1200 g	34.0	34.0–35.0
1200–1500 g	33.5	33.0–34.0
1501–2500 g	32.2	31.1–33.4
Over 2500 g (and >36 wk)	31.3	29.8–32.8
>4–12 days		
Under 1500 g	33.5	33.0–34.0
1501–2500 g	32.1	31.0–33.2
Over 2500 g (and >36 wk)	31.0	29.5–32.6
4–5 days	30.9	29.4–32.3
5–6 days	30.6	29.0–32.2
6–8 days	30.3	29.0–31.8
8–10 days	30.1	29.0–31.4
10–12 days		
>12–14 days		
Under 1500 g	33.5	32.6–34.0
1501–2500 g	32.1	31.0–33.2
>2–3 wk		
Under 1500 g	33.1	32.2–34.0
1501–2500 g	31.7	30.5–33.0
>3–4 wk		
Under 1500 g	32.6	31.6–33.6
1501–2500 g	31.4	30.0–32.7
>4–5 wk		
Under 1500 g	32.0	31.2–33.0
1501–2500 g	30.9	29.5–32.2
>5–6 wk		
Under 1500 g	31.4	30.6–32.3
1501–2500 g	30.4	29.0–31.8

Normal body temperature of the newborns is typically in the range of 36.5 °C to

37.5 °C. Temperatures below 36.5 °C (hypothermia) or above 37.5 °C (hyperthermia) can be life threatening, and the clinical grading of hypothermia is summarised in Figure 1.1 based on national newborn care guidance.

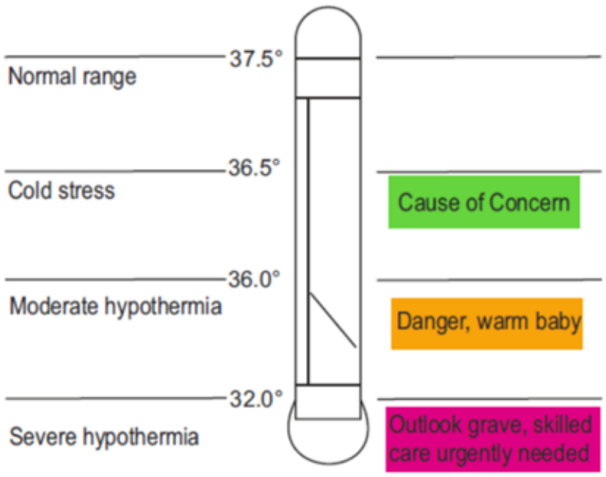


Figure 1.1: Grading of hypothermia

Infant weight, especially for premature and low birth weight babies, is a key indicator of growth and clinical risk, and standard treatment protocols provide specific guidance for the management of low birth weight neonates. Besides management by temperature regulation, there is also humidity management which becomes important in premature infants because of potential problems due to humidity anomalies. To reduce excessive water loss and maintain a suitable incubator microenvironment, relative humidity is commonly maintained around 50-60% in neonatal incubators [11, 12]. Considering these clinical requirements, the automated system in this study is designed around temperature, humidity, and infant weight as critical parameters for monitoring and recommendation.

1.2 Problem Statement

Traditional infant incubator care in many NICUs still depends heavily on continuous manual observation and parameter adjustment by clinical staff. This approach can be time consuming and may lead to delayed corrective action in understaffed or high workload settings. Maintaining a stable microenvironment is critical for premature infants, yet conventional systems may show variability in temperature and humidity control, and practical limitations such as limited connectivity reduce the feasibility of timely alerts and remote review. In addition, many existing workflows do not provide structured, continuous data that can be used for trend monitoring or for generating evidence based recommendations aligned with newborn care guidelines.

A further challenge is that interpretation of infant distress cues is often subjective and workload dependent. Without reliable automated support, events such as prolonged crying or early signs associated with jaundice may not be recognised promptly, especially when staff attention is distributed across multiple infants. Similarly, parents

are typically separated from direct observation and depend on staff updates, which can increase anxiety and reduce transparency during NICU stays. These gaps collectively motivate the need for a practical monitoring and decision support system that provides continuous visibility of incubator conditions, produces timely alerts for abnormal events, and enables secure role-based remote access for clinicians and parents without increasing the burden on bedside staff.

1.3 Objectives and Scope

1.3.1 Objectives

- To develop an edge based monitoring and decision support system for neonatal incubators that continuously acquires temperature and humidity readings and presents them in real time through secure interfaces.
- To implement Neutral Thermal Environment (NTE) based recommendations using infant age and weight to provide evidence based target temperature ranges for clinical use.
- To design and integrate a camera based incubator LCD reading pipeline to enable non invasive acquisition of incubator parameters without modifying commercial incubator hardware.
- To integrate and deploy machine learning modules for auxiliary clinical support, including neonatal jaundice risk detection from images and infant cry detection/classification from audio, with alert generation for clinical review.
- To develop secure role based web dashboards and a mobile application that provide clinicians with live readings, trends, and alarms, while providing parents with restricted access (e.g., authorised live video streaming and basic status) controlled by clinical staff.
- To design the overall system as a practical and scalable prototype suitable for resource limited environments, with emphasis on reliability, maintainability, and secure data handling.

1.3.2 Scope

- The project is more about developing a functional prototype monitoring and decision support system and not about redesigning the incubator's internal control system.
- The data acquisition process is mainly carried out through non contact technologies (camera based LCD reading) with additional sensing through the camera and audio. Integration with the incubator's internal sensors and actuators is not possible due to constraints in the safety and accessibility aspects.
- The scope includes implementation of software services on an edge device (Raspberry Pi device), telemetry publishing via MQTT, cloud visualisation and device

management using ThingsBoard, and deployment of supporting services on a cloud environment.

- The scope includes UI development for clinicians, parents, and administrators with authentication and role based access control; administrative management functions remain web only.
- Comprehensive clinical trials and full medical device regulatory approval are not included in this stage; however, the prototype is developed with consideration of good engineering practice, patient privacy, and safety oriented deployment.
- Scaling to multiple incubators is considered in the system architecture and data model, but large scale deployment and long term hospital integration are treated as future work.

Chapter 2

Literature Review

2.1 Previous Work

2.1.1 Sensor Integration and Real-time Data Monitoring

Measurements of air or skin temperature, humidity, and (where applicable) oxygen concentration are the main ways that modern neonatal incubators control the infant's microenvironment. Thermistors are frequently used in conventional units to regulate temperature in both air mode and skin mode operation, with reported accuracy levels of $\pm 0.8\,^{\circ}\mathrm{C}$ for air mode and $\pm 0.5\,^{\circ}\mathrm{C}$ for skin mode [13]. Humidity control is typically achieved by warming air over a water reservoir, where settings may be adjusted mechanically or via active humidification modules which operating ranges such as 40%–90% RH are commonly described in manufacturer documentation [7, 14]. For infants requiring supplemental oxygen, some incubators provide controlled oxygen delivery and display the concentration, often within clinically used ranges (e.g., 21%–60%) [7, 14]. In addition to contact sensors, non contact approaches (e.g., infrared sensing and camera-based monitoring) are explored to reduce infection risk and improve practicality in situations where direct access to internal sensors is limited.

ECG, pulse oximetry, and respiration monitors are commonly used to continuously monitor the cardiorespiratory state of newborns. Electrodes are applied to the baby's body to measure the electrical activity of the heart during electrocardiography (ECG). chest and continues to be the standard technique for clinical heart rate monitoring [14]. Motion and low perfusion can impact the accuracy of pulse oximetry, which uses optical sensors at peripheral locations to measure heart rate and oxygen saturation [14]. Recent work also explores contactless heart rate estimation using camera-based photoplethysmography (cbPPG), which analyses subtle skin colour variations associated with the cardiac cycle, although performance may be sensitive to illumination changes and movement [11]. While specialized respiration or apnea monitors follow chest and abdominal motion to determine respiratory rate, impedance pneumography with ECG electrodes can measure impedance changes over the thorax/abdomen to set off sirens during episodes of apnea [15, 16]. A summary comparison of commonly used neonatal monitoring sensors and their trade-offs is provided in Table 2.1.

Real time acquisition and processing are essential because they enable timely detection of deviations and support rapid clinical response. Continuous streams of environmental and physiological measurements allow automated systems to identify trends,

detect abnormal conditions early, and generate alerts before instability becomes severe [17]. In practice, the value of real time monitoring is not only in measurement accuracy, but also in the ability to maintain consistent updates, reduce delayed interventions, and support decision making with time stamped evidence.

Table 2.1: Comparison of neonatal monitoring sensors

Sensor Type	**Measured**	**Technology**	**Advantages**	**Disadvantages**	**Ref.**
Thermistor	Temperature	Resistance varies with temperature	Simple, low cost	Contact-based; placement affects accuracy	[13]
Non-contact infrared	Temperature	Measures emitted infrared radiation	Non-invasive; reduces infection risk	Sensitive to distance and ambient conditions	[14]
Electrochemical	Oxygen	Current from oxygen reaction	Direct measurement	Limited sensor lifespan	[7, 14]
Optical	Oxygen	Light absorption at specific wavelengths	Non-contact options available	Affected by motion and ambient light	[7, 14]
ECG	Heart rate	Electrical activity of the heart	High accuracy	Requires electrode placement	[14]
Pulse oximetry	Heart rate, SpO_2	Optical absorption through tissue	Non-invasive; widely used	Motion/low perfusion sensitivity	[14]
Camera-based PPG	Heart rate	Skin colour variation analysis	Contactless	Sensitive to lighting and motion	[11]
Impedance pneumography	Respiratory rate	Thoracic/abdominal impedance changes	Can reuse ECG electrodes	Motion and cardiogenic artefacts	[15]
Respiration/apnea monitor	Respiratory rate	Detects chest/abdominal movement	Targeted apnea detection	May require additional sensors	[16]
DHT sensor	Temp., humidity	Capacitive humidity + thermistor	Low cost; easy interface	Lower accuracy; limited range	[17]
Load cell	Weight	Strain gauge	High accuracy	Requires calibration; overload risk	[17]
Optical HR sensor	Heart rate	Photoplethysmography (PPG)	Simple sensing	Affected by motion/ambient light	[17]
General O_2 gas sensor	Oxygen conc.	Multiple technologies available	Direct oxygen measurement	Accuracy/lifespan vary by technology	[17]

2.1.2 Automation and Microcontroller Systems

Medical devices that provide functions of modern care require components that integrate processing units with memory and peripheral functions in forms using limited power. These components appear in systems for measurement and observation at the bedside and in tools that support examination and assessment. The systems use these components to collect data from measurement devices, to conduct operations that control functions, and to provide oversight for processes that occur in an automated manner. Analysis of the market for these components in medical devices indicates a value of four hundred fifteen million dollars in two thousand twenty three. The same analysis suggests the market will show a value of six hundred seventy five million dollars by two thousand thirty, following data that industry sources provide [18, 19].

A broad set of microcontroller families is used in medical and IoT oriented designs, depending on power, performance, and integration requirements. Entry level platforms such as Arduino based boards are often adopted in prototypes because of their ease of development and ecosystem support, while WiFi enabled modules such as ESP8266/ESP32 are commonly used when low cost connectivity is required. For more demanding applications, ARM Cortex-M based families (e.g., STM32 and MAX326xx) offer higher computational capability, lower power operation, and richer peripheral sets suitable for sensor interfacing and real time tasks. Other widely used device families include PIC microcontrollers, ultra low power Silicon Labs variants, Renesas RA series microcontrollers, and radiation tolerant Cortex-M options for specialised environments [17, 20–25].

Microcontrollers usually offer closed loop control in incubator automation and monitoring by reading measurements relating to temperature, humidity, and oxygen and controlling actuators like heaters, fans, humidifiers, and valves based on control algorithms (such as PID style logic or threshold control) [17, 20, 21]. While local computation lessens need on ongoing human intervention, their capacity to analyze various sensor streams in real time facilitates continuous tracking of incubator conditions and vital measures [17, 19, 23]. Deterministic timing, sufficient memory and processing headroom, dependable interfacing, and (where necessary) wireless connectivity for telemetry and remote monitoring are among the requirements that influence microcontroller selection for medical applications. Energy efficiency is also crucial for bedside systems that must run continuously [1].

Table 2.2 summarises representative microcontroller platforms and common characteristics reported in the literature and vendor documentation.

Table 2.2: Comparison of microcontroller platforms used in medical and IoT oriented devices

Platform	**Core**	**Clock**	**Memory (Flash/RAM)**	**Key features**	**Ref.**
Arduino Uno (ATmega328P)	8-bit AVR	16 MHz	32 KB / 2 KB	Simple prototyping, large ecosystem	[17]
Arduino Mega 2560	8-bit AVR	16 MHz	256 KB / 8 KB	More I/O and memory	[17]
Arduino Nano 33 IoT	ARM Cortex-M0+	48 MHz	256 KB / 32 KB	Wi-Fi/BLE options, compact form	[20]
ESP8266	32-bit RISC	80–160 MHz	Up to 16 MB / varies	Low-cost Wi-Fi connectivity	[20]
ESP32 (typical)	32-bit dual-core	Up to 240 MHz	Up to 16 MB / 520 KB	Wi-Fi + BLE, higher compute	[20]
STM32 (various)	ARM Cortex-M	Up to 600 MHz	Up to 2 MB / 1.4 MB	Rich peripherals, low power, security	[23]
MAX326xx	ARM Cortex-M4F	Up to 100 MHz	Up to 3 MB / 1 MB	Low power, sensor-centric features	[24]
PIC (Microchip)	Proprietary	varies	varies	Wide device range, mature tooling	[21]
C8051F96x	8051	25 MHz	Up to 128 KB / 8 KB	Ultra-low power, AES options	[22, 25]
Renesas RA family	ARM Cortex-M	varies	varies	Security features, low-power variants	[17]
UT32M0R500	ARM Cortex-M0+	50 MHz	128 KB / 16 KB	Radiation tolerant, robust interfaces	[17]

2.1.3 Machine Learning for Predictive Health Analysis

Machine learning (ML) is increasingly used in neonatal monitoring to analyse continuous physiological and environmental data streams and to identify early indicators of deterioration before they become clinically obvious. By learning patterns from historical and real time sensor measurements, ML models can detect subtle deviations, support earlier clinical review, and reduce delayed intervention in high workload NICU environments [17].

For incubator monitoring and performance estimates, a variety of supervised learning techniques, such as linear regression, decision trees, Naive Bayes, random forests, and deep neural networks, have been investigated. These approaches are used to model relationships among incubator environmental variables (e.g., temperature and humidity), infant related measurements, and operational performance indicators. In the context of incubator operation, ML models may support predictive maintenance by flagging abnormal system behaviour in post market data, and may also contribute to adaptive environmental control by recommending parameter adjustments based on trends and infant status [17, 26].

Predictive analytics has also been studied for higher level clinical outcomes such as mortality risk, where algorithms including gradient boosted models and random forests are trained on clinician selected variables to generate risk estimates that can

assist early decision making. Practical challenges include class imbalance, particularly when adverse outcomes are rare; therefore, resampling and synthetic data techniques such as SMOTE and ADASYN are often applied to improve model learning in minority classes. Since many neonatal measurements are time varying, recurrent neural networks, especially Long Short Term Memory (LSTM) models, are commonly adopted for time series forecasting and anomaly detection. LSTM networks learn temporal dependencies from sequential vital sign streams and have been reported to achieve high predictive accuracy for incubator related parameters in controlled studies [27].

2.1.4 Medical Device Design for Neonatal Care Systems

Research on neonatal incubator design increasingly emphasises "smart" systems that support both environmental stability and continuous monitoring. Typical design objectives include minimizing hood variability, preserving a regulated warm and humid microclimate, and delivering prompt alerts when parameters diverge from clinical objectives. In order to promote safer bedside care, many suggested designs incorporate sensor feedback with automated temperature and humidity regulating mechanisms, and others expand monitoring to infant related measures (such as skin temperature) [20]. Including connectivity options for data logging and remote monitoring is a popular trend that is frequently defined inside Internet of Things (IoT) frameworks. These methods allow clinicians to keep longitudinal records that aid in maintenance and decision-making, check readings on computers or mobile devices, and provide alerts for abnormal situations [20].

Robustness and affordability are other common themes, especially in environments with limited resources. To increase availability and resilience in settings with erratic electrical supplies, a number of research suggest low cost incubator variations and dual power approaches [4, 20, 21, 28]. Other work focuses on improving neonatal comfort by approximating womb like conditions through controlled temperature, adequate humidity, and, in some designs, managed oxygen concentration. The underlying objective is to reduce thermal stress and fluid loss while supporting stable physiological adaptation in premature infants [12, 17, 20]. These directions motivate designs that are modular, maintainable, and adaptable to different clinical constraints.

The National Medicines Regulatory Authority (NMRA), which oversees the registration, licensing, importation, sale, and distribution of medical devices, has regulations that must be taken into account when developing medical devices in Sri Lanka [29–31]. Depending on the type of device, local testing may be necessary. The registration process is usually multi stage (sample import licence, device registration, and import licence) and may require extensive documentation, such as authorization letters, free sale certificates, product labels, user manuals, and technical specifications [29, 32, 33]. A Marketing Authorization Holder (MAH) is typically needed to handle local compliance duties and submit applications for international manufacturers [30, 32]. Additionally, registration frequently requires proof of quality control and compliance, such as ISO 13485 certification and CE accreditation [29].

Clinical safety expectations for neonatal equipment further emphasise proper prepa-

ration of the NICU and delivery room, prevention of hypothermia, and strict infection control practices, including hand hygiene, use of personal protective equipment, sharps management, and safe handling of blood and body fluids [34–36]. International standards are also relevant. For example, IEC 60601-2-19 specifies particular requirements for basic safety and essential performance of infant incubators, and related neonatal transport standards are referenced for transport equipment [37]. Together, these design and regulatory considerations highlight the importance of safety oriented engineering, traceable requirements, and practical deployment constraints when developing neonatal monitoring systems.

Chapter 3

Methodology

3.1 Overall System Architecture

At a high level, the proposed system follows a pipeline that begins at the incubator bedside, performs sensing and machine learning inference on an edge device, publishes derived measurements and events to a cloud back end, and finally presents information and alerts to clinicians and parents through role based user interfaces. Rather than modifying or enclosing the internal electronics of existing incubators, the system introduces an external sensing, analysis, and connectivity layer designed specifically to operate within the constraints of NICU with minimal disruption.

On the input side, the system collects data using multiple sensors: a camera pointed at the infant's face and skin (for image-based analysis), a camera capturing the incubator's LCD display (for temperature and humidity extraction), and a microphone recording the infant's cry (for cry analysis). At the edge, these signals are processed by a Raspberry Pi 4B+ executing multiple services that acquire these signals, run ML inference modules (jaundice detection and cry detection/classification), extract incubator readings through the LCD reading pipeline, and compute Neutral Thermal Environment (NTE) recommendations based on infant age and weight. On the output side, the edge device publishes time stamped telemetry and events via MQTT to a ThingsBoard Community Edition instance deployed on Google Cloud Platform (GCP). These data are then consumed by Cloud Run services and visualised through a React based web dashboard, while a Flutter mobile application provides notifications and authorised views for clinicians and parents. Figure 3.1 presents a simplified input-processing-output view of the architecture before the detailed components are discussed.

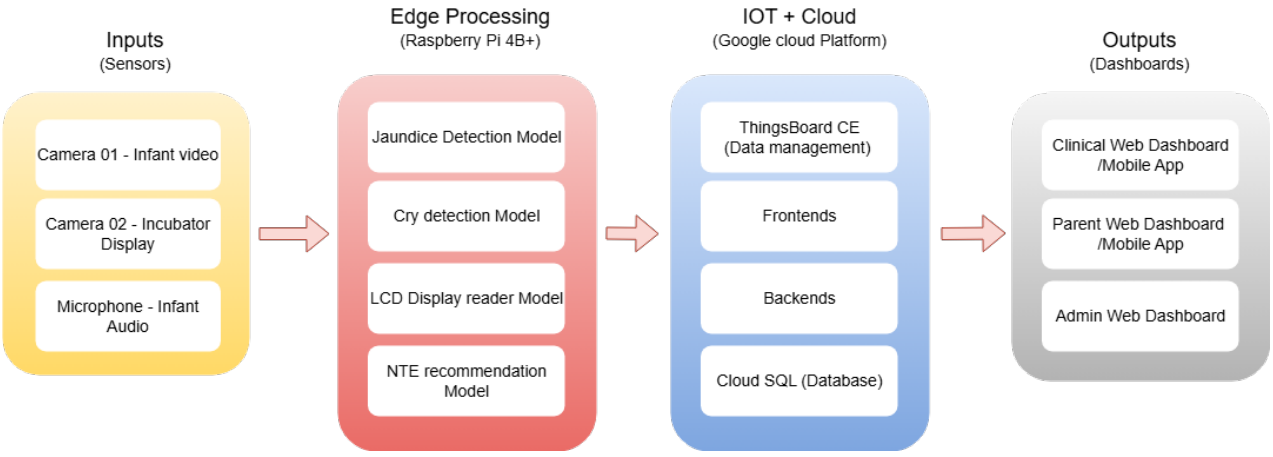


Figure 3.1: High level architecture.

The remainder of this section provides a detailed view of the architecture. Subsection 3.1.1 outlines the structure of the edge and cloud components, including the use of Tailscale and Nginx for secure connectivity. Subsection 3.1.2 then explains the end-to-end data flow, from frame and audio capture through processing and cloud transmission to dashboard visualisation and notifications.

3.1.1 Edge-Cloud Architecture with Raspberry Pi Device

At the edge, a Raspberry Pi 4B+ (4 GB RAM) is installed near the incubator and connected to two USB cameras and an audio input. The infant facing camera provides frames for image based analysis and authorised live video viewing, while the LCD facing camera captures the incubator display for non invasive extraction of numerical readings (e.g., temperature and humidity, and other parameters when presented on the display). Both video streams are made available through an MJPEG stream using [38], and are consumed by the edge microservices maintained in the Raspberry Pi services repository [39].

The LCD reading service loads a YOLOv8 detector and an OCR engine (EasyOCR) from the display reader repository [40] and exposes a local HTTP endpoint (e.g., `/readings`) that returns validated measurements as structured JSON. The jaundice inference service wraps a MobileNetV3 Small model exported to ONNX from the jaundice repository [41]; it applies basic input gating (e.g., image brightness and infant presence checks) before producing a jaundice risk output. For audio, the cry pipeline performs real time detection and triggers short recordings that are then classified locally using the cry models and feature extraction pipeline [42]. In parallel, the NTE service calls into the NTE recommendation engine [43] to map infant age and weight to recommended temperature ranges and to generate advisory messages when readings deviate from the target range.

After edge processing, time stamped telemetry is published via MQTT to a ThingsBoard Community Edition instance deployed on Google Cloud Platform (GCP). The ThingsBoard configuration (device profiles, rule chains, and dashboards) is maintained in the ThingsBoard repository [44]. Telemetry fields include incubator readings

(e.g., `air_temp`, `humidity`), derived ML states (e.g., `jaundice_status`, `cry_status`), and recommendation outputs (e.g., `nte_range`, `nte_advice`). ThingsBoard stores these data and exposes REST/WebSocket interfaces that are consumed by the application layer.

The application layer is built from the integration repository [45]. The React dashboard is deployed on Cloud Run and presents role based interfaces for clinicians, parents, and administrators. Two Node.js services (admin and parent/clinician backends) provide authentication (JWT based), user and infant management workflows, invitation/PIN flows, messaging, and metadata APIs; persistent application data are stored in a Cloud SQL PostgreSQL instance. Administrative actions (e.g., account and device management) are limited to the web only admin interface, while parent access is restricted to authorised live viewing and basic status as configured by clinical staff.

Secure access to edge hosted endpoints (including live video and local inference APIs) is implemented using a Tailscale based VPN overlay and an Nginx reverse proxy [46]. A small VM joins the same Tailscale network as the Raspberry Pi and runs Nginx rules (maintained in the integration repository) that forward only specific routes (e.g., camera stream and selected edge APIs) to the Raspberry Pi over the encrypted Tailscale path. This design avoids exposing Raspberry Pi services directly to the public internet and limits inbound access to controlled, proxied endpoints while the edge device maintains only outbound connectivity. Figure 3.2 summarises the resulting edge cloud structure, including the Raspberry Pi services, ThingsBoard CE, GCP hosted application services, and the client applications.

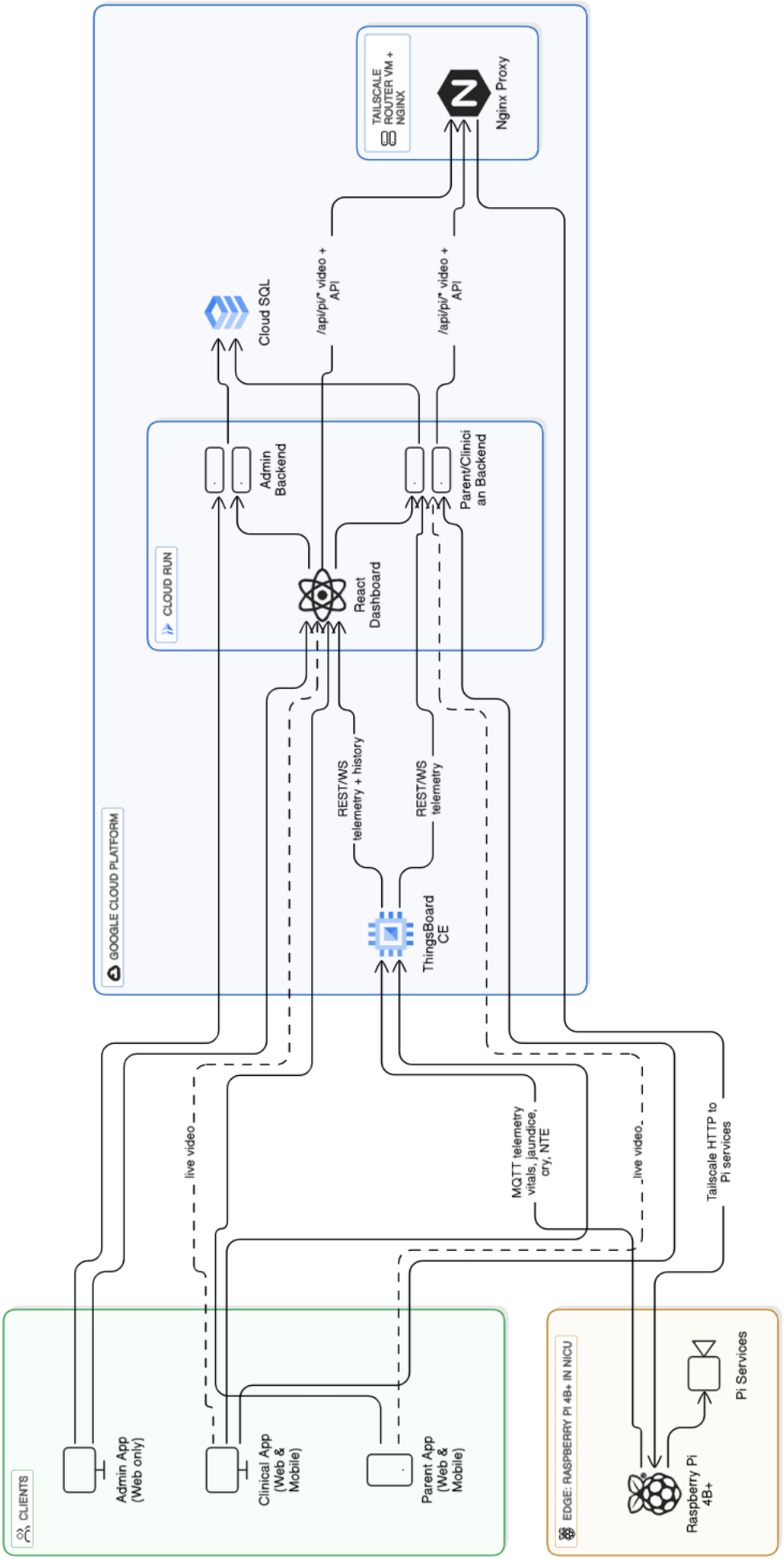


Figure 3.2: Overall architecture.

3.1.2 End-to-End Data Flow.

Beyond the static component layout, it is important to describe how data moves through the system during normal operation. The flow begins at the incubator bedside, where the Raspberry Pi device continuously acquires frames from two cameras and audio from the microphone. Frames from the infant facing camera are routed to the jaundice subsystem [41]. Before inference, each frame is filtered using a brightness check and a baby presence gate so that only usable images are forwarded to the MobileNetV3 Small model. In parallel, frames from the LCD facing camera are processed by the display reader pipeline [40]: YOLOv8 localises the relevant display regions and EasyOCR converts the detected digits into numerical values, followed by validation logic such as range checking based on neonatal guidance [47]. Audio is processed in short windows; a real-time detector identifies likely cry segments and triggers a short recording that is passed to the cry classification model to produce a cry/no cry state and a cry type label [42]. Separately, the NTE service periodically calls the rule engine [43] using infant age and weight together with current incubator air temperature, returning an NTE temperature band and a concise recommendation.

The resulting measurements and derived states are combined into structured telemetry messages and published via MQTT to the ThingsBoard Community Edition instance. ThingsBoard persists time series values and evaluates rule chains to create alarms when configured thresholds are crossed (e.g., incubator temperature outside the recommended band, abnormal extracted readings, or persistent cry events). The web application layer running on Cloud Run [45] uses the ThingsBoard REST/WebSocket APIs to fetch current telemetry and historical data for charting, while the backend services use Cloud SQL to manage application data such as user accounts, infant profiles, and authorisation links between parents and infants. When a clinician opens the clinical dashboard, the frontend first retrieves the list of authorised infants/incubators via the backend, then queries ThingsBoard for the selected device's latest values and history. When the user requests live video, the dashboard accesses a proxied endpoint that is forwarded by an Nginx reverse proxy over a Tailscale secured path to the Raspberry Pi stream, and the returned MJPEG feed is rendered in the browser. A similar flow is used in the mobile application: the app obtains an authorised proxy URL from the backend and consumes the same secured video resource. Parent access is restricted to approved infants and basic views, and permissions are managed by clinical staff.

Overall, the end-to-end flow links edge sensing and inference, MQTT based telemetry, ThingsBoard storage and alarm generation, Cloud Run services, Cloud SQL metadata, and the web/mobile clients into a single operational pipeline. Figure 3.3 summarises the main steps in sequence form.

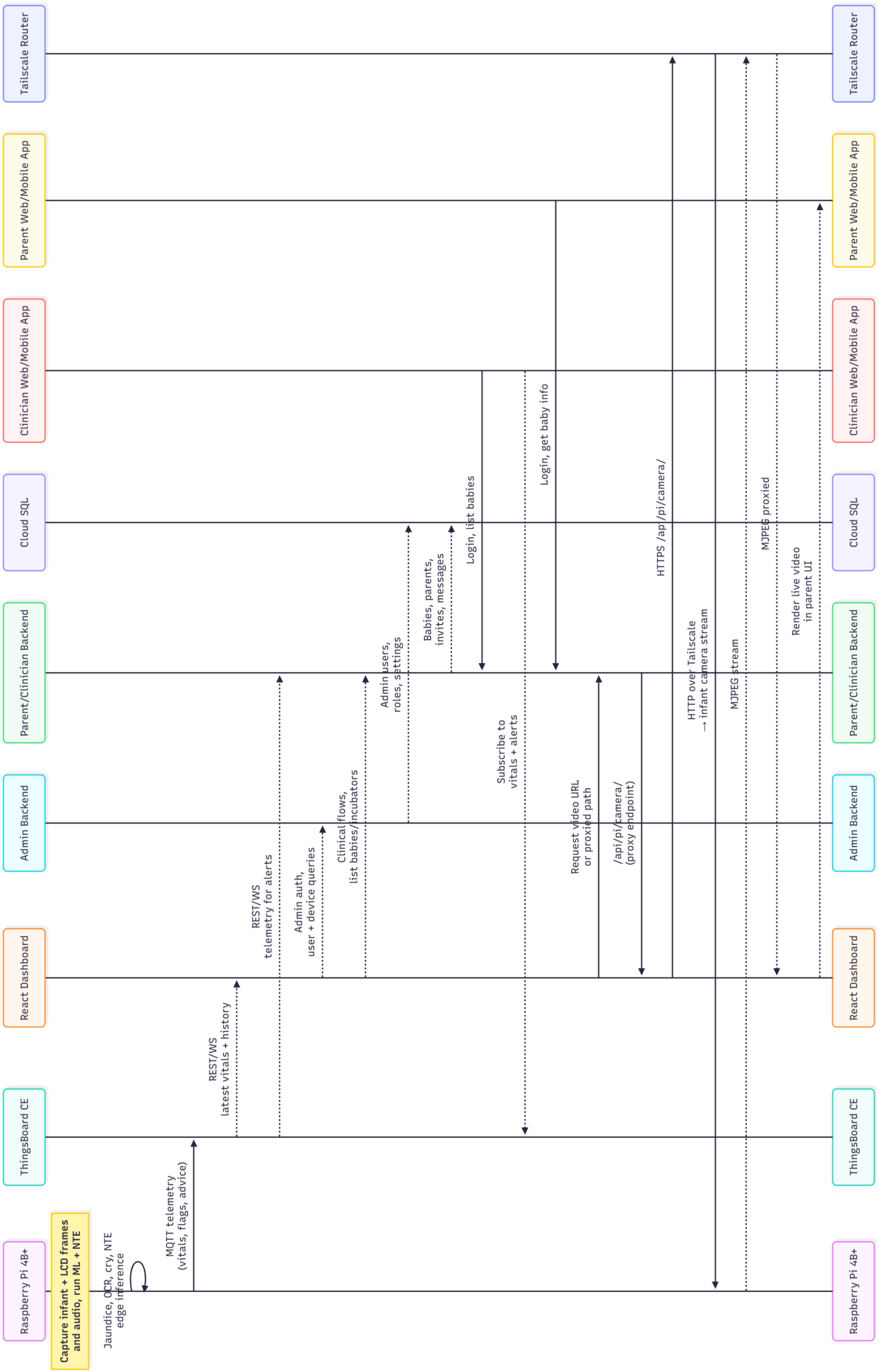


Figure 3.3: End-to-end data flow from edge inference to dashboards and mobile apps.

3.2 Machine Learning Models and Integration

The system incorporates three main machine learning components: (i) neonatal jaundice detection from infant images, (ii) infant cry detection and cry type classification from audio, and (iii) incubator LCD reading using computer vision and OCR. For each component, model development followed the same workflow: offline training and evaluation were carried out in a dedicated repository using curated datasets and notebooks; the trained artefacts were then exported in deployment-friendly formats (e.g., ONNX, PyTorch checkpoints, or scikit-learn serialisations) and finally integrated on the Raspberry Pi device as a microservice. Each microservice exposes a narrow HTTP interface and/or publishes telemetry, which simplifies integration and isolates failures. Figure 3.4 summarises this common workflow.

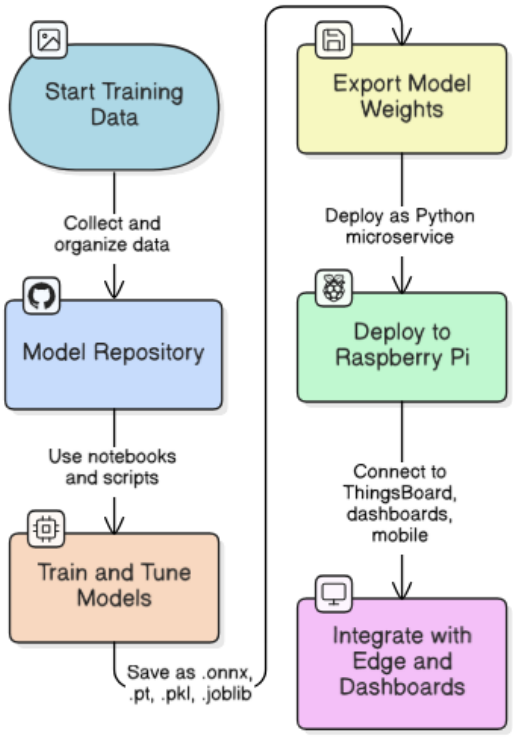


Figure 3.4: Common ML workflow.

The following subsections present each model in turn. For each component, the model architecture and training procedure are described first, followed by the edge deployment and integration within the overall pipeline.

3.2.1 Model 1 – Neonatal Jaundice Detection

The neonatal jaundice detector is implemented in the `Neonatal_jaundice_detection` repository [41]. The design goal is to obtain a binary classifier that remains computationally feasible on a Raspberry Pi device while preserving sufficient representational capacity to capture colour and illumination patterns associated with jaundice. After evaluating alternative backbones, MobileNetV3 Small was selected due to its

favourable accuracy efficiency trade-off for edge deployment [48].

3.2.1.1 Model architecture and training procedure

Training is performed using the notebook `jaundice-detection.ipynb` in the same repository [41]. The model is fine tuned on the Kaggle Jaundice Image Dataset [49], which contains colour images labelled as *jaundiced* or *normal*. Images are resized to 224×224 pixels, converted to RGB, and augmented using small rotations, horizontal flips, and brightness/contrast perturbations to reflect realistic variation in camera pose and lighting around incubators. The dataset is split into training and validation subsets (85/15) with a fixed random seed.

The `mobilenet_v3_small` network from `torchvision` is initialised with ImageNet weights and its classifier head is replaced with a single linear unit that outputs one logit for binary classification. The network is trained using `BCEWithLogitsLoss`, the AdamW optimiser, and a `ReduceLROnPlateau` scheduler driven by validation loss. Training is conducted for approximately ten epochs, after which validation accuracy, sensitivity, and specificity stabilise. The best-performing weights are exported as a PyTorch checkpoint (`.pt`) and as an ONNX model (`.onnx`) to support lightweight edge inference. Figure 3.5 summarises the training pipeline.

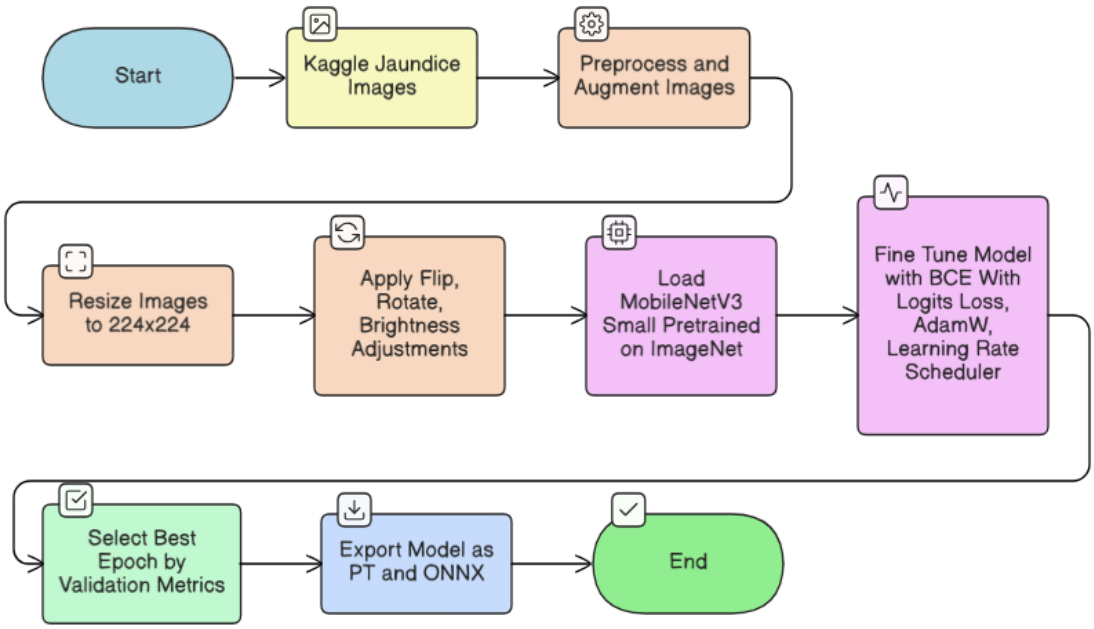


Figure 3.5: Jaundice model training and export.

3.2.1.2 Edge deployment and system integration

On the Raspberry Pi, the ONNX model is loaded using ONNX Runtime and exposed as an edge microservice via the Pi services repository [39]. Before invoking inference, two input gating steps are applied to each infant facing frame to reduce false alarms caused by unusable inputs. First, a brightness gate computes the mean grayscale

intensity; if the intensity is below a threshold the service returns a "too dark" status without running the CNN. For moderately dim frames, contrast limited adaptive histogram equalisation (CLAHE) is applied and the output is marked as reduced reliability.

Second, a baby presence gate checks whether the frame contains sufficient infant skin region for meaningful analysis. This gate is implemented in `baby_presence_detector.py` using a Haar cascade face detector combined with a YCrCb based skin mask. The face area ratio and the skin pixel ratio are compared against configured thresholds; frames that fail are labelled "no baby detected" and are not classified.

Frames that pass both gates are resized to 224×224, normalised using ImageNet statistics, and forwarded to the MobileNetV3 Small ONNX model. The output logit is converted to a probability using a sigmoid, thresholded to produce a binary flag, and returned together with the probability and a reliability indicator. The microservice publishes the latest outputs to ThingsBoard CE [50] (e.g., `jaundice_flag`, `jaundice_prob`), and the React dashboard displays the most recent status for each incubator/infant record [45]. Figure 3.6 illustrates the deployed inference pipeline.

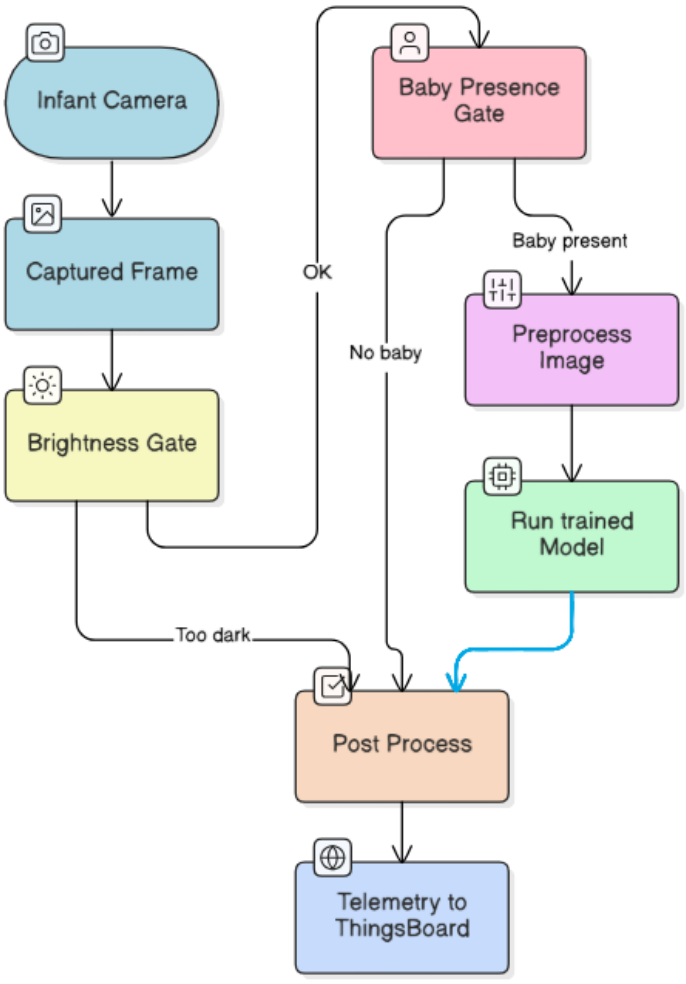


Figure 3.6: Edge jaundice inference.

3.2.2 Model 2 – Infant Cry Detection and Classification

The infant cry subsystem combines lightweight signal processing with supervised learning to support real-time operation on the Raspberry Pi device. Model training and evaluation are performed in the `Cry-Detection-Classification-Model` repository [42]. On the edge device, the runtime implementation is split into two cooperating components: `cry_detector.py` performs continuous cry event detection from the live microphone stream, and `cry_classification_service.py` classifies short cry segments and returns the predicted cry type together with confidence scores.

3.2.2.1 Model architectures and training procedure

Two related tasks are addressed. The first is binary cry detection (cry vs. non cry). The second is multi class cry classification, where an audio segment identified as a cry is assigned to a cry type category. For detection, the pipeline uses YAMNet [51] as a fixed embedding extractor followed by a Logistic Regression classifier. For classification, a feature engineering approach based on `librosa` [52] is combined with an ensemble classifier.

For cry detection, audio clips are resampled to 16 kHz and forwarded through YAMNet to obtain time indexed embeddings. The embeddings are aggregated across time by computing summary statistics (mean and standard deviation), producing a compact feature vector that is then normalised and optionally reduced using PCA. A Logistic Regression model is trained to discriminate cry vs. non cry. During evaluation, an empirical operating threshold (implemented as `DETECTION_THRESHOLD` in the repository) is selected to balance sensitivity against false positives.

For cry classification, the pipeline computes classical descriptors such as MFCCs, chroma features, Mel spectrogram energies, spectral contrast, tonnetz features, and related spectral statistics using `librosa`. These descriptors are summarised into a fixed length feature vector. Multiple candidate classifiers are evaluated and the final model is implemented as a voting ensemble, accompanied by a scaler, feature selection logic, and a label encoder. The trained artefacts are serialised (e.g., using `joblib`) and exported as deployment files (e.g., Logistic Regression and ensemble models together with their preprocessing objects) [42]. Figure 3.7 summarises the training workflow for cry detection and cry type classification.

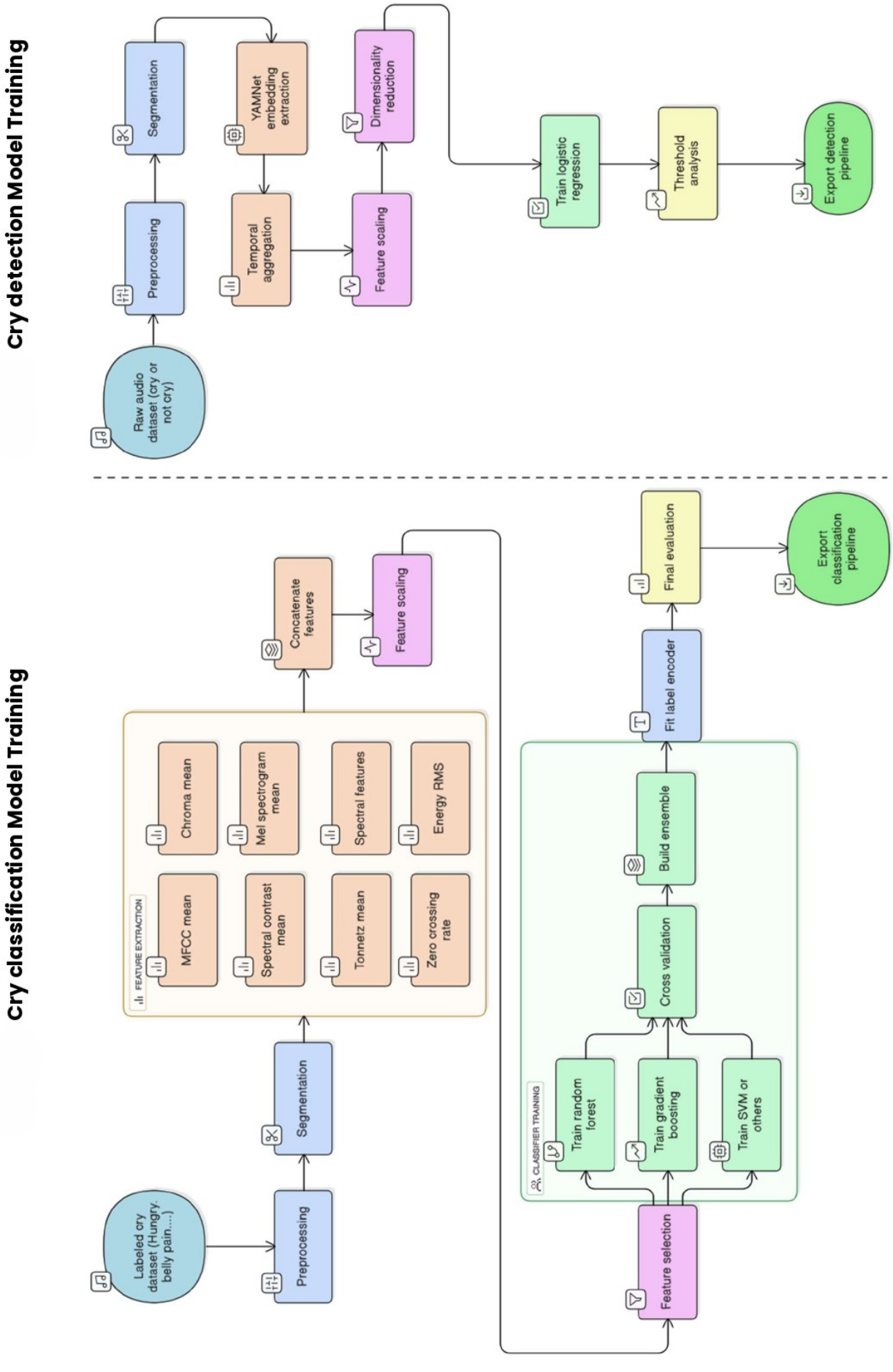


Figure 3.7: Cry Classification and Detection models training.

3.2.2.2 Edge deployment and system integration

On the Raspberry Pi, continuous monitoring starts in `cry_detector.py` [39]. Audio is captured from a USB microphone at 48 kHz using `PyAudio`. The detector processes short buffers (approximately one to two seconds) and computes the root mean square (RMS) amplitude and a short time Fourier transform (STFT). It then estimates the ratio of spectral energy within a cry relevant frequency band (300–2000 Hz). If both the RMS level and the band energy ratio exceed configured thresholds (tuned via parameters such as `sensitivity`, `cry_threshold`, and `noise_threshold`), the buffer is treated as a cry onset candidate.

When a cry onset is detected and no recording is in progress, the detector records a 5 s audio segment (configured by `RECORD_DURATION = 5`). The segment is then resampled to 16 kHz and posted to the local classification service `cry_classification_service.py`. The classification service loads the exported detection and classification artefacts from the cry repository [42], computes YAMNet embeddings and `librosa` features, and returns a JSON response that includes a binary decision (`is_cry`), confidence scores, a cry type label (when confidence exceeds a configured threshold), and the probability distribution over available cry classes.

The detector updates an internal state (e.g., current cry flag, timestamps, detection counts, and last predicted cry type). To avoid repeated classification during prolonged crying, a cooldown interval (configured by `classification_cooldown`) limits how frequently segments are reclassified. A summary of the current state is exposed through local HTTP endpoints and is also published to ThingsBoard CE as telemetry (e.g., `cry_detected`, `cry_audio_level`, `cry_total_detections`, and the most recent crytype label) [50]. These values are consumed by the React dashboard and mobile clients to display indicators and to trigger clinician facing notifications when repeated cry episodes are observed [45, 53]. Figure 3.8 summarises the on device cry monitoring pipeline from real-time detection to classification and telemetry publication.

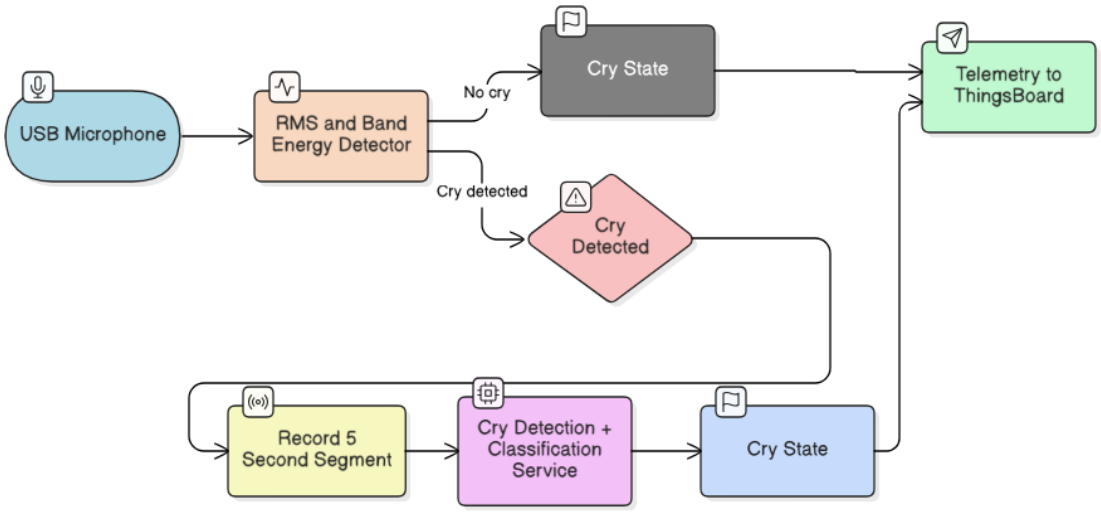


Figure 3.8: Cry subsystem on the Edge device.

3.2.3 Model 3 – Incubator LCD Display Reader

The incubator LCD display reader enables non invasive acquisition of incubator measurements by reusing the numeric values already shown on the incubator's built-in display, rather than installing additional wired sensors. Development of this component is carried out in the `Neonatal_incubator_displayReader` repository [40], while the production edge service (`lcd_reading_server.py`) is hosted in the Raspberry Pi services repository [39]. An auxiliary HTML based display simulator [54] is used to reproduce display layouts and generate additional test cases during development.

3.2.3.1 Model and training procedure

The LCD reader performs two main tasks: (i) detecting the display regions that contain relevant numeric fields and (ii) recognising the digits within those regions. For region detection, a YOLOv8n object detector [55] is trained using images of incubator displays annotated with bounding boxes around each parameter region (e.g., heart rate, SpO_2, skin temperature, humidity, and air temperature). Annotations are converted into YOLO format and organised into training, validation, and test splits (recorded in `artifacts/yolo/splits/incubator.yml`). Training is conducted using the Ultralytics implementation with a 640 pixel input resolution and standard augmentations, while performance is monitored using mean average precision on the held out set. The trained weights (e.g., `incubator_yolov8n_v4.pt`) are exported for deployment on the edge device.

For digit recognition, two OCR backends are evaluated. The initial implementation uses Tesseract OCR [56] with a fast configuration intended for real-time operation and an alternative high accuracy mode intended for batch processing; optimisations such as ROI caching and frame skipping are applied to improve runtime feasibility. However, in experiments with real incubator displays, Tesseract is sensitive to glare, reflections, and low contrast. Therefore, a second pipeline is implemented using EasyOCR [57]. Although EasyOCR is also model-based, its pretrained weights handle the display digit patterns and lighting variations more robustly in our tests. Based on repeated evaluations using both the display simulator [54] and NICU captured images, EasyOCR is selected as the default OCR backend for on device operation, while Tesseract is retained as an optional alternative for offline workflows. Figure 3.9 summarises the development pipeline for the LCD reader.

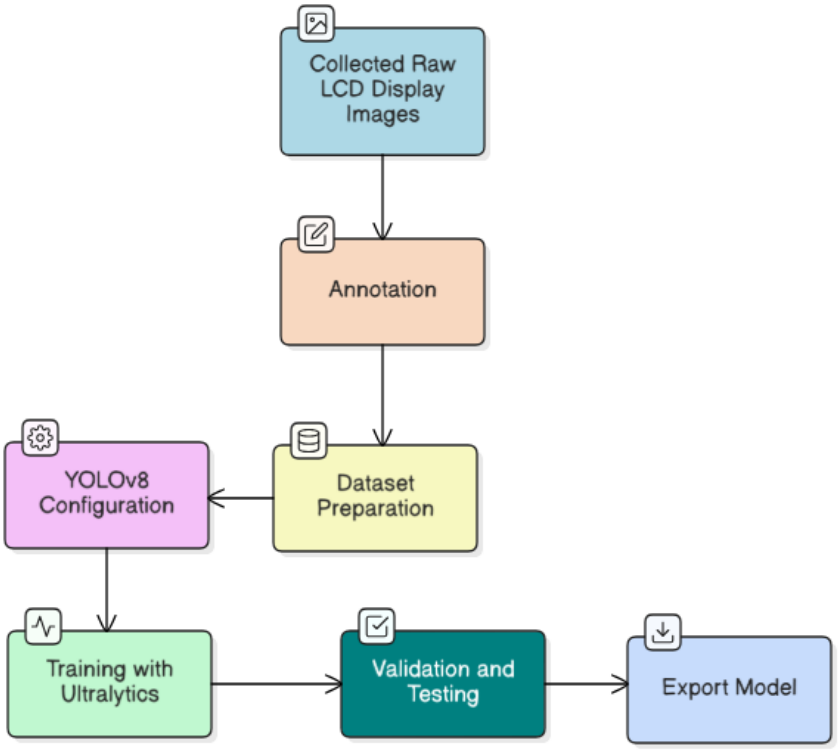


Figure 3.9: LCD reader Model development.

3.2.3.2 Edge deployment and system integration

On the Raspberry Pi, the LCD pipeline is implemented in `lcd_reading_server.py` [39]. The LCD camera is connected to `mjpg-streamer` [38], exposing an MJPEG stream (e.g., on port 8081). The server reads frames from the stream, decodes them using OpenCV, and runs the YOLOv8n detector to obtain bounding boxes for each parameter region. Each detected region is cropped and optionally preprocessed (e.g., grayscale conversion, normalisation, or contrast adjustment) before OCR. EasyOCR is then applied to extract the digit string, which is post processed by removing non numeric characters, enforcing expected formats (including decimal placement where relevant), and converting the result into numeric values.

Extracted parameters are validated using plausible ranges derived from neonatal guidance [47]. If a reading falls outside the acceptable range, it is marked invalid. The service maintains a cache of the last valid value for each parameter so that transient OCR failures (e.g., due to blur or reflections) do not immediately corrupt the displayed state. The capture-detect-OCR-validate cycle runs at a fixed interval (e.g., every five seconds), and the current cached readings are exposed through a `/readings` endpoint (port 9001). These readings are consumed by other edge services, including the NTE recommendation engine (Section 3.3), and are also published to ThingsBoard CE as telemetry when required.

From the dashboard perspective, the LCD reader is accessed through the Nginx reverse proxy on the Tailscale router VM, via the path `/api/pi/lcd/`, which maps to `/readings` on the Raspberry Pi [45, 46]. The React dashboard displays the returned

vitals in the clinical view, and the NTE widget uses the air temperature reading as a key input. The on device execution path is summarised in Figure 3.10.

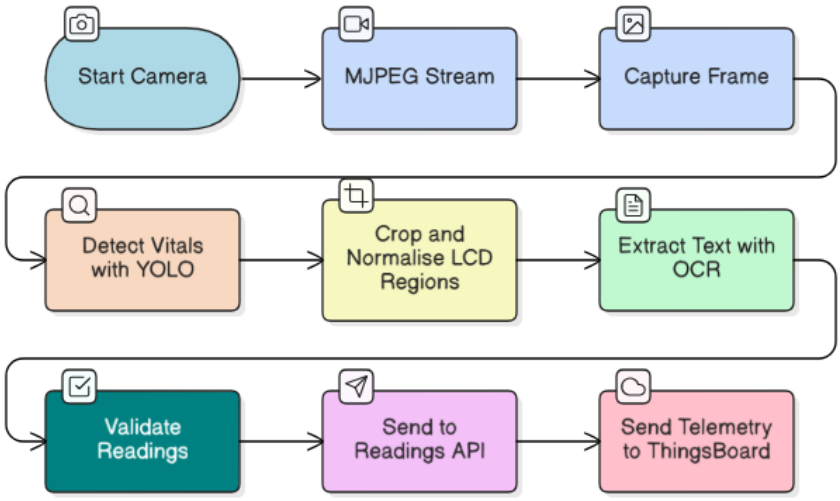


Figure 3.10: On Edge device LCD reader.

3.3 NTE Recommendation Engine

The NTE recommendation engine provides a rule based decision layer that maps incubator air temperature measurements to clinically motivated target ranges. Instead of learning targets from data, this component encodes the NTE temperature guidance published in Sri Lankan newborn care protocols [47] (summarised earlier in Table 1.1) into a small rules engine and exposes the logic through a simple API. Although it is not a machine learning model, it operates alongside the learned components in Sections 3.2.1 and 3.2.2 as part of the system's decision support layer. The implementation is maintained in a dedicated repository [43] and is wrapped on the Raspberry Pi to generate temperature related recommendations and alerts.

3.3.1 Rule Formulation and Implementation

The NTE engine is derived directly from the age and weight specific temperature table in the national newborn care guidelines [47]. The table specifies recommended incubator air temperature ranges across multiple age bands and weight categories (e.g., $< 1200\,\text{g}$, 1200–1500 g, 1501–2500 g, and $> 2500\,\text{g}$). To avoid manual lookup during operation, the engine converts these entries into a structured data representation and provides a lookup function that returns the correct temperature band for a given infant age (in hours) and weight (in grams).

This logic is implemented in `nte_rules.py` in the NTE recommendation engine repository [43]. The table is represented as a list of rule rows, each containing an age interval (`age_min_h` to `age_max_h`) and a set of weight band mappings to temperature ranges. A helper function (conceptually `find_nte_range(age_hours, weight_g)`) selects the

appropriate row based on age, then chooses the relevant weight band to return the target temperature interval. If an exact match is not available (e.g., very large ages), the engine falls back to the closest supported category. In addition, the implementation records reference ranges used elsewhere in the system, such as the newborn normothermia band (36.5–37.5 °C) and the typical target humidity band (approximately 50–60% RH) [47].

Once the NTE range is selected, the function `recommend()` compares the current measurements against the guideline band and generates structured advice. The function accepts infant age and weight together with current readings (air temperature and humidity, and optionally skin temperature). It outputs recommendation items containing a category (e.g., temperature or humidity), a severity label, and a concise message that can be displayed in the dashboard. When air temperature is below the lower bound (considering a tolerance), the engine recommends increasing the incubator setting; when it is above the upper bound, it recommends decreasing the setting. Similar comparisons are applied to humidity when a valid humidity reading is available. Figure 3.11 summarises how Table 1.1 is transformed into a code level representation and recommendation generator.

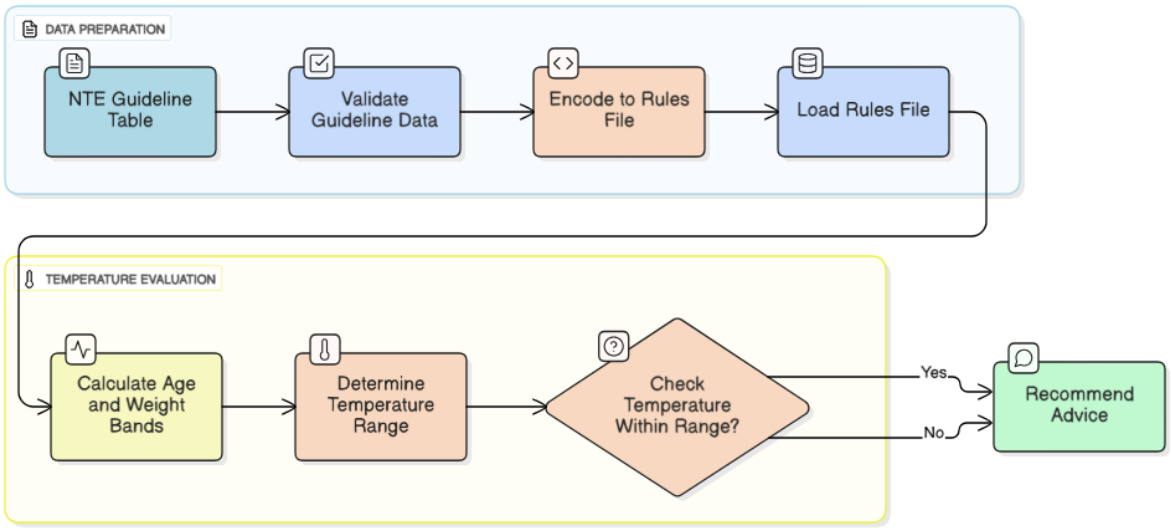


Figure 3.11: Encoding NTE Table (Table 1.1) into an NTE rule engine and recommendation generator.

3.3.2 Edge Integration and Cloud Connectivity

To expose the rules for practical use, the repository defines a lightweight API in `recommend_api.py` [43] using FastAPI. A `POST /recommendations` endpoint accepts a JSON payload containing infant age (hours), weight (grams), and the latest measurements (air temperature and humidity, with optional additional fields). The endpoint calls the underlying rule functions and returns the resulting advice items as JSON. This API is useful both for automated operation on the Raspberry Pi and for testing the rule behaviour with synthetic inputs during development.

On the Raspberry Pi, the NTE service wrapper in the edge device repository [39] periodically collects the current air temperature and humidity from the LCD reader, together with infant metadata (age and weight) from the application database. These inputs are passed to the NTE API (or the rules are invoked locally when running all services on the same device). The returned advice is then (i) summarised and published as part of MQTT telemetry to ThingsBoard CE [50] so it can be displayed in the dashboards, and (ii) made available through a local endpoint for debugging and future extensions. Figure 3.12 illustrates this integration path from LCD derived readings to NTE advice and cloud visualisation.

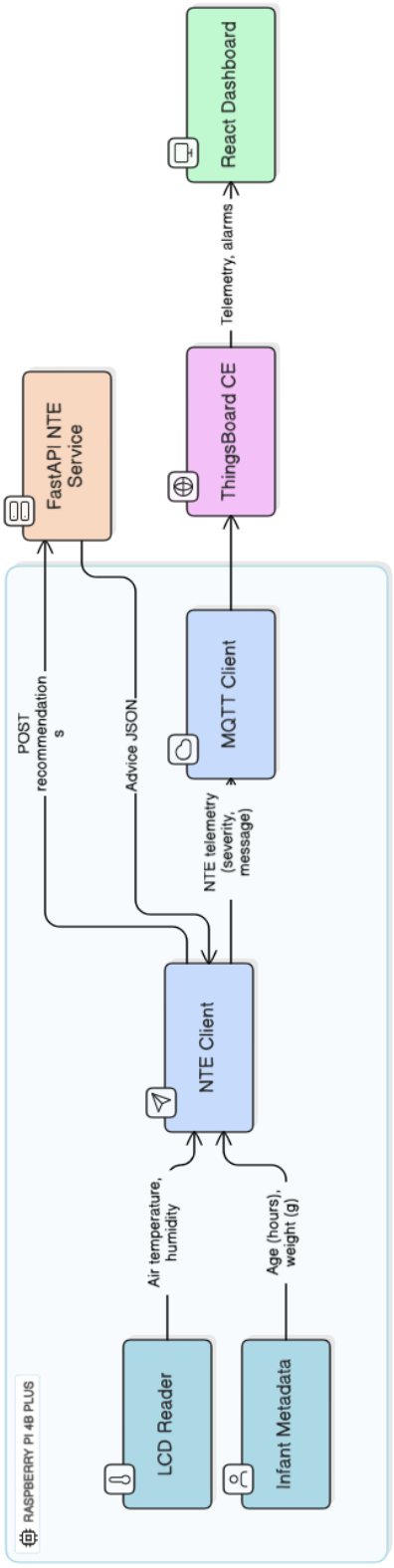


Figure 3.12: NTE integration.

From the user perspective, the NTE widget presents the recommended incubator air temperature range for the selected infant (from Table 1.1), the current measured air temperature, and a short recommendation such as “within target range” or “increase incubator temperature.” This supports day to day decision making by making guideline based targets available at the point of care without requiring manual lookup from

printed tables [47].

3.4 Web Dashboards

The web dashboards constitute the primary user interface for clinicians, parents and administrators. All web client code is implemented in the `react_dashboard` directory of the cloud integration repository [45], using React [58] for the frontend and a small service layer to communicate with the ThingsBoard CE APIs [50] and the Node.js backends. The dashboards are deployed as a single page application (SPA) on Google Cloud Run and are reachable through a single HTTPS endpoint. Internally, the SPA is partitioned into three role-based portals: a clinical dashboard for doctors and nurses, a parent dashboard, and an administrator dashboard. Role specific routing is implemented in `App.jsx`, and access is controlled via JSON Web Tokens issued by the backends [45]. While clinicians and parents also have corresponding Flutter mobile clients [53], the administrative portal is deliberately limited to the web dashboard and not exposed in any mobile application, reflecting its more sensitive functions (user and device management, configuration). Figure 3.13 shows how the web dashboards sit between the user's browser, the Node.js backends, ThingsBoard CE, and the Raspberry Pi edge services.

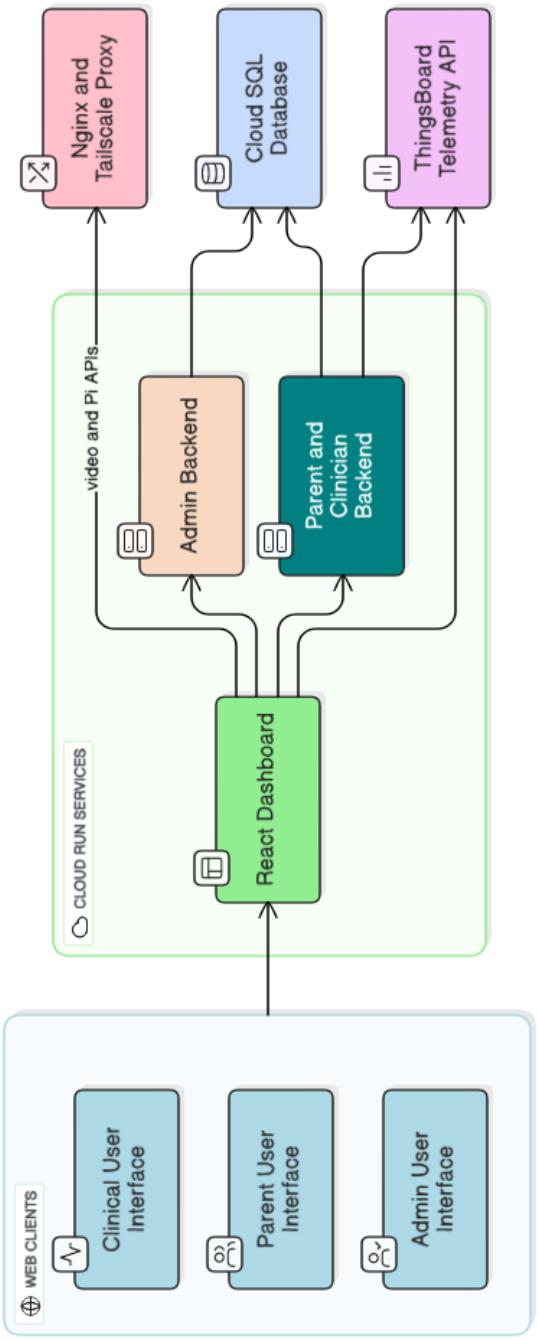


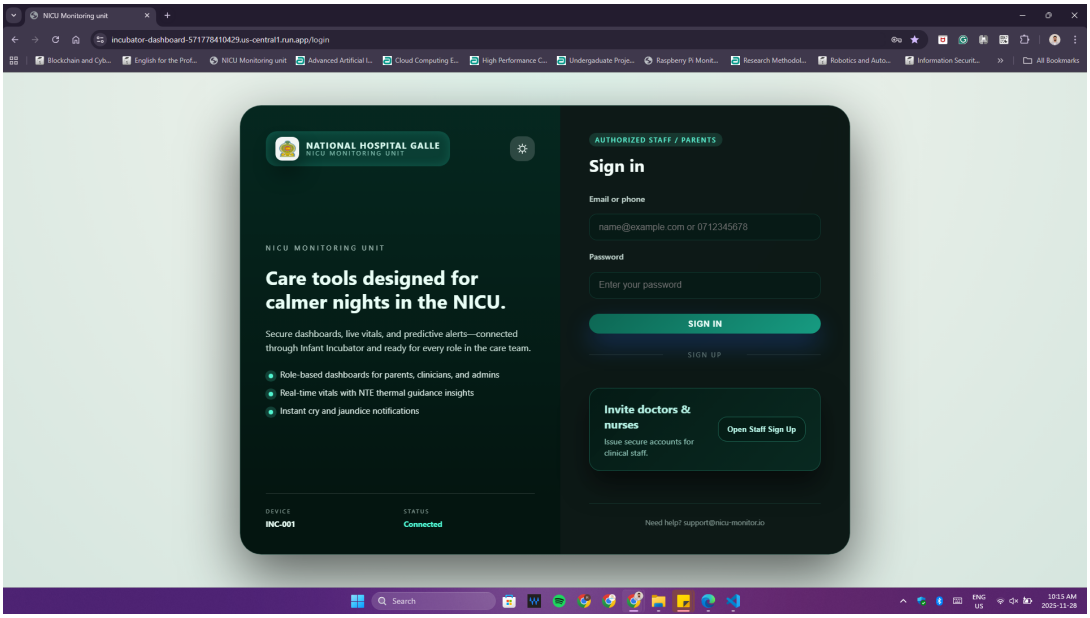
Figure 3.13: Architecture of the web dashboards.

3.4.1 Clinical Dashboard

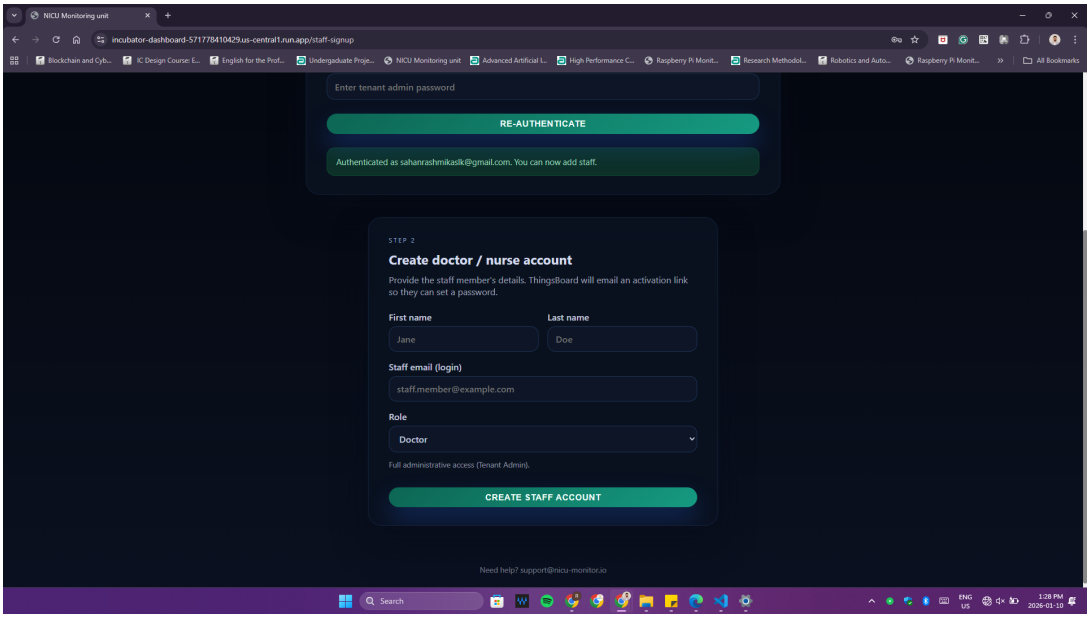
The clinical dashboard is the primary working interface for authorized NICU staff. It provides a table level overview of active incubators and a detailed view per infant. The left side table lists all registered incubators and linked infant metadata stored in Cloud SQL [45]. For each incubator, the dashboard displays a concise snapshot of the latest readings (e.g., heart rate, SpO_2, skin temperature, humidity, and air temperature) together with indicators for cry status and jaundice status. These values are obtained from the ThingsBoard telemetry store through the ThingsBoard REST API [50], rather than polling the Raspberry Pi directly.

When a clinician selects an incubator entry, the dashboard opens a detail panel showing per parameter status cards and time series trends. Charts are rendered using Chart.js [59] to help staff identify changes over time. The NTE widget is displayed in the same view; it shows the recommended air temperature band (Section 3.3), the current measured air temperature, and a short recommendation message. The live infant camera feed is embedded below the vitals. From the browser perspective, the stream is accessed through an HTTPS endpoint on the Cloud Run domain (e.g., `/api/pi/camera/`), which is forwarded through the Tailscale–Nginx proxy (Figure 3.2) to the `mjpg-streamer` service running on the Raspberry Pi [38]. This design keeps the edge device off the public internet while preserving a simple URL based integration for the React frontend.

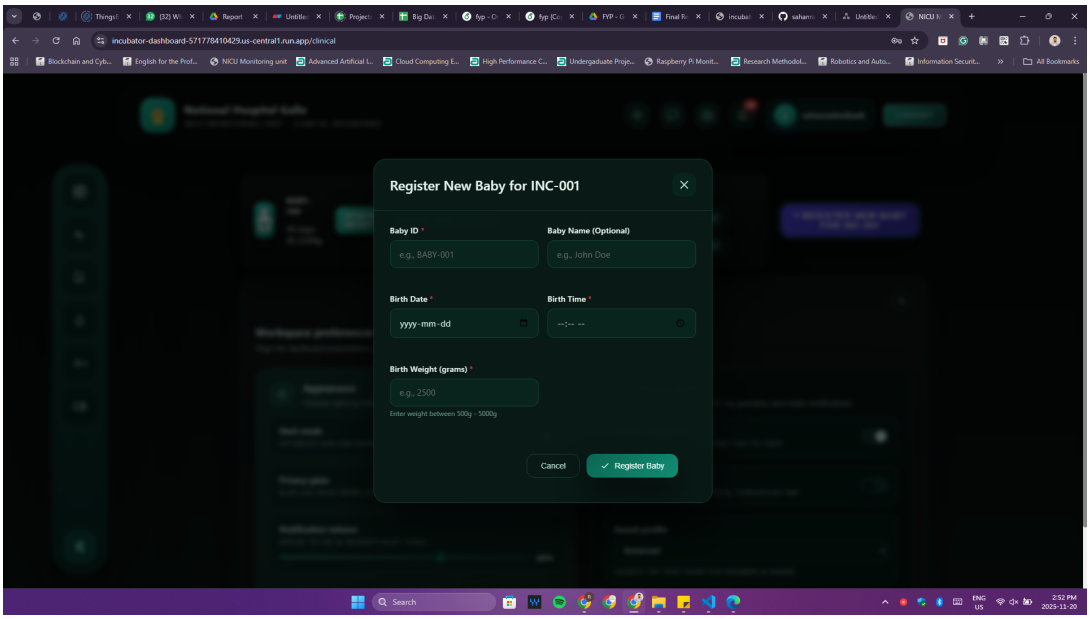
Figure 3.14 summarises the main clinical web interfaces. Figure 3.14(a) presents the clinician login screen and Figure 3.14(b) shows the clinician registration flow used during onboarding. Figure 3.14(c) illustrates the QR based parent linking workflow, where clinical staff can associate a parent account with a specific infant and incubator under controlled permissions. Figure 3.14(d) shows the main clinical dashboard view, while Figure 3.14(e) shows the notification panel used to review alarm events, system messages, and acknowledgement status.



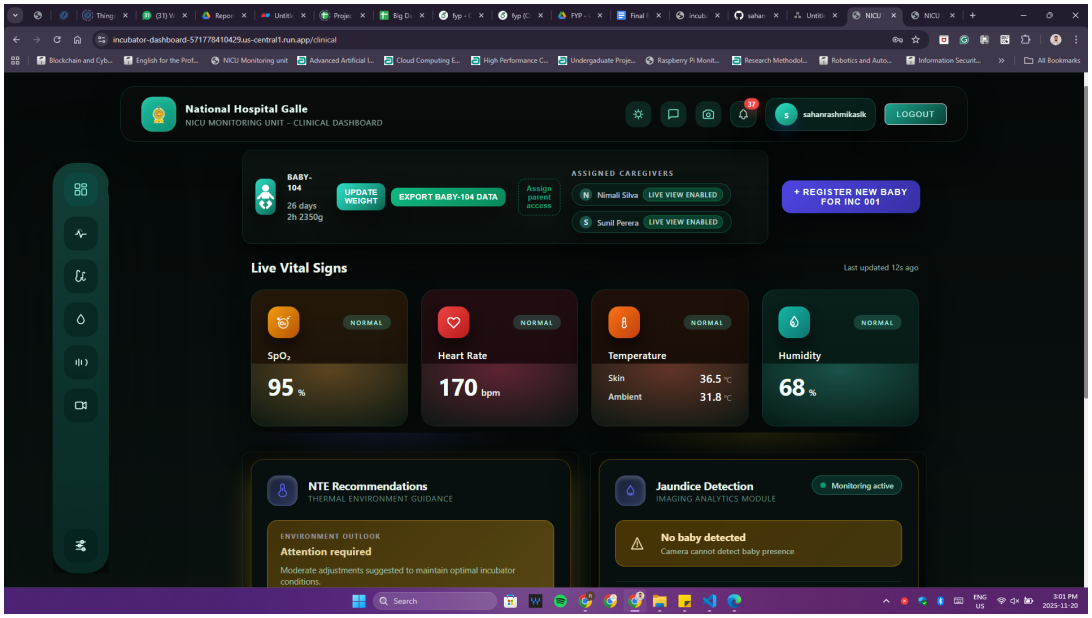
(a) Clinician / Parent / Admin login



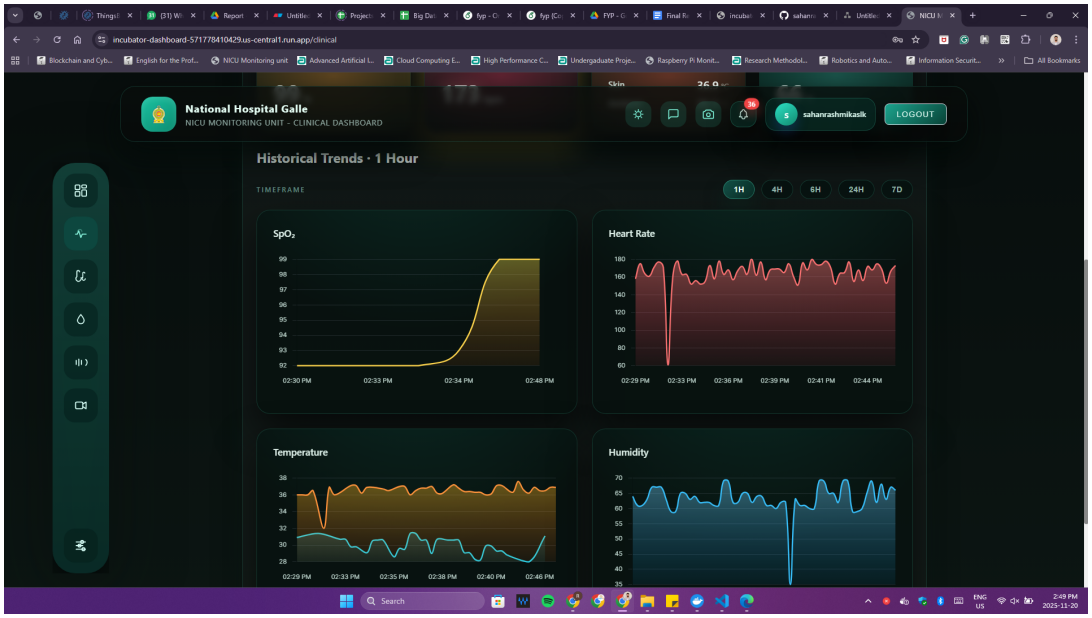
(b) Clinician registration



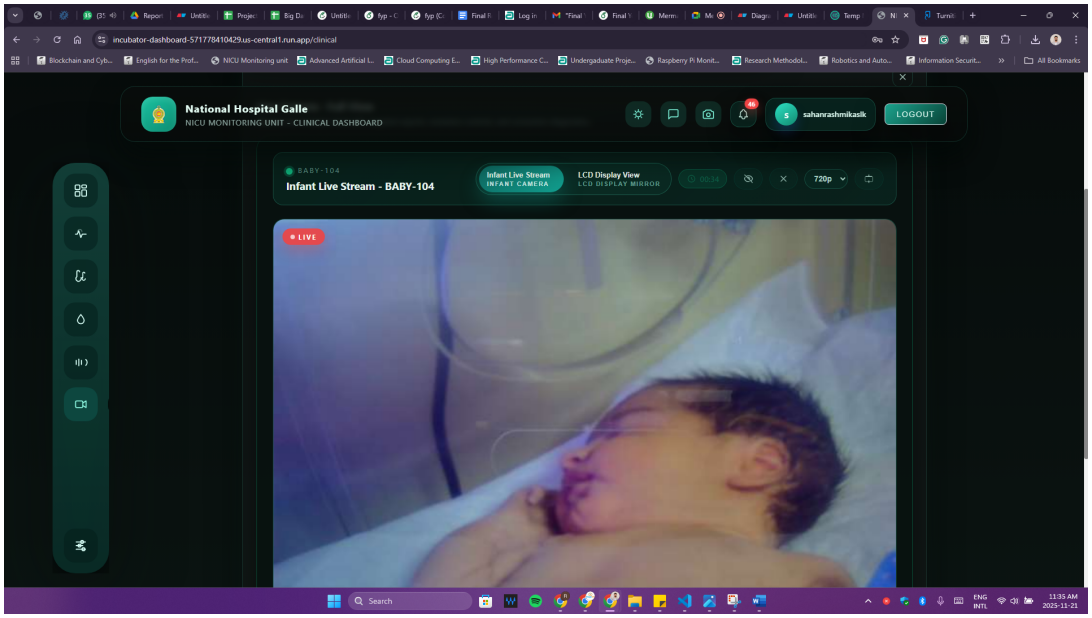
(c) Baby registration



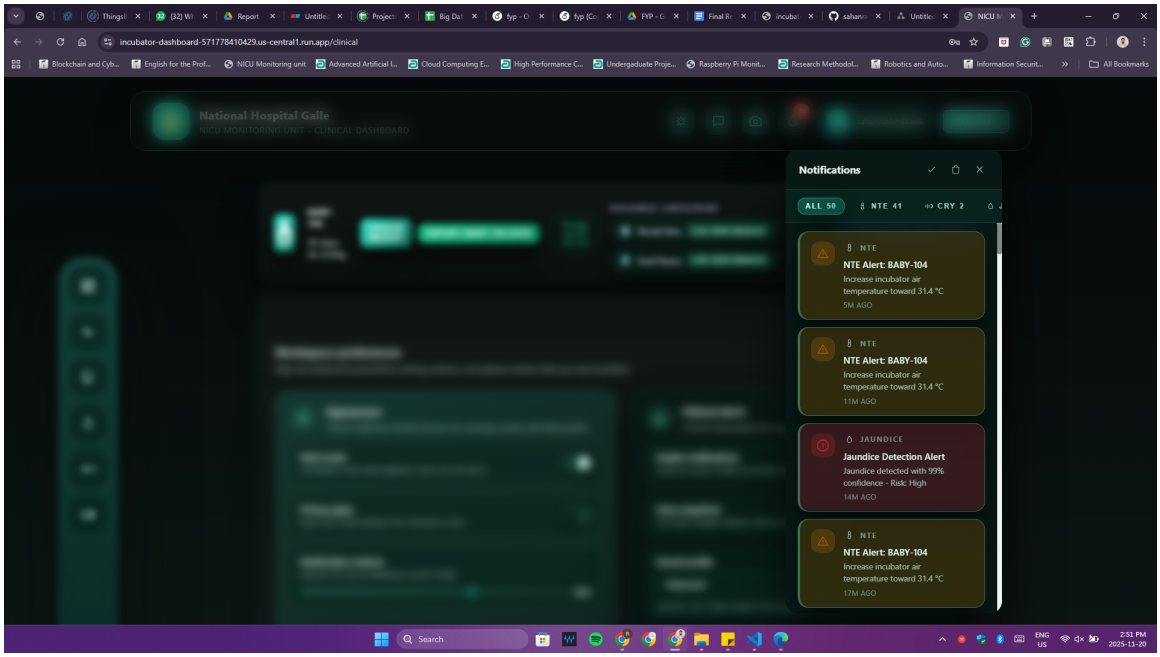
(d) Clinical dashboard - Summarize View



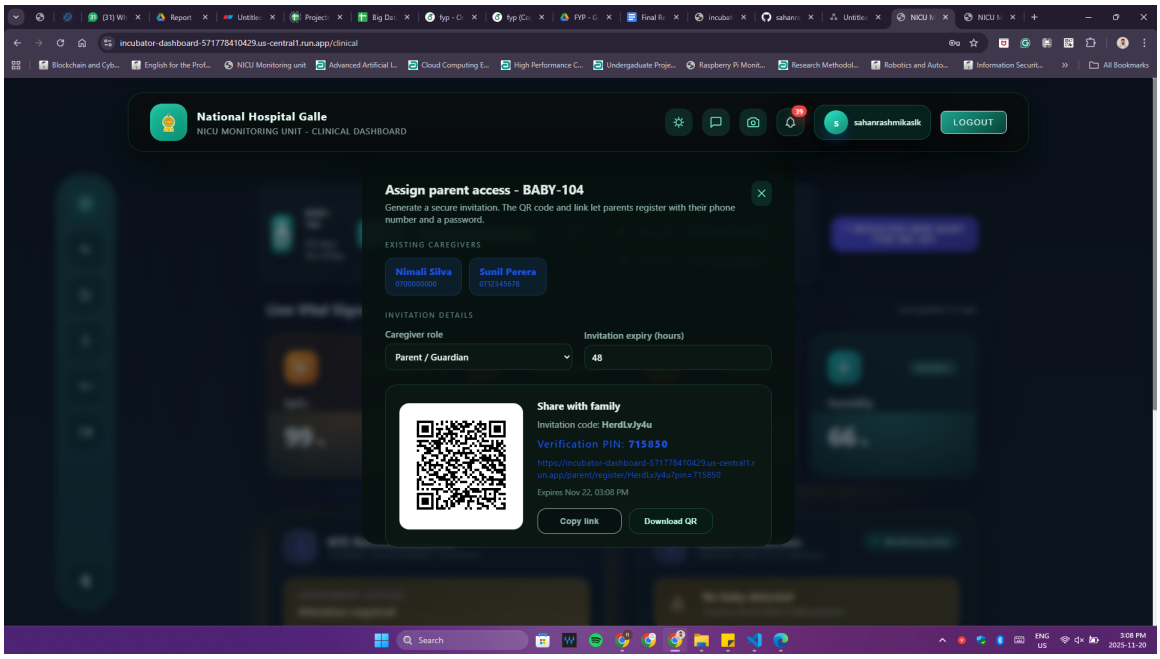
(e) Vitals Graph View



(f) Infant's Live Stream



(g) Notifications and alerts panel



(h) QR-based parent linking

Figure 3.14: Clinical web interfaces of (a) Login, (b) Registration, (c) Baby registration, (d),(e),(f) Dashboard view, (g) Notifications and alerts, and (h) QR linking.

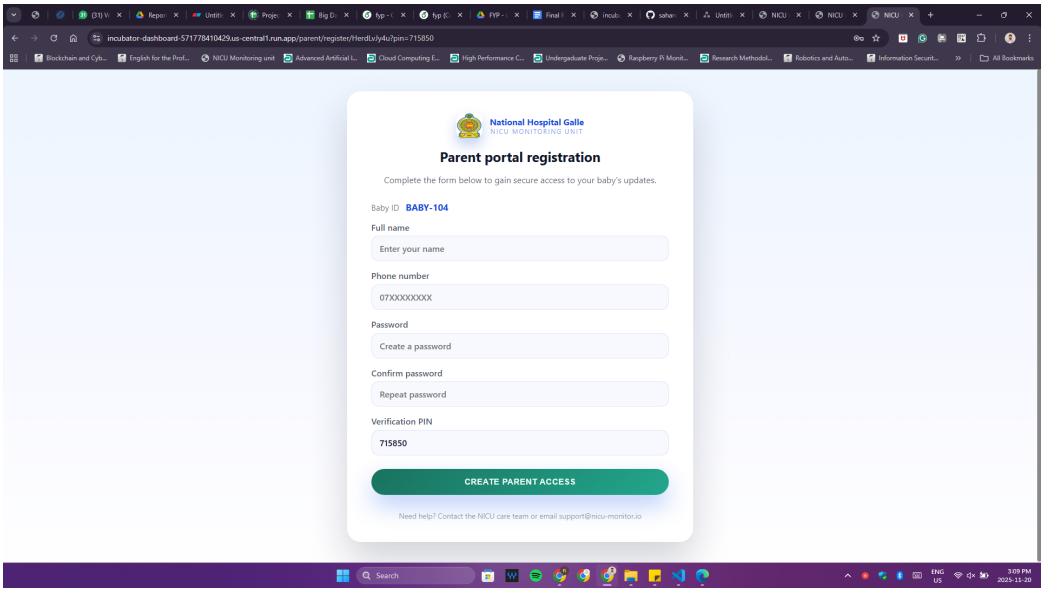
3.4.2 Parent Dashboard

The parent dashboard is implemented using the same React stack as the clinical interface, but it provides a simplified and carefully restricted view intended for non clinical users. Parent accounts are authenticated through the parent/clinician backend [45] using invitation codes and PINs that are linked to specific infants. After login, the parent typically sees only the infant(s) they are authorised to access. The main element of the dashboard is a large live video panel showing the infant in the incubator,

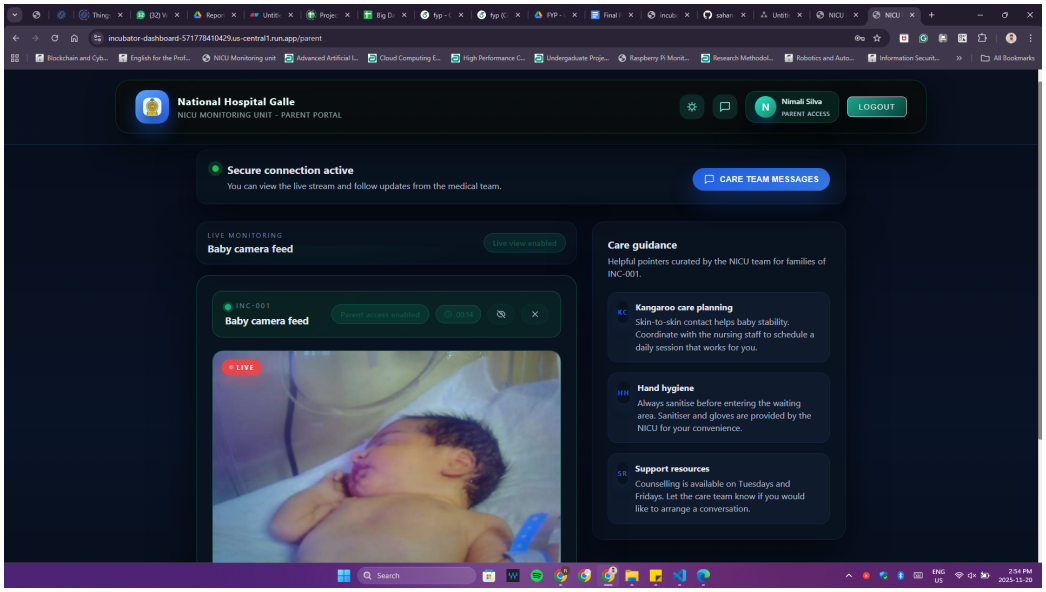
accessed via the proxied endpoint and rendered as an MJPEG stream. Numerical vitals and internal alarm states are not displayed to parents; instead, the interface provides a short, human readable status indicator. This design reduces the likelihood of misinterpretation of continuous numeric fluctuations without clinical context, while still providing transparency and reassurance through authorised visual access.

A messaging section is provided below the live stream to support communication between parents and clinical staff. Messages are handled by the parent/clinician backend, stored in Cloud SQL, and protected by role based access control [45]. The same message threads are visible to clinicians through their dashboard view and the NICU mobile application [53], allowing replies from either a workstation or a mobile device.

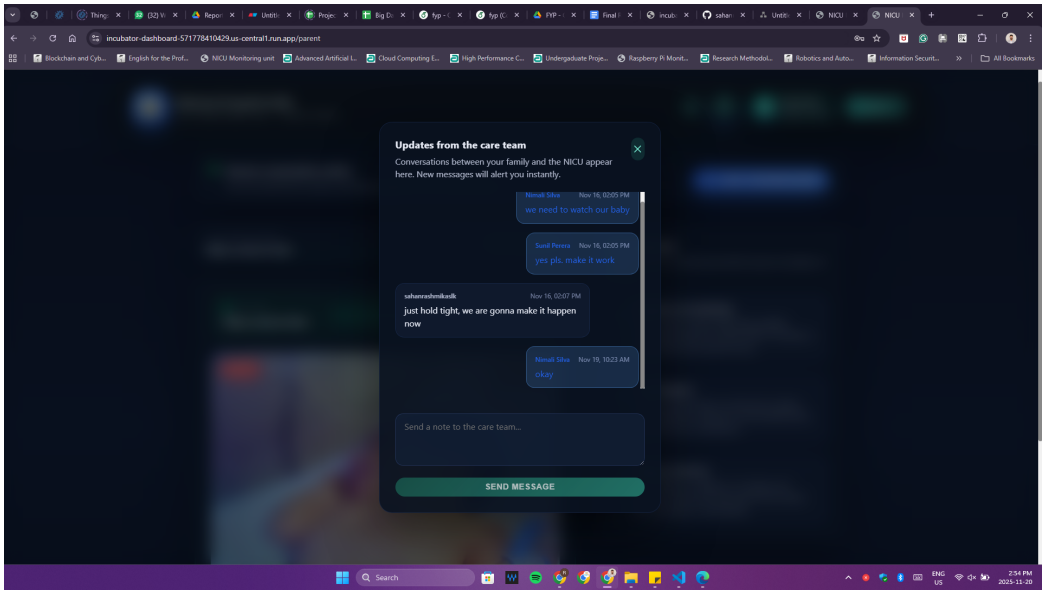
Figure 3.15 shows key interfaces related to parent access. Figure 3.15(a) presents the parent registration screen, Figure 3.15(b) illustrates the parent dashboard view focused on live video and simplified status, and Figure 3.15(c) shows the messaging interface used for parent clinician communication.



(a) Parent registration



(b) Parent dashboard view



(c) Parent–clinician messaging

Figure 3.15: Parent web interfaces of (a) Registration, (b) Dashboard view, and (c) Messaging.

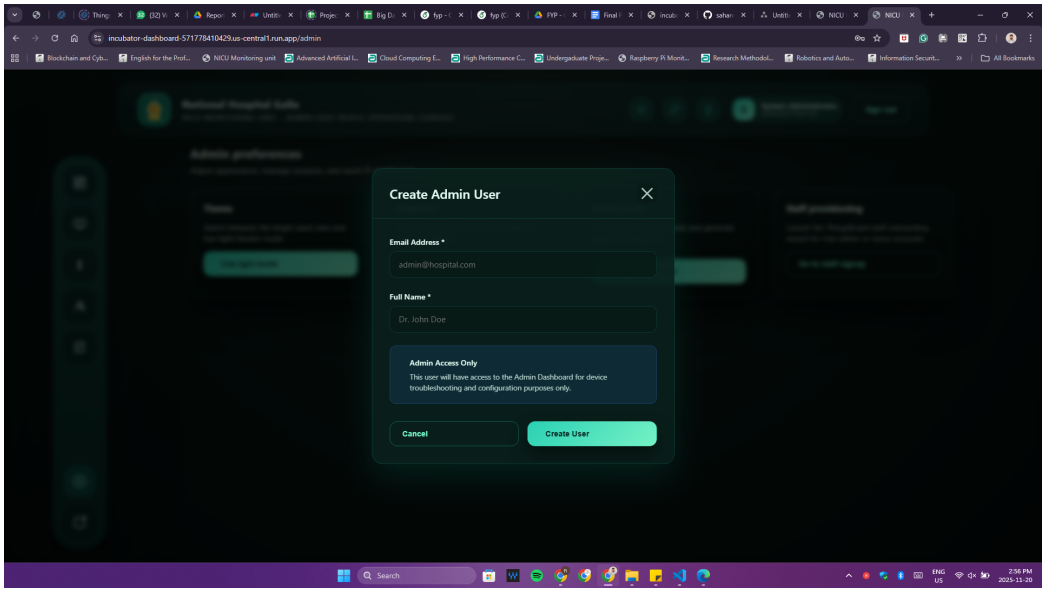
3.4.3 Administrator Dashboard

The administrator dashboard is intended for system configuration, user management, and edge device supervision rather than clinical monitoring. It is implemented as a separate portal within the same React SPA [45] and is accessible only via the web. An administrator view is intentionally not provided in the mobile application in order to reduce the attack surface and to ensure that privileged actions are performed from controlled workstations.

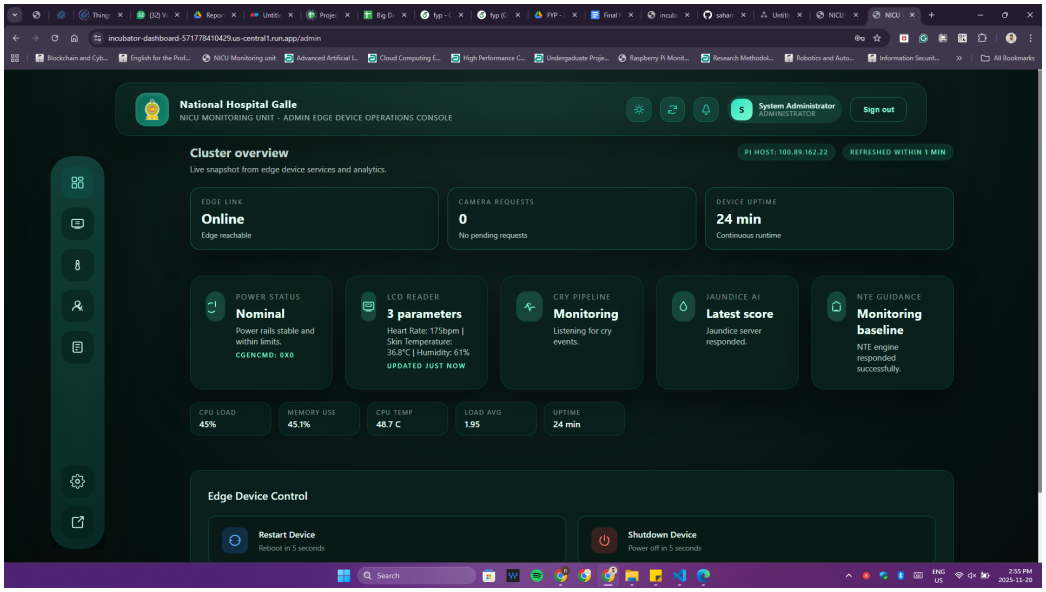
The admin portal functions as an edge device and access management console. Administrators can manage administrator accounts (e.g., create new admin users, reset credentials, and deactivate accounts); account and authorisation metadata are stored in Cloud SQL under the application's database schemas [45]. The same portal supports device onboarding by registering incubator entries, linking them to ThingsBoard device identities and Raspberry Pi nodes, and attaching operational metadata such as ward, bed number, and physical location. This mapping establishes the association between the physical incubator at the bedside and the logical device used for telemetry and dashboards.

To support operations, the admin portal also provides a basic health and service status view for each edge device. The Raspberry Pi exposes lightweight endpoints that report online status, recent telemetry timestamp, and resource usage. These endpoints are accessed through the Tailscale secured reverse proxy path [39, 46], allowing administrators to detect connectivity loss, unusually high load, or service failures without directly exposing the edge device to the public internet. This operational view is particularly important when scaling to multiple incubators, where manual checking at each bedside is impractical.

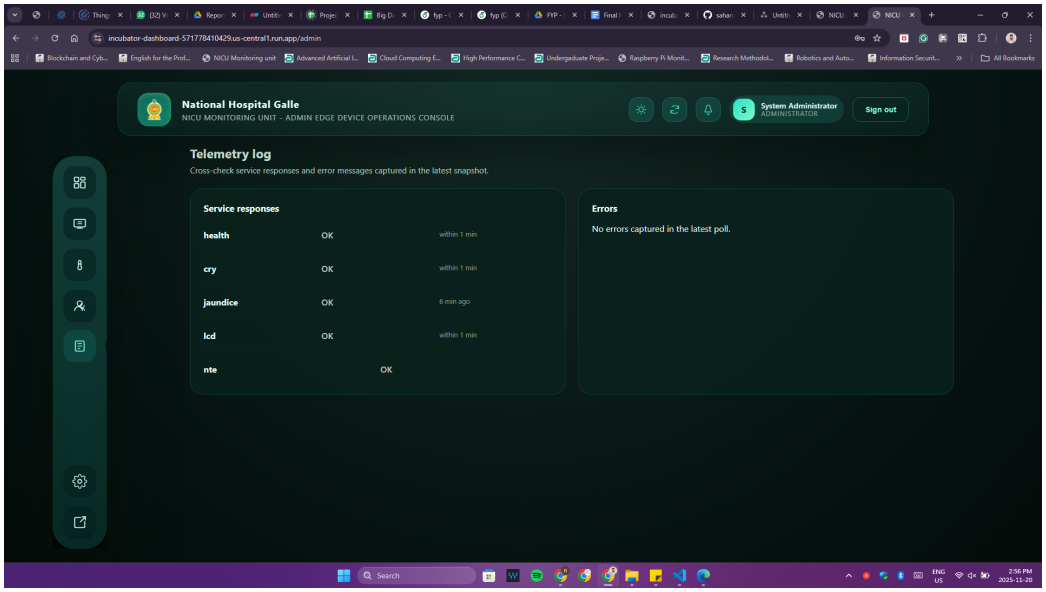
Figure 3.16 shows representative administrator interfaces. Figure 3.16(a) shows admin account creation, Figure 3.16(b) illustrates incubator and device onboarding (linking an incubator entry to a ThingsBoard device and edge node), and Figure 3.16(c) shows the edge device controlling health and service status view used to verify correct operation.



(a) Admin account creation



(b) Edge-device controlling



(c) Edge-device health monitoring

Figure 3.16: Admin web interface of (a) Account management, (b) Edge device controlling, and (c) Edge device health monitoring.

Overall, the three portals reuse a common React codebase and service layer but differ in the level of detail and the actions they permit. Clinicians view full vitals and decision support widgets such as the NTE panel; parents see authorised live video and simplified status with messaging; and administrators manage accounts, device mappings, and system health. This separation aligns with the underlying backend responsibilities and supports safer operation in a NICU context [45].

3.5 NICU Mobile App

In addition to the web dashboards described in Section 3.4, the system includes a cross platform mobile application for clinicians and parents. The application is implemented in Flutter [60] and maintained in a dedicated repository [53]. Although the repository snapshot does not include every minor UI adjustment made during the last development cycle, the deployed application follows the same architecture and provides the same core features described in this section. The overall design mirrors the web portals while enforcing role separation: clinicians access detailed monitoring and alerts, parents access a restricted view focused on authorised live video and messaging, and administrator functions remain web only.

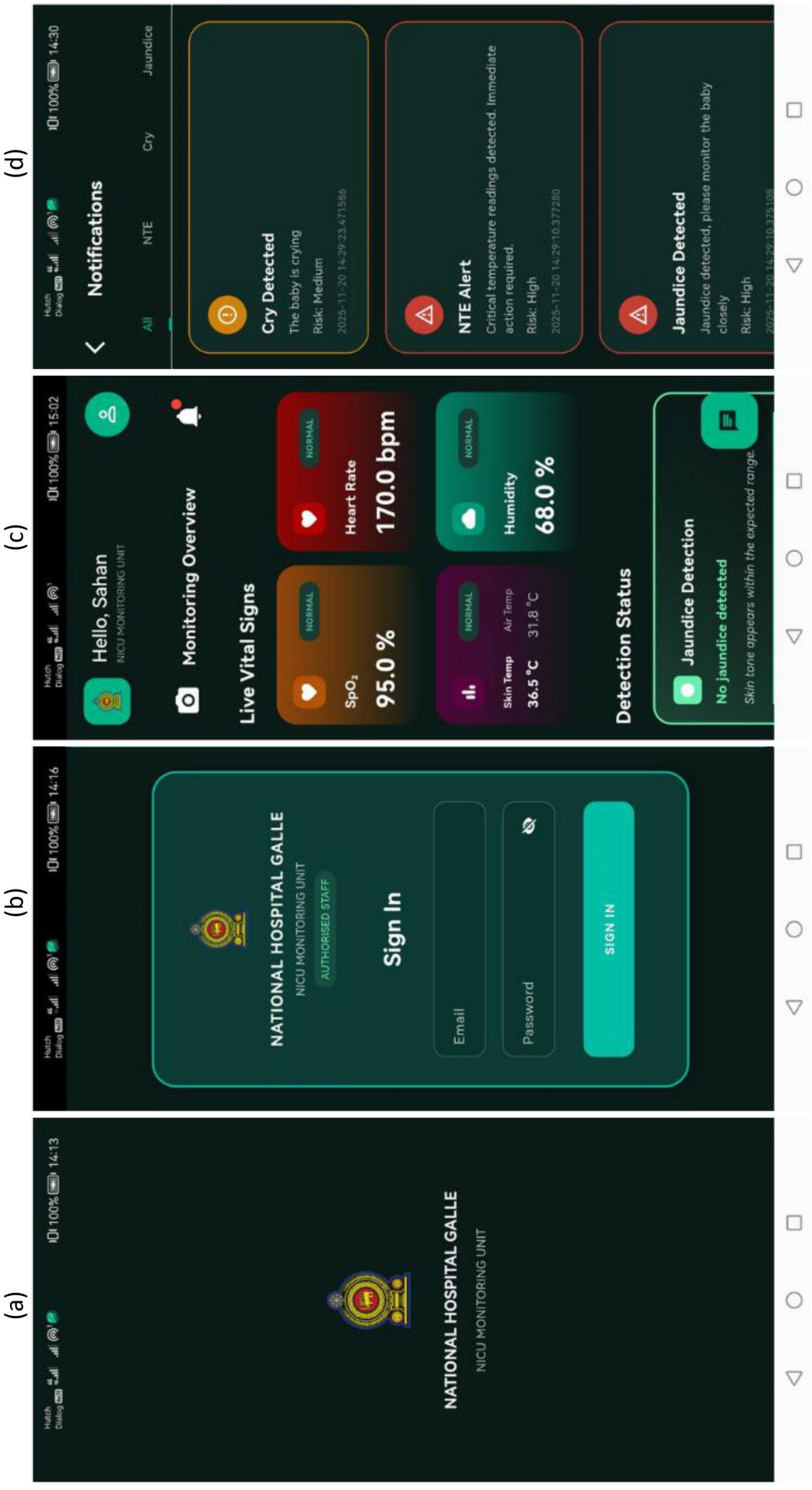


Figure 3.17: NICU mobile app user interfaces of (a),(b) Login, (c) Overview, (d) Notification view.

3.5.1 Architecture and Communication

The mobile application follows Flutter's standard layered design [60]. The UI is implemented using stateful/stateless widgets for screens such as login, infant list, infant detail, notifications, and messaging. Navigation uses Flutter routing, and the visual design is based on Material components for consistency across Android and iOS. Network interactions are encapsulated in a small Dart service layer that performs authenticated HTTP requests to the Node.js backends and, where needed, to ThingsBoard CE [50].

From the backend perspective, the app behaves similarly to the React SPA [45]. Authentication is performed against the parent/clinician backend using JSON Web Tokens (JWT). After login, the JWT is stored securely on device (e.g., using Flutter secure storage) and attached to subsequent API calls. Clinician accounts can retrieve assigned infants/incubators and telemetry summaries, while parent accounts can retrieve only infants to which they have been explicitly linked through invitation/PIN workflows. The backends use Cloud SQL for application metadata and ThingsBoard for telemetry access, ensuring that both web and mobile clients share the same source of truth.

Live video access follows the same secure approach used in the web dashboards. The mobile app does not connect directly to Raspberry Pi camera ports. Instead, it requests an authorised proxy URL from the backend, which points to the Nginx reverse proxy on the Tailscale router VM (Section 3.1.1). This proxy maps a path such as `/api/pi/camera/` to the Raspberry Pi's `mjpg-streamer` feed [38, 46]. The app renders the stream using an MJPEG capable viewer (e.g., a WebView or a dedicated stream widget), while the edge device remains isolated from direct public access.

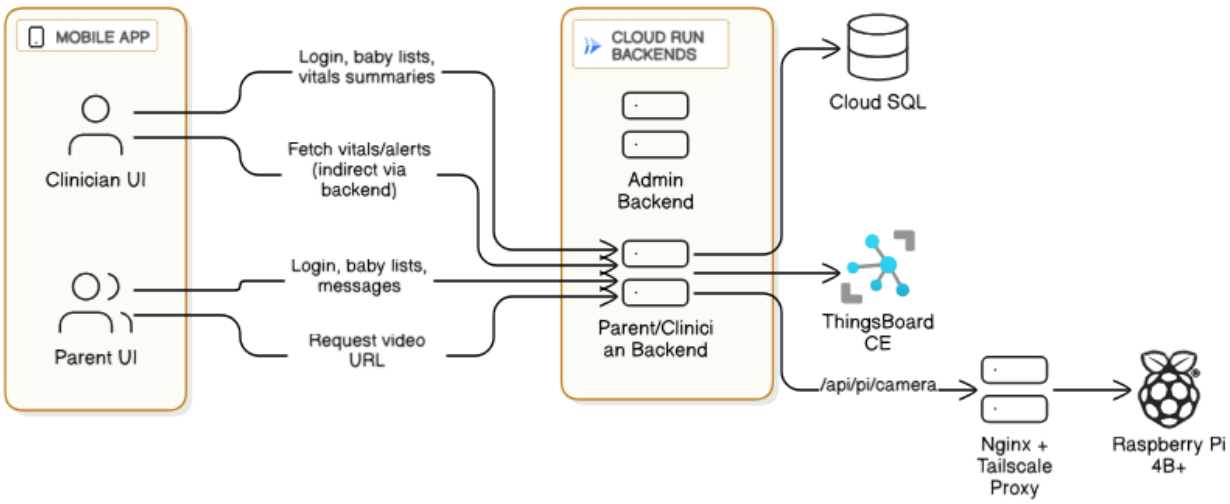


Figure 3.18: NICU mobile app architecture.

3.5.2 Clinician and Parent Workflows

The clinician workflow presents a ward oriented or assigned list of infants/incubators with compact summaries such as latest vitals, and visual flags for cry events, jaundice status, and NTE warnings derived from ThingsBoard telemetry. Selecting an infant opens a detail view showing numerical readings and recent trend summaries, together with the current NTE recommendation and recent alert history. Where policy and bandwidth allow, clinicians can also open a live video view to visually assess the infant without returning to a workstation.

The parent workflow is deliberately restricted. Parents authenticate using invitation/PIN mechanisms managed by the parent/clinician backend [45] and can view only the infant linked by clinical staff. The primary screen focuses on authorised live video and a simplified status indicator, while detailed vitals and internal alarm states remain hidden to prevent misinterpretation. A messaging screen enables parents to receive updates from the clinical team and send questions. Message threads are persisted in Cloud SQL and are visible to clinicians in both web and mobile interfaces [45, 53].

3.5.3 Alerting and Notifications

The mobile application is also designed as a notification surface, particularly for clinicians. Alerts can be raised when telemetry indicates repeated cry episodes, NTE deviations, or other abnormal events, and the app can also notify users of new messages. At the time of writing, notifications are primarily demonstrated in a controlled setting rather than a full clinical trial; however, the design supports integration with a push notification provider such as Firebase Cloud Messaging (FCM) [61]. The backend services already include event hooks for telemetry thresholding and ThingsBoard alarm state changes [45, 50], which can be extended to deliver platform specific notifications and to support future policies such as escalation paths, acknowledgement requirements, and per user preferences. Administrative actions (e.g., account disabling and device registration) remain confined to the web only admin dashboard (Section 3.4.3) and are intentionally excluded from the mobile application.

3.6 GCP and ThingsBoard CE Deployment Architecture

The Raspberry Pi device performs bedside sensing and inference, while the cloud back end provides persistent storage, device management, user authentication, and web/mobile access. The deployment is hosted on GCP [62] and consists of four main components: (i) a self hosted ThingsBoard Community Edition (CE) instance for IoT telemetry [50], (ii) Cloud Run services hosting the React dashboard and Node.js backends [63], (iii) a Cloud SQL PostgreSQL database for application metadata [64], and (iv) a Compute Engine VM that runs Tailscale and Nginx to securely proxy selected edge-device endpoints [46]. Deployment scripts and configuration files are maintained

in the `incubator_monitoring_with_thingsboard_integration` repository [45].

3.6.1 Core Cloud Components

3.6.1.1 ThingsBoard CE (IoT layer)

The IoT layer is centred on a ThingsBoard CE instance running on a Compute Engine VM. ThingsBoard is deployed as a Docker service on a Debian based host, and dashboards, rule chains, and device profiles are versioned as exports in a separate repository for reproducibility [44]. Each incubator is represented as a ThingsBoard device with a unique access token. Raspberry Pi nodes publish telemetry via MQTT using these tokens, while the application layer consumes telemetry through ThingsBoard REST/WebSocket APIs [50].

3.6.1.2 Cloud Run (web and API layer)

The React dashboard and the Node.js backends (admin and parent/clinician) are deployed as separate Cloud Run services [63]. Container images are built from Dockerfiles stored in the integration repository [45]. The React frontend is served as a static SPA through an Nginx container, while the backend services expose REST APIs for authentication (JWT), invitation/PIN onboarding, user and infant metadata, messaging, and integration logic.

3.6.1.3 Cloud SQL (metadata store)

Persistent application data are stored in a Cloud SQL PostgreSQL instance [64]. Logical separation is maintained using dedicated databases/schemas for administrator accounts, parent/clinician accounts and invitations, and incubator/device mappings. Cloud Run services connect using the Cloud SQL connector/proxy configuration, keeping the database off the public internet and restricting access to authorised services only.

3.6.1.4 Tailscale router VM with Nginx proxy (edge access bridge)

A dedicated VM runs Tailscale in subnet router mode and hosts Nginx as a reverse proxy [46]. This VM joins the same Tailscale network as the Raspberry Pi devices and forwards only selected endpoints (e.g., live video and specific edge APIs) over the encrypted overlay network. This approach prevents direct public exposure of Raspberry Pi ports while keeping the client side integration simple (single HTTPS domain with a stable path namespace).

Figure 3.19 summarises the cloud-side components and how they connect to edge devices.

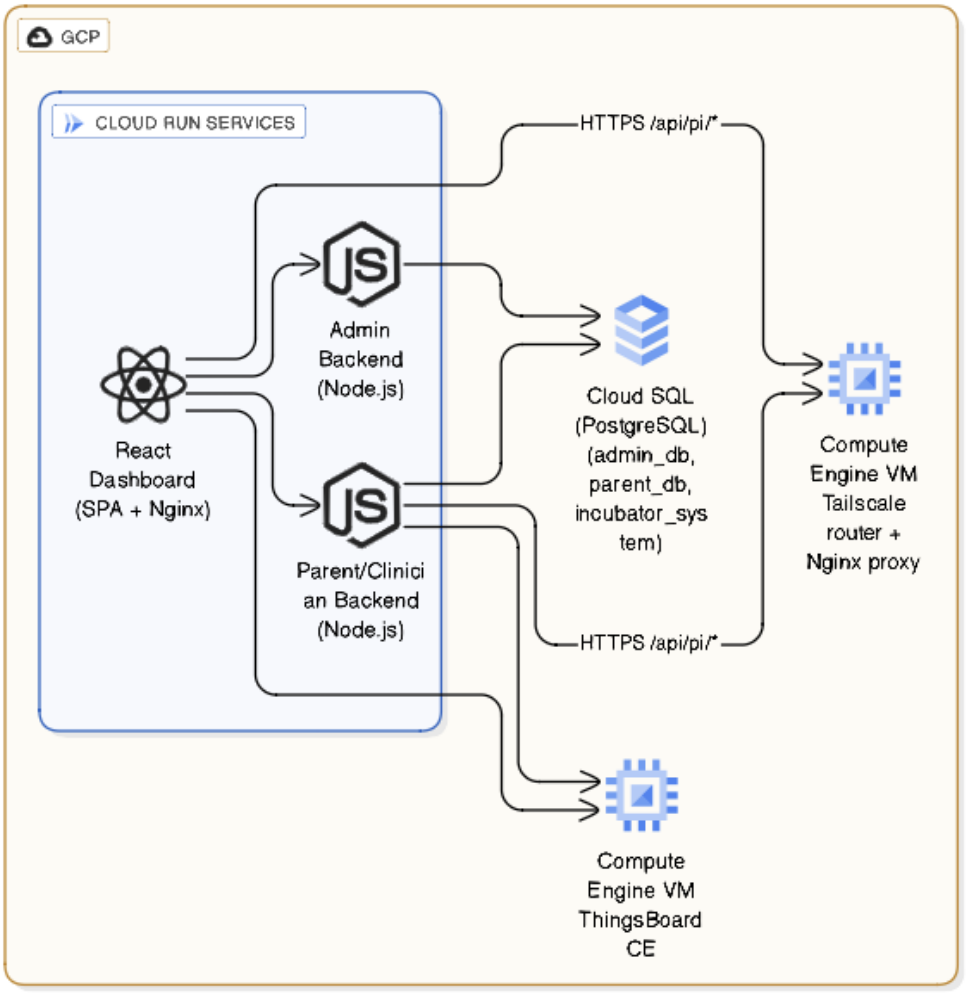


Figure 3.19: GCP deployment overview.

3.6.2 Network Topology and Security Considerations

The network design aims to provide secure remote access for clinicians and parents while keeping edge devices private. Cloud Run services act as the public HTTPS entry points for web and mobile clients [63]. Cloud SQL is configured without a public IP and is accessed privately via the Cloud SQL connector from Cloud Run [64]. ThingsBoard CE is hosted on a Compute Engine VM, and telemetry is published to it using device specific access tokens [50].

Edge devices do not accept inbound connections from the public internet. Instead, Raspberry Pi nodes join a Tailscale network, and the router VM provides a controlled bridge for selected services [46]. Nginx on the router VM exposes a small set of proxied routes under the `/api/pi/` namespace, mapping each route to a service on the Raspberry Pi via its Tailscale IP. Table 3.1 lists representative route mappings used in the deployment.

Table 3.1: Representative reverse-proxy routes for edge services

Public path (Cloud Run domain)	**Forwarded service (via Tailscale+Nginx)**
`/api/pi/camera/`	`http://<Pi-Tailscale-IP>:8080/` (infant camera MJPEG stream)
`/api/pi/lcd/`	`http://<Pi-Tailscale-IP>:9001/readings` (LCD reader)
`/api/pi/jaundice/`	`http://<Pi-Tailscale-IP>:8887/` (jaundice service)
`/api/pi/cry/`	`http://<Pi-Tailscale-IP>:8888/` (cry service API)
`/api/pi/snapshot/`	`http://<Pi-Tailscale-IP>:8090/` (snapshot/health endpoints)

With this arrangement, the dashboards and mobile app interact only with public HTTPS endpoints, while all edge traffic traverses the encrypted Tailscale overlay and remains behind the proxy. Firewall rules on the cloud side restrict inbound access to required services only, and device credentials are managed per incubator through ThingsBoard access tokens and application level authentication [45, 50]. Figure 3.20 illustrates the relationship between external clients, Cloud Run services, cloud internal services, and the Tailscale connected edge layer.

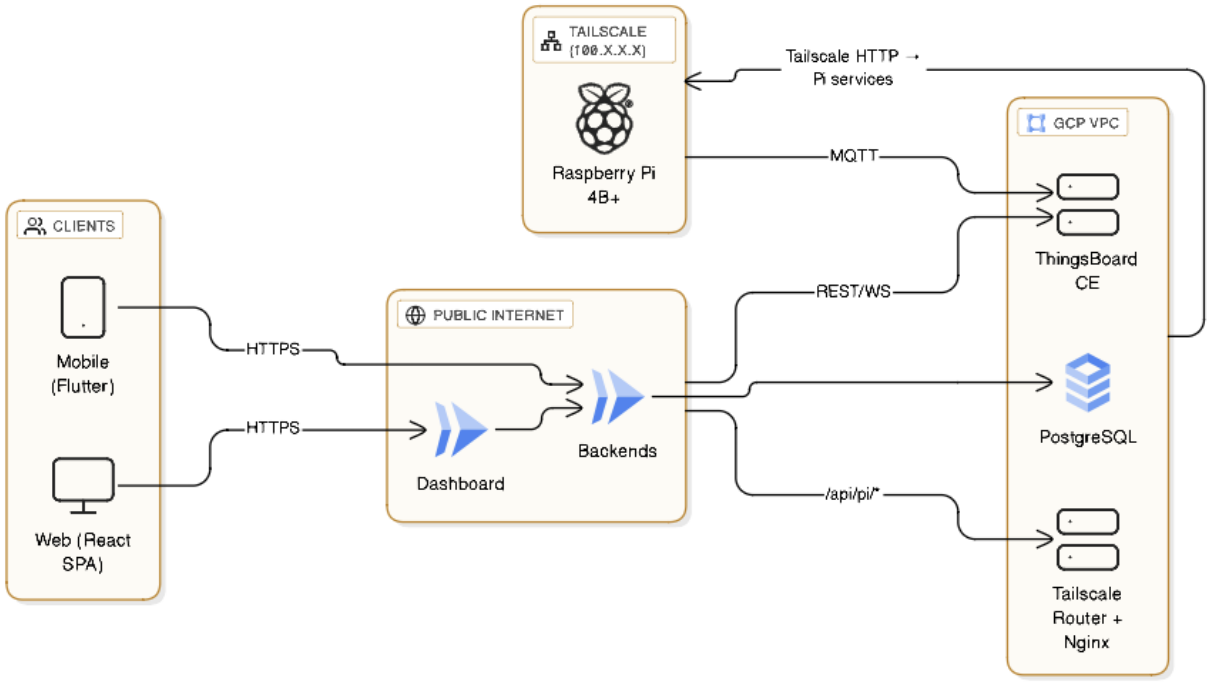


Figure 3.20: Network topology linking clients, Cloud Run, ThingsBoard CE, Cloud SQL, and Raspberry Pi devices through a Tailscale router VM.

3.6.3 Deployment Workflow

Deployment is kept reproducible through containerisation and configuration versioning. The dashboard and backend services are built as Docker images from the integration repository [45] and deployed to Cloud Run, with runtime configuration provided via environment variables (e.g., ThingsBoard endpoints, JWT secrets, Cloud SQL connection identifiers). ThingsBoard dashboards, device profiles, and rule chains are

exported and stored in the ThingsBoard configuration repository [44] so that a fresh instance can be reconstructed if required.

The router VM is provisioned following Tailscale subnet router guidance [46]. Setup scripts install and authenticate Tailscale, then start Nginx with proxy rules stored in the integration repository [45]. Raspberry Pi devices are provisioned using scripts and `systemd` service definitions in the edge-device repository [39], including installation of dependencies and configuration of device credentials for telemetry publication. This workflow has been sufficient to reproduce deployments across test environments and to keep cloud and edge configurations aligned during iterative development.

Chapter 4

Results and Evaluation

4.1 Overview

This chapter reports the observed behaviour of the NeoCare prototype under lab and NICU style test conditions. First, we evaluate the three on device intelligence components: the neonatal jaundice model, the two stage infant cry pipeline, and the incubator LCD reading pipeline. Next, we summarise end to end validation of the complete edge cloud path, including latency, reliability, and expected failure modes. Finally, we present initial usability findings from informal demonstrations with doctors, nurses, and parents, focusing on onboarding, monitoring views, alerts, and messaging.

4.2 Performance of Machine Learning Components

This section describes practical performance of the machine learning components as implemented in this project, using the datasets and evaluation scripts maintained in each model repository. The intention is to report what we actually observed during development and deployment (including failure patterns such as low light images, glare, or noisy audio), rather than to claim exhaustive benchmarking against all related work.

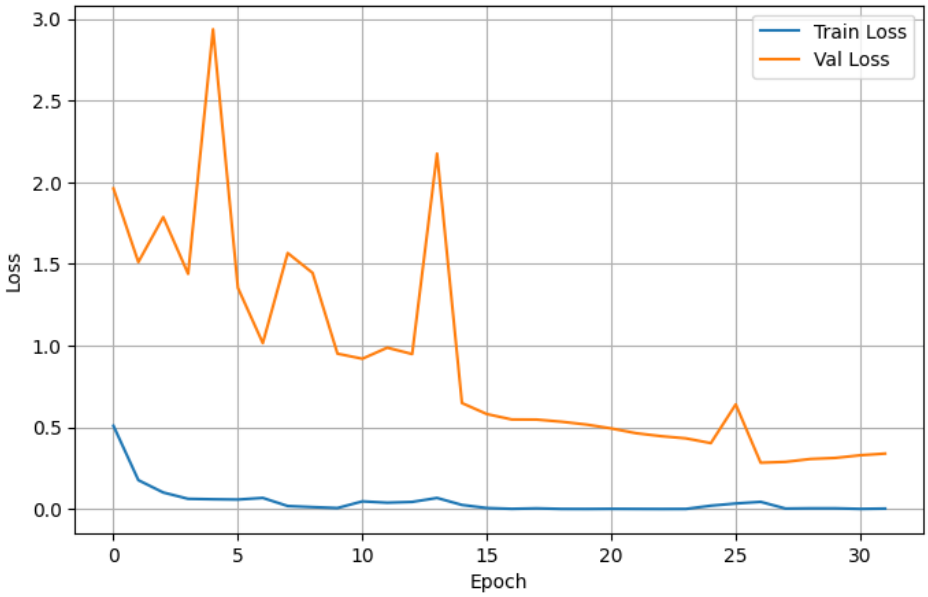
4.2.1 Jaundice Detection Model

The neonatal jaundice model (Section 3.2.1) is implemented in the `Neonatal_jaundice_detection` repository [41]. In the final version, we used a dual head multitask setup with a MobileNetV3 Small backbone [48]: one head predicts jaundice (Normal/Jaundice), while the second head estimates image quality. The training data source was the Kaggle Jaundice Image Data set [49], and training/evaluation were performed using the notebook and scripts provided in the repository.

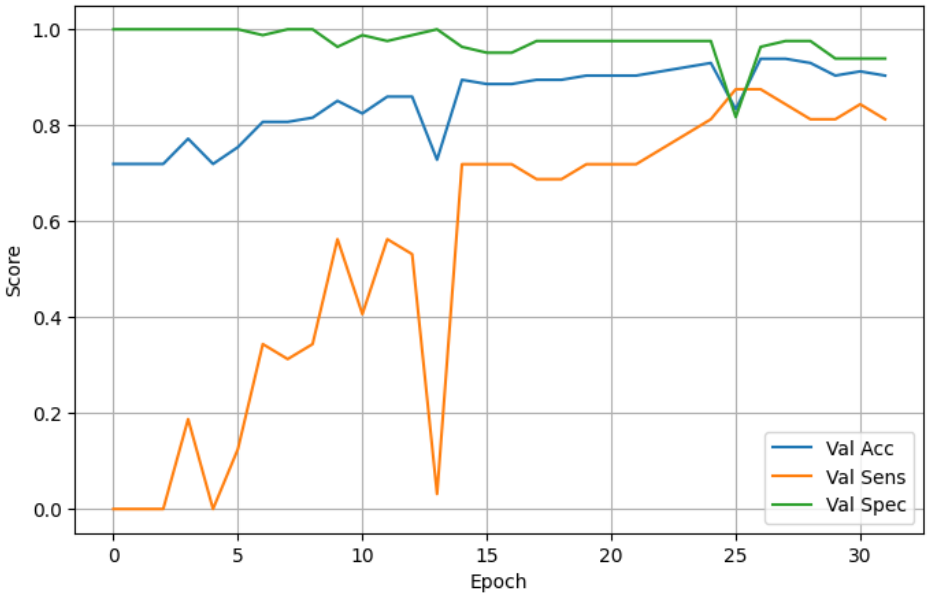
4.2.1.1 Training behaviour

Figure 4.1 shows the typical training dynamics (loss reduction and validation metrics over epochs). In practice, training was run with early stopping, and the "best" check-

point was selected based on validation behaviour. The plotted validation metrics are useful mainly to confirm that learning stabilised and did not collapse due to class imbalance or poor optimisation.



(a) Loss



(b) Val. metrics

Figure 4.1: Jaundice training curves.

4.2.1.2 Evaluation metrics

For a held out evaluation split of $n = 114$ images, the model was evaluated at a decision threshold of 0.80 (selected via threshold sweep in the repository evaluation workflow). Table 4.1 summarises the measured classification performance and the confusion counts. The ROC and Precision Recall curves are shown in Figure 4.2.

These plots were generated by the repository evaluation script and reflect the final exported model used for edge deployment.

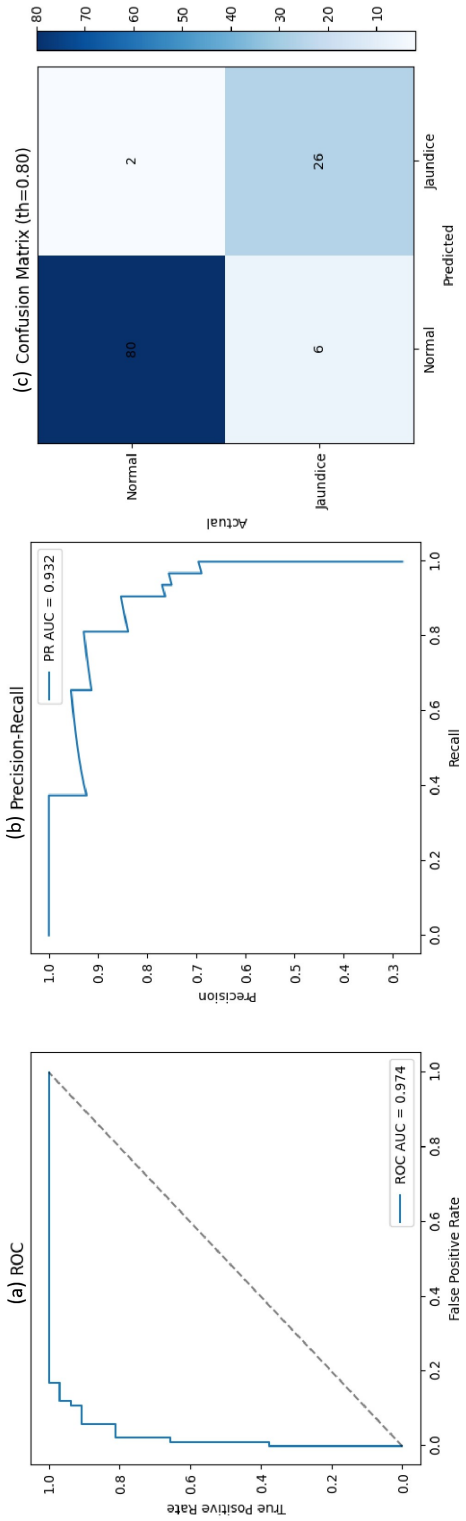


Figure 4.2: (a) ROC, (b) PR, and (c) confusion matrix.

Table 4.1: Jaundice performance on the evaluation split (threshold = 0.80).

Metric	Value
Accuracy	0.930
Sensitivity (TPR)	0.813
Specificity (TNR)	0.976
ROC AUC	0.974
PR AUC	0.932

On the evaluation split at threshold 0.80, the model achieved an accuracy of 0.930, sensitivity of 0.813, and specificity of 0.976 [41]. In this context: (i) *accuracy* summarises overall correctness on the held out split, but it can hide class imbalance effects, (ii) *sensitivity* (true positive rate) indicates how often jaundice cases are correctly flagged, which matters because missed jaundice can delay attention, (iii) *specificity* (true negative rate) indicates how often normal cases are correctly rejected, which matters to reduce unnecessary alarms and avoid "alert fatigue" in a NICU workflow. The ROC AUC (0.974) and PR AUC (0.932) provide threshold independent summaries of separability.

The trade off is visible in these numbers, the high specificity indicates few false positives (useful for reducing unnecessary alerts), while the lower sensitivity implies that some true jaundice cases can be missed, particularly if a strict threshold is chosen. In practice, for a clinical facing decision support tool, a higher sensitivity operating point may be preferred if the workflow can tolerate more false positives. However, acceptable performance ultimately depends on local clinical policy and would require a proper clinical validation study, which is beyond the scope of this prototype.

In the edge pipeline, the quality head is used as a reliability signal before surfacing a jaundice alert to clinicians. This reduced "noisy" outputs in frames that are clearly unsuitable (very dark frames, glare, or motion blur), and it also made it easier to explain why a given frame was rejected (i.e., low quality score) instead of producing a misleading probability. In the overall system, the jaundice output is treated as a decision support indicator rather than a diagnostic result, and it is shown alongside the raw incubator readings and other alerts.

The model was trained using a public Kaggle dataset [49], not a Sri Lankan NICU specific dataset. Therefore, performance may shift under local lighting, camera placement, and skin tone distributions. Mild jaundice remains more difficult than obvious cases, and performance can degrade under very low light, glare, motion blur, or partially obstructed views. We also performed a small "realistic" check on 20 images, but this sample size is too small to generalise. It is mainly useful as a sanity check rather than a statistical evaluation.

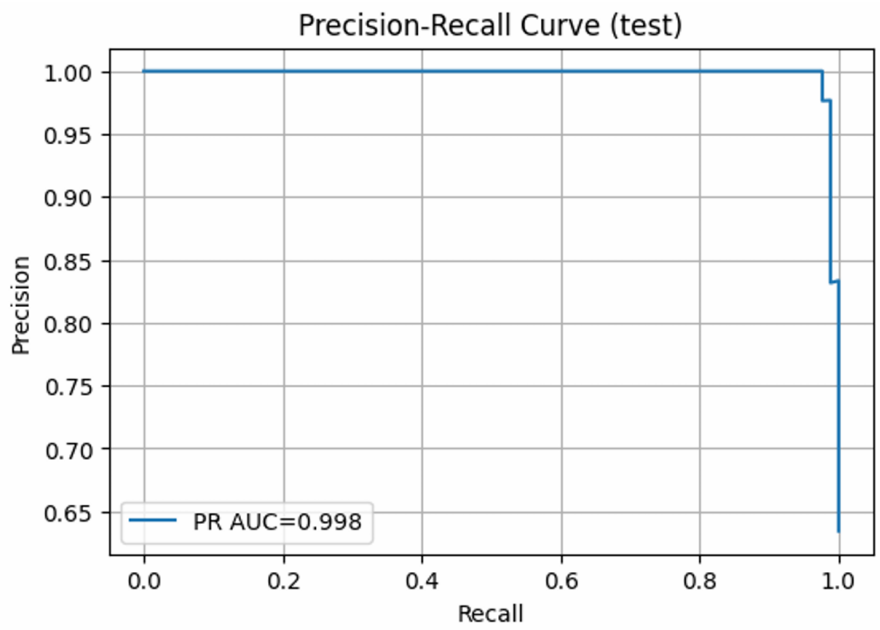
4.2.2 Cry Detection and Classification

The infant cry subsystem (Section 3.2.2) was trained and evaluated using the workflow in the `Cry-Detection-Classification-Model` repository [42]. The design follows a

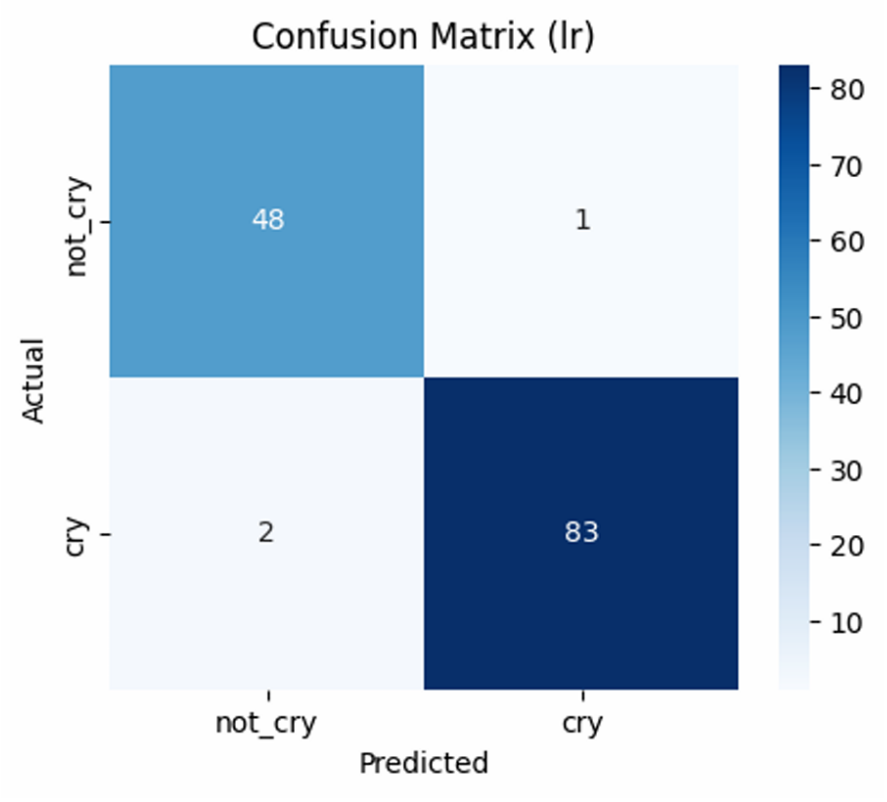
two-stage approach. First, a lightweight detector decides whether an audio segment contains a cry event. Only segments that pass this stage are forwarded to the second stage, which classifies the cry into one of five categories (*belly_pain*, *burping*, *discomfort*, *hungry*, *tired*). This separation is important in practice because the edge device continuously listens in the NICU environment, and it is not efficient (or necessary) to run a multi class classifier on every audio window.

4.2.2.1 Cry detection performance (cry vs. not_cry)

For the binary detection task, YAMNet [51] embeddings are used as a fixed feature extractor, followed by a Logistic Regression (LR) classifier trained on aggregated embeddings. On the test split reported in the repository documentation [42], the detector achieved a test accuracy of 0.978 with PR-AUC of 0.998. The corresponding confusion counts were TN = 48, FP = 1, FN = 2, and TP = 83 (test set size $n = 134$). In other words, false positives were rare in this evaluation, and most errors occurred near the boundaries of cry segments where energy is low and the acoustic pattern is less stable. Figure 4.3 visualises the detector behaviour using the PR curve and the LR confusion matrix as exported from the training notebook in the repository [42].



(a) PR curve (test)



(b) Confusion matrix

Figure 4.3: Cry detector evaluation.

4.2.2.2 Cry classification performance (five cry types)

For cry type classification, `librosa` [52] features are used with an ensemble classifier. On the reported test set ($n = 549$), the classifier achieved accuracy of 0.929, with macro averaged F1 of 0.80 and weighted F1 of 0.93 [42]. The confusion structure

in Figure 4.4 indicates that the majority class (*hungry*) is recognised strongly, while minority classes (e.g., *burping* and *belly_pain*) contribute most of the confusion. This is consistent with the smaller support sizes for those classes.

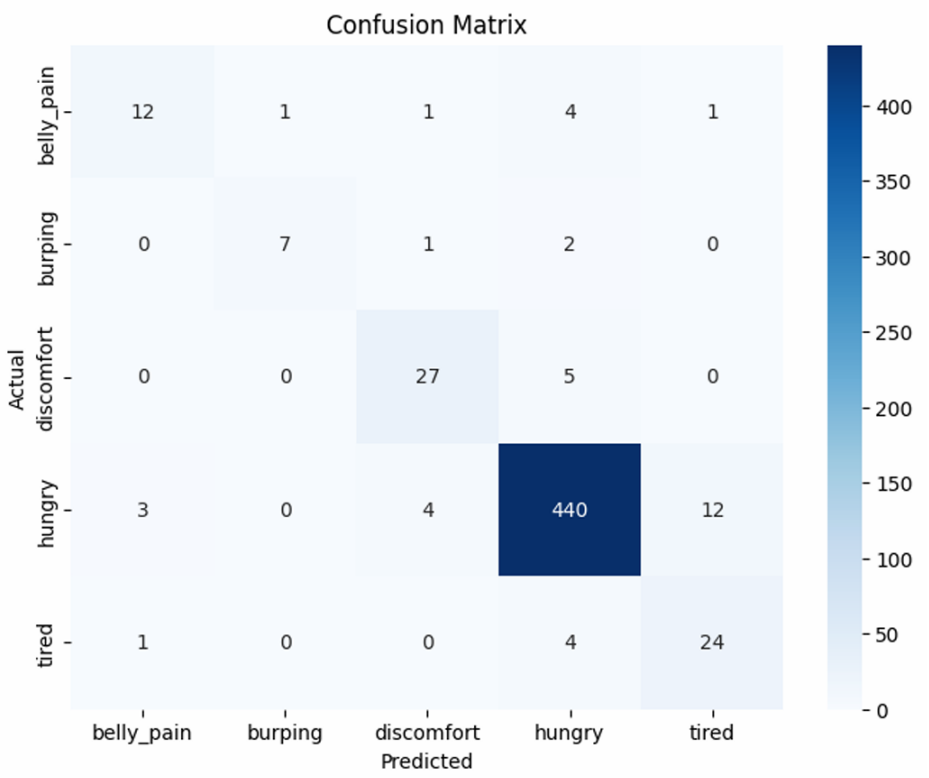


Figure 4.4: Cry type confusion matrix.

For clarity, Table 4.2 summarises the reported performance of the two stage cry pipeline on the held out test splits used in the `Cry-Detection-Classification-Model` repository [42]. The detection stage is evaluated as a binary classifier (*cry* vs. *not_cry*), while the classification stage is evaluated across five cry types.

Table 4.2: Cry subsystem performance summary on test splits.

Trained Model	Metric	Value
Cry detection	Accuracy	0.978
	Precision	0.99
	Recall	0.98
	PR AUC (validation)	1.000
	PR AUC (test)	0.998
Cry classification	Accuracy	0.929
	Precision	0.81
	Recall	0.79
	F1-score	0.80

On the Raspberry Pi, the real time service described in Section 3.2.2 adds an inexpensive pre filter before invoking the learned models. Audio is monitored continuously,

and a cry candidate is triggered when the energy concentration is consistent with a cry band (approximately 300–2000 Hz) and the amplitude exceeds a noise threshold. When a candidate is detected, the system records a 5 s segment and submits it to the two stage inference pipeline: (i) YAMNet+LR detection to confirm the event, followed by (ii) ensemble classification if the event is accepted as a cry. This design reduced unnecessary classification calls and kept the pipeline responsive during long runs, while still providing a usable "cry status" indicator for dashboards and alerts.

The classification task is constrained by class imbalance and by the limited diversity of available labelled recordings (background NICU sounds, microphone placement, and noise profiles can differ substantially). As a result, cry type predictions should be interpreted as supportive indicators rather than definitive causes. Similar to the jaundice module, what constitutes acceptable performance for clinical use depends on local policy and would require structured clinical validation and threshold-setting, which is not covered in this prototype stage.

4.2.3 Incubator LCD Display Reader

The LCD reader described in Section 3.2.3 was implemented in the `Neonatal_incubator_displayRea` repository [40]. The goal of this component is non invasive extraction of incubator vitals by reading the incubator's built in display, so that temperature, humidity, and other fields can be integrated into the monitoring pipeline without modifying the incubator hardware. The deployed pipeline combines a YOLOv8n detector [55] (to localise each numeric field on the screen) with EasyOCR [57] (to convert the cropped regions to digits), followed by rule based validation and smoothing logic on the edge device.

4.2.3.1 YOLO field detection performance

The YOLO model was trained on an onsite dataset collected from live incubator displays at the National Hospital Galle NICU and annotated (YOLO format) to detect the regions of interest corresponding to the fields used in this project (e.g., air temperature, humidity, heart rate, SpO_2, and skin temperature) [40]. The training run used an input size of 640 with early stopping (patience = 10) and standard Ultralytics training settings [55]. On the held out validation split, the detector achieved very high precision and recall, with mAP@0.5 close to 0.99 (Table 4.3). Practically, this means that the model is able to place bounding boxes around the correct display fields consistently, which is a prerequisite for reliable OCR and poor localisation would propagate directly into digit reading errors.

Table 4.3: YOLOv8n detection metrics (validation split).

Metric	Value
Precision	0.993
Recall	0.993
mAP@0.5	0.991
mAP@0.5:0.95	0.747

Figure 4.5 shows the training summary curves reported by Ultralytics (loss terms, precision/recall, and mAP trends). The precision recall curves per class, and the overall mAP@0.5 trace, are shown in Figure 4.6. These plots indicate that the detector quickly reaches a stable high performing regime after the initial epochs, and then improves gradually under stricter IoU scoring.

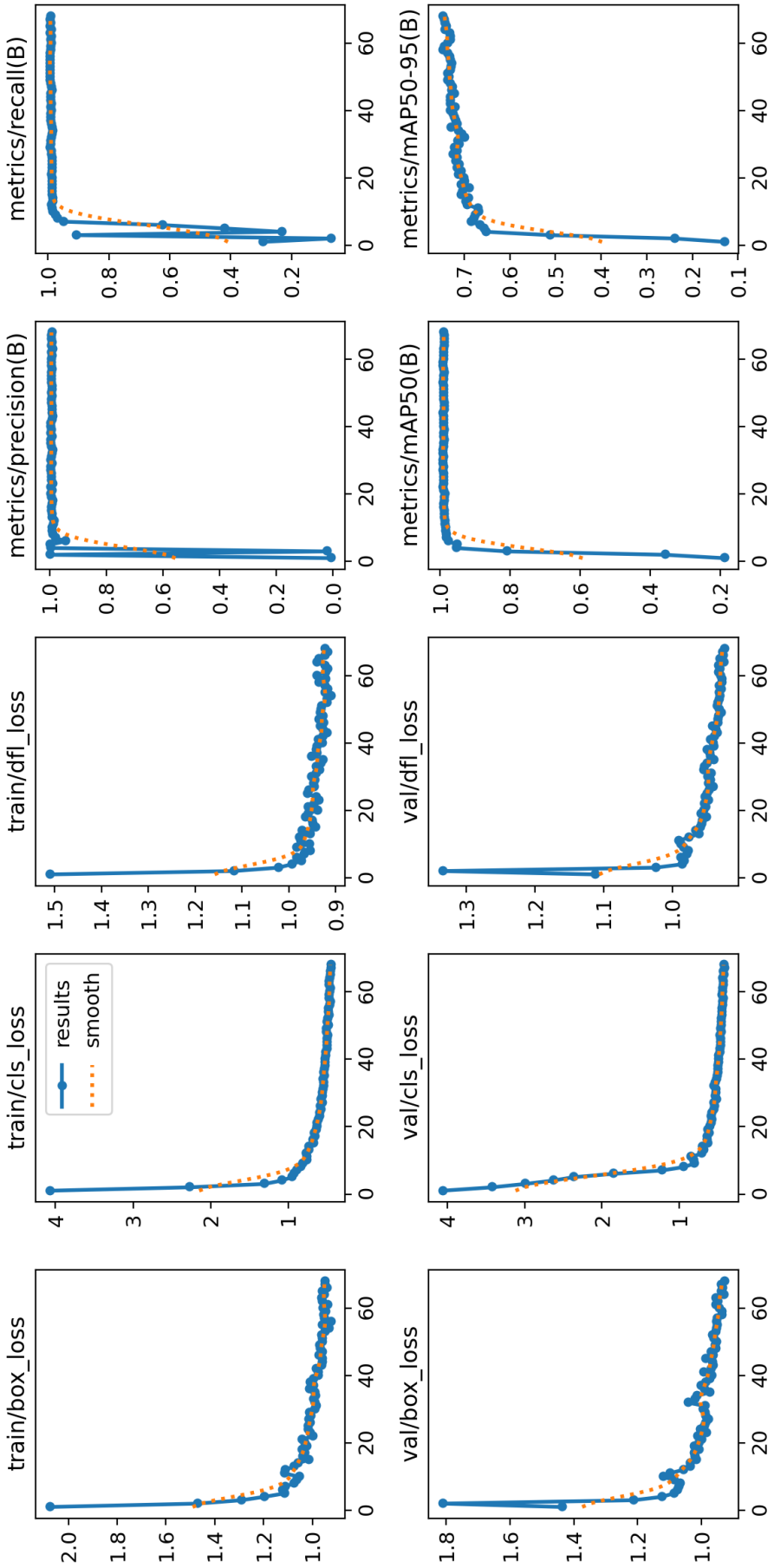


Figure 4.5: YOLO training summary curves.

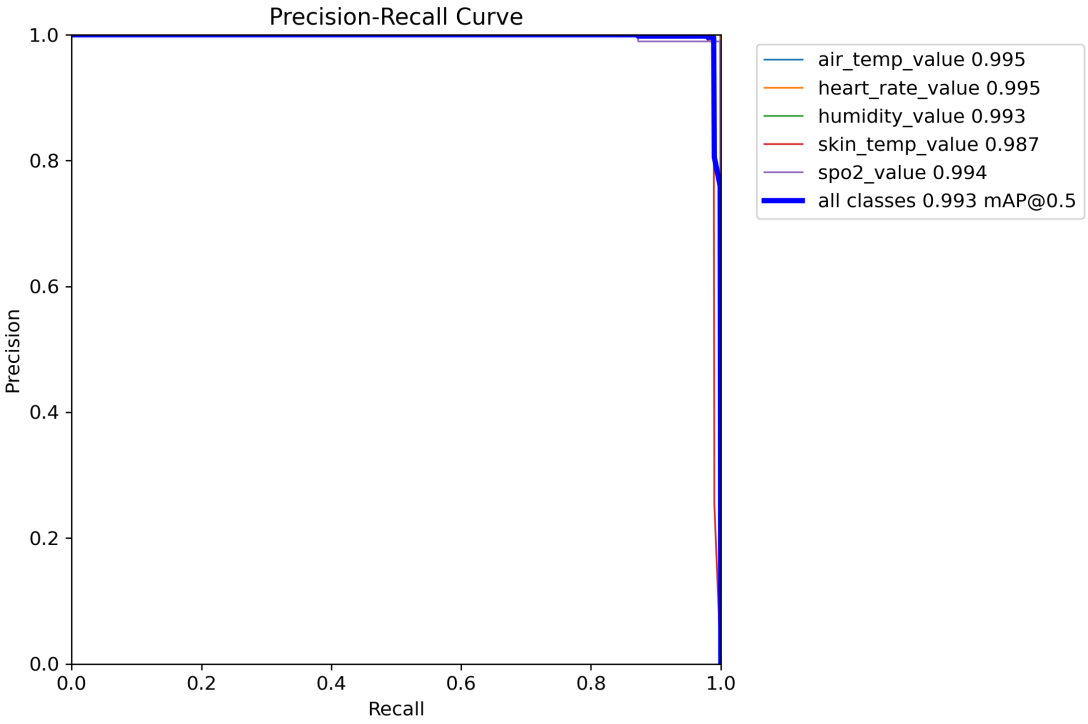


Figure 4.6: YOLO precision–recall curves.

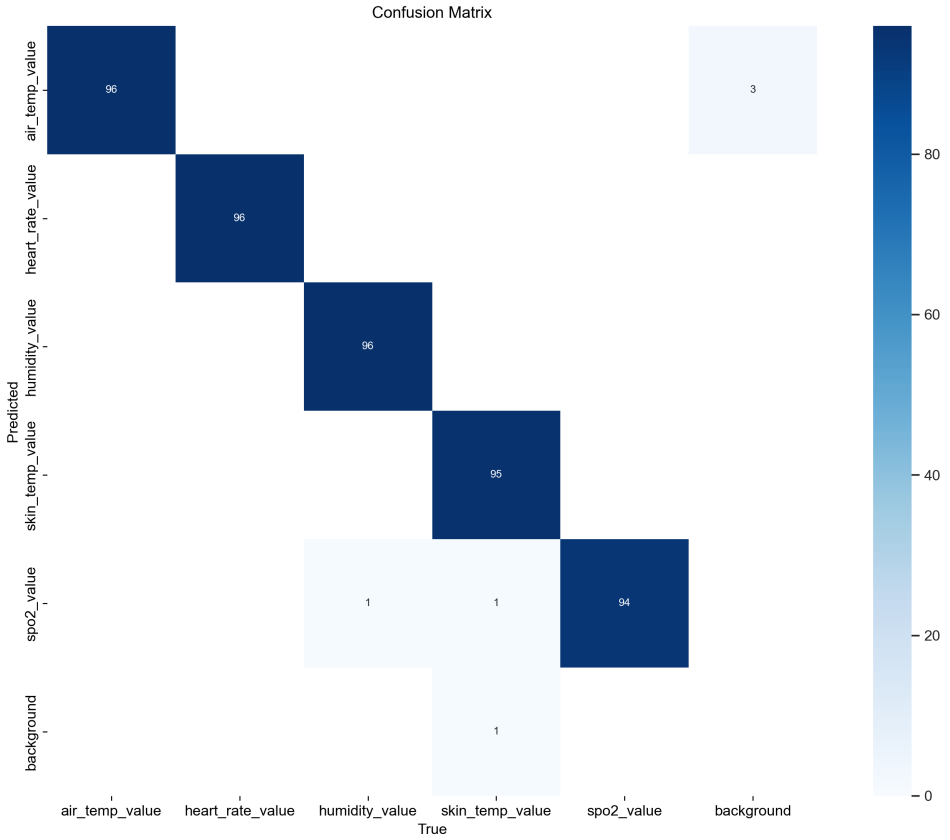


Figure 4.7: YOLO confusion matrix.

While the YOLO detector provides the display-field crops, OCR quality depends on digit sharpness, glare, camera exposure, and motion blur. For this reason, the project evaluated both Tesseract OCR [56] and EasyOCR [57] during development, and selected EasyOCR for the deployed pipeline because it produced more stable digit outputs on real incubator frames (especially under low contrast and mild glare), while Tesseract was more sensitive to small changes in preprocessing and often required heavier tuning.

In the running system, the output of OCR is passed through validation rules (expected numeric formats and plausible clinical ranges) and a "last-good" caching strategy implemented on the Raspberry Pi. This reduces the effect of occasional single frame failures: if one frame produces an implausible jump due to blur or partial occlusion, that reading is rejected and the previous stable value is retained until a consistent update is observed.

Figure 4.8 shows representative validation frames with predicted bounding boxes overlaid. In addition, Figure 4.7 shows the detector confusion matrix used in the training report; most errors are concentrated in the `background` class rather than between the vital field classes, which is consistent with the high per class precision/recall observed in the PR curves. Together, these figures support the practical observation that the localisation stage is not the limiting factor and errors that do occur in the full reader are more often caused by OCR failures under glare/blur rather than by incorrect field detection.

Figure 4.8: Example predictions on validation frames.

To illustrate the relationship between what the camera sees and what the model outputs, Figure 4.9 shows an example where the YOLO boxes and EasyOCR predictions are overlaid on a captured frame (the figure in the report is based on a representative example rather than a specific commercial display). In practice, the numeric outputs from the LCD reader are very close to those reported by the incubator itself, which was the minimum acceptable requirement for this component.

Figure 4.9: Illustrative LCD sample.

The LCD reader remains sensitive to physical setup. Camera placement, focus, and reflections can dominate OCR quality even when detection is correct. Although range checks and caching improve stability, the system still benefits from periodic human checking during installation and after any disturbance (camera bumped, display partially covered, or extreme lighting changes). In addition, the trained detector is based on the incubator displays available during data collection. Different incubator models or display layouts would require additional labelled data and re training to maintain the same performance level.

4.3 End to End System Validation

The component level results in Section 4.2 indicate that the jaundice, cry, and LCD modules work in isolation, but the system is only useful if the complete edge cloud path behaves reliably during real operation. Therefore, we carried out end to end validation where the Raspberry Pi edge node, ThingsBoard CE, the Cloud Run backend services, and the web/mobile dashboards were exercised together. The purpose was to confirm that (i) telemetry generated at the incubator reaches the cloud intact, (ii) the dashboards reflect the same state with acceptable delay, and (iii) the live video

and Pi side APIs remain reachable through the secured proxy route.

4.3.1 Edge Cloud Data Flow Tests

The first set of tests focused on connectivity and correctness of the data path. On the edge side, the Raspberry Pi 4B+ ran all relevant microservices from the edge device repository [39] (LCD reader, jaundice service, cry detector, NTE client, MQTT telemetry publisher, and health/snapshot services). Inputs were provided either by a physical incubator display or by the LCD display simulator [54] for repeatable runs. On the cloud side, ThingsBoard CE [50] and the Cloud Run services and dashboards [45] were configured as in normal operation.

To verify the forward path (Pi → ThingsBoard → dashboards), we enabled timestamped logging at the edge publisher and compared it against (i) ThingsBoard *latest telemetry* and (ii) REST API queries. Over multi-hour runs with a 5 s sampling interval, the telemetry values retrieved from ThingsBoard matched the values logged at the Pi, aside from the expected network and broker delay. Under normal network conditions, dashboard updates followed edge publication within a short delay (typically below a second for small payloads), which is sufficient for the parameters monitored in this project.

We also validated the reverse direction through the secured proxy route. The clinical dashboard and mobile app access edge resources through the Tailscale+Nginx gateway described in Section 3.1.1 [45, 46]. Using browser network logs and the Pi snapshot UI, we confirmed that requests were consistently mapped to the correct Pi services once the Nginx configuration was finalised.

Figure 4.10 summarises the main stages in the telemetry path. The diagram is intended to be indicative (showing where latency accumulates) rather than a strict benchmark, but it reflects the observed sequence, edge inference and packaging, MQTT publication, ThingsBoard ingestion, and UI refresh.

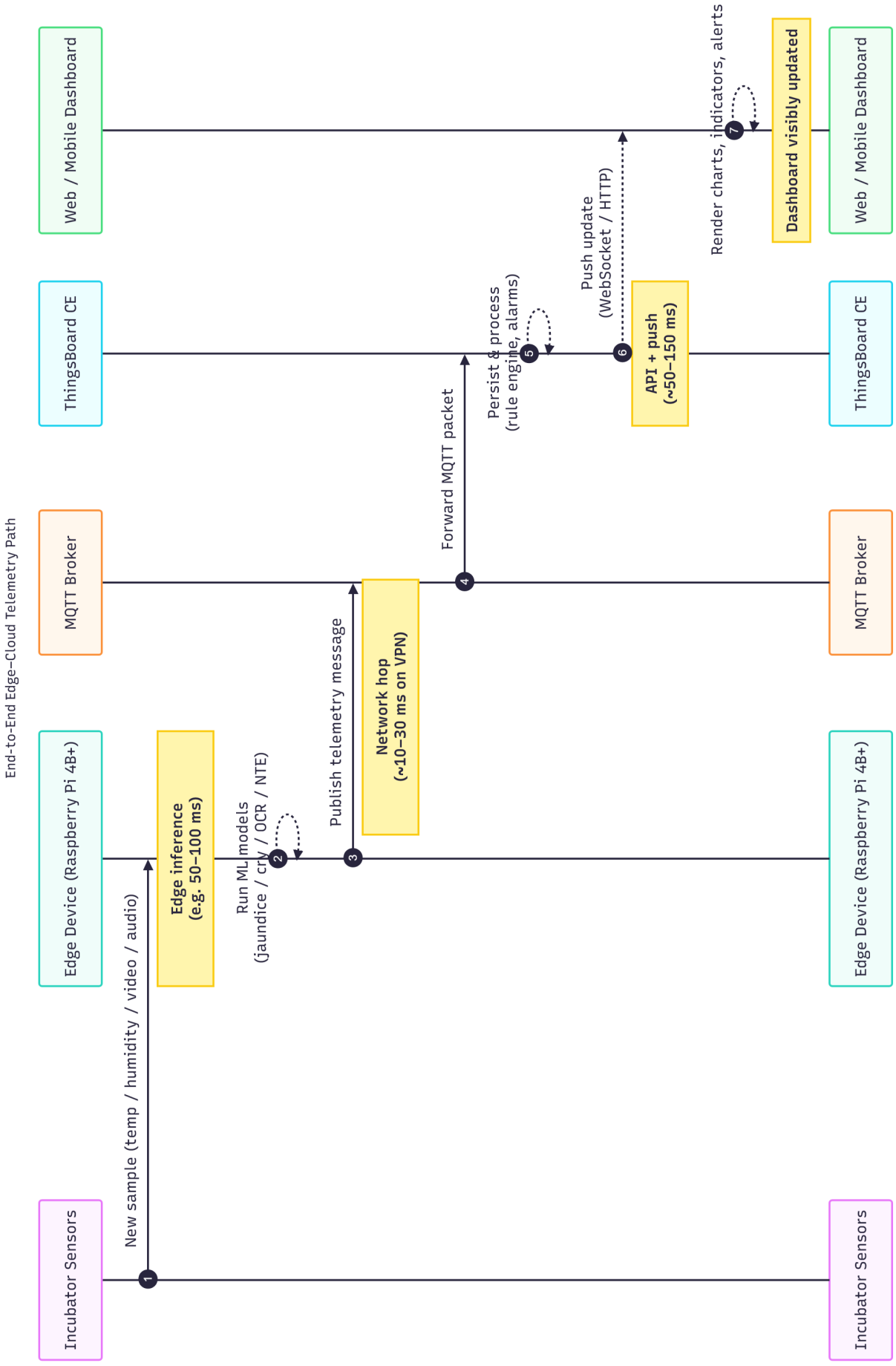


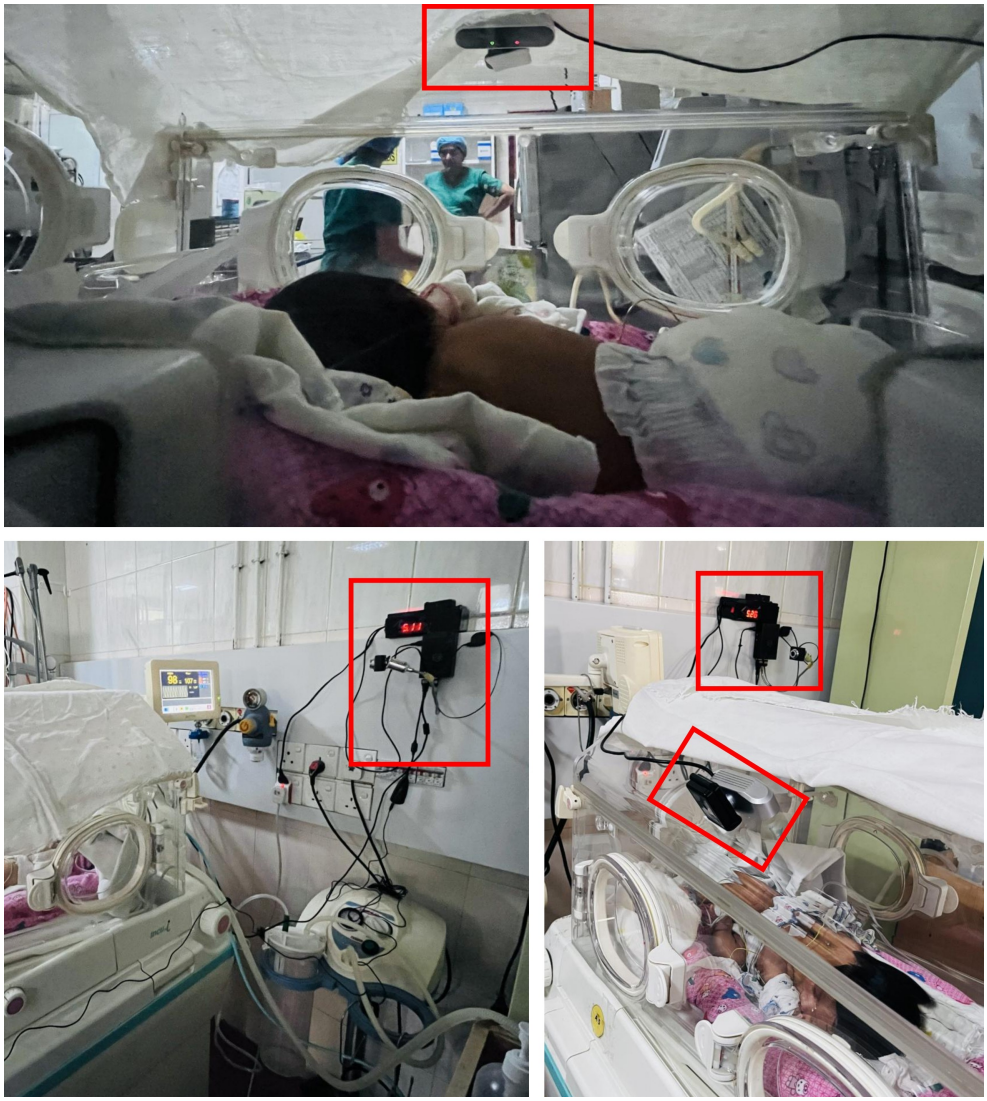
Figure 4.10: Indicative telemetry timing path.

4.3.2 NICU Test Run at National Hospital Galle

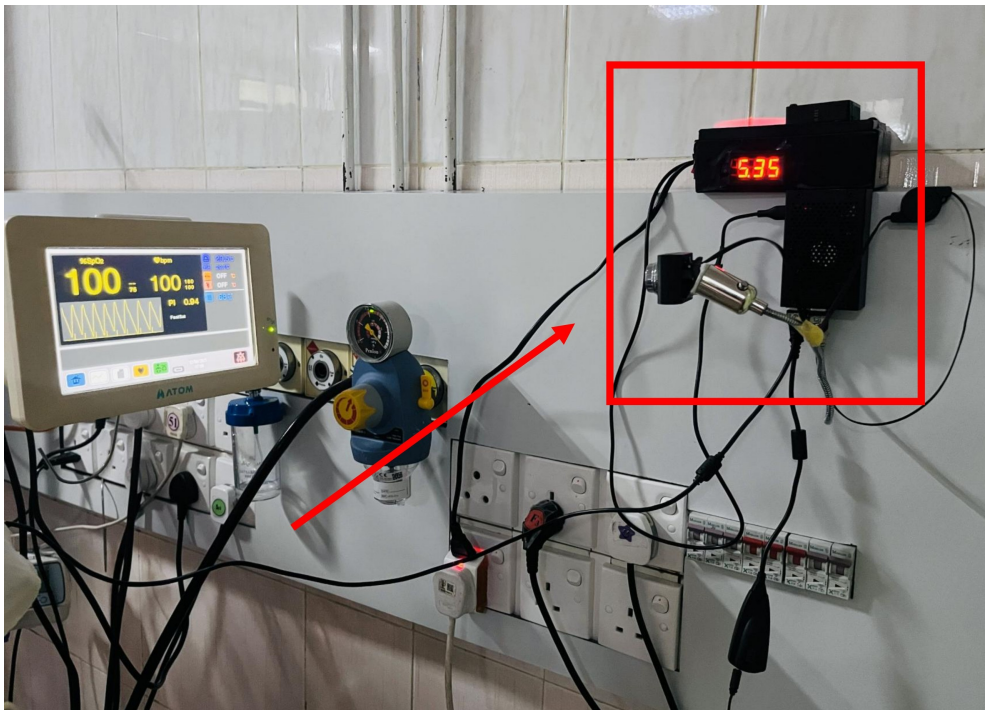
In addition to lab style and simulator assisted checks, we conducted a limited real environment test run at the Neonatal Intensive Care Unit (NICU) of the National Hospital Galle, under institutional permission and supervision. The purpose of this test was not to claim clinical validation, but to confirm that the complete NeoCare system can be deployed next to an incubator and remain operational while producing meaningful outputs in the actual NICU setting (lighting, background noise, and workflow constraints).

During the test run, the edge device (Raspberry Pi device with the dual camera setup) was positioned so that one camera captured the infant view and the second camera captured the incubator display. The Pi services executed continuously, publishing telemetry to ThingsBoard CE and updating the web dashboards through the deployed cloud stack [39, 45, 50]. A key outcome was that the pipeline successfully propagated a jaundice positive indication for a clinically jaundiced infant, while the quality gating helped avoid unreliable classifications when frames were unsuitable. Clinicians were able to view the same state through the clinical dashboard, and the parent view remained restricted to its intended features.

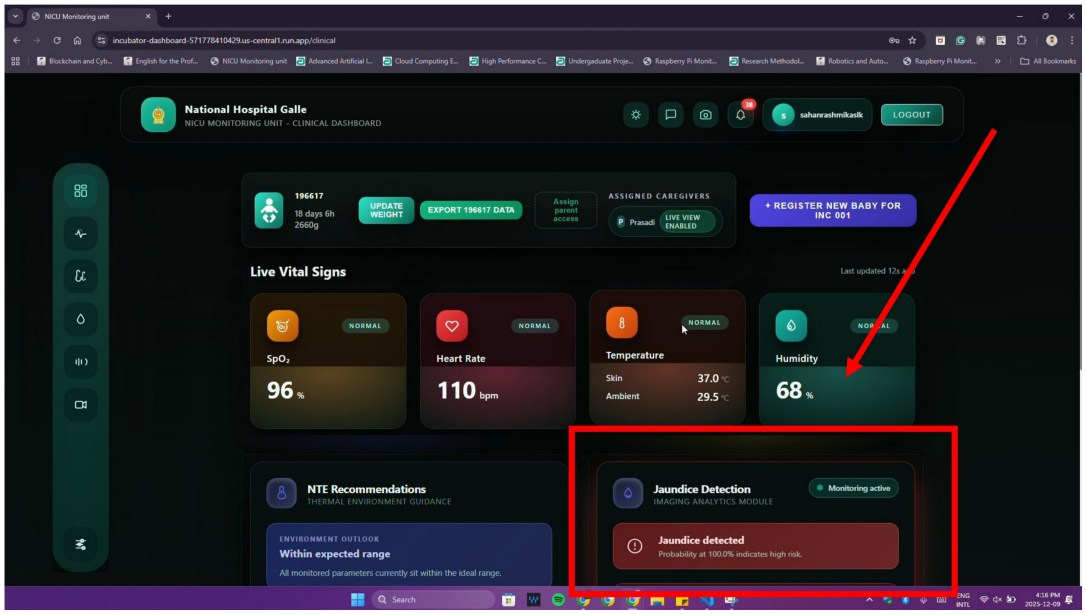
Figure 4.11 documents the NICU test run at a high level (example captures/photos). The figure is included as evidence of physical deployment and end to end operation in the intended environment.



(a) Infant in incubator



(b) Edge device setup



(c) Clinical dashboard

Figure 4.11: NICU test run at National Hospital Galle.

This NICU test run was limited in scope and sample size, and it should be interpreted as a feasibility demonstration rather than a clinical trial. Broader validation would require structured study design, larger cohorts, controlled ground truth (e.g., bilirubin measurements for jaundice), and formal ethics approval procedures aligned with hospital and national guidelines.

4.4 Usability and User Feedback

Beyond model accuracy and end to end correctness, the practical value of NeoCare depends on whether clinicians and parents can understand the interfaces quickly and trust what they see. To obtain an initial indication of usability, we conducted informal demonstrations and short discussions with a small number of doctors, nurses, and parents during our evaluation activities at the National Hospital Galle NICU and during controlled demonstrations. Participants interacted with the web dashboards [45] and the NICU mobile application [53] using typical workflows such as locating a baby, checking status indicators, viewing live video, and sending or reading messages. The feedback reported in this section is qualitative and limited in scale, therefore it should be read as indicative rather than as a formal user study.

4.4.1 Clinical Staff Evaluation (Doctors and Nurses)

Clinicians were first introduced to the clinical dashboard and asked to perform simple tasks, locating a specific infant record, checking whether the infant appeared stable, and identifying whether any indicators suggested that attention was needed. In most cases, users were able to interpret the colour coded status indicators quickly once they understood the basic layout. Several staff members commented that having the

incubator readings consolidated in one screen (instead of walking between incubators to read the displays) could reduce effort during busy periods, especially when monitoring multiple infants.

The NTE widget (based on Table 1.1 and the rule engine described in Section 3.3) was viewed as one of the more practical features. Nurses in particular noted that the explicit display of a target band alongside the current air temperature reduces reliance on memory and reduces the need to repeatedly consult printed guideline tables. The cry indicator was generally perceived as a useful supplementary signal. Staff noted that in a typical NICU it is not always possible to distinguish individual cries clearly due to ambient noise and competing alarms, a simple cry status marker provides an additional attention cue, provided the false trigger rate remains low. The jaundice indicator was received more cautiously. Clinicians emphasised that they would treat it as a reminder to recheck rather than as a diagnostic output, which is aligned with the intended role of the model.

When the clinician mode of the mobile app was shown, feedback was mixed by preference rather than by usability. Staff who already use mobile tools frequently indicated that they would be comfortable using it for quick checks while moving through the ward, whereas others preferred the workstation view. Across both groups, the general expectation was that mobile access is beneficial when it remains responsive and when connectivity is stable.

4.4.2 Parent Evaluation

Parents evaluated the parent dashboard and the parent mode of the mobile app. Consistently, the live video stream was the feature parents valued most. Parents reported that being able to see their baby remotely was reassuring, particularly because NICU visiting is often restricted and brief. At the same time, parents generally agreed with the design decision not to expose detailed vital sign numbers or alarm states directly in the parent interface, since these can be difficult to interpret without clinical context and could increase unnecessary anxiety.

The messaging feature was also well received. Parents indicated that short written updates and the ability to ask simple questions helped them feel more connected to what was happening in the unit. Importantly, most parents did not expect instant replies. They viewed messaging as a structured update channel rather than a real time chat.

4.4.3 QR Based Onboarding and Access Control

A specific usability focus was the QR based onboarding mechanism used to assign parents to the correct infant record. In the implemented workflow, staff generate an invitation that encodes the relevant access information (e.g., invitation code/PIN and the linked infant record) and provide it as a QR code (Figure 3.14h). Parents can then scan the QR code and complete registration with minimal manual entry. In

practice, this reduced the amount of manual typing required and reduced the likelihood of parents being linked to the wrong infant by accident.

Staff emphasised that QR based onboarding is only one part of access control, identity checks and consent remain clinical responsibilities. Nevertheless, the QR invitation mechanism was seen as a practical way to reduce administrative overhead while maintaining the intended separation of access between clinical users and parents.

4.5 Quantitative Analysis of Usage and Feedback

To complement the qualitative feedback above, we collected simple rating scale responses during evaluation sessions. This was not conducted as a formal study, but rather as a lightweight way to confirm whether the overall direction of the UI is acceptable and to identify any obvious usability problems early.

4.5.1 Likert Scale Ratings

After interacting with the web dashboards and the mobile app, participants were asked to rate a small set of statements on a 5 point Likert scale (1 = strongly disagree, 5 = strongly agree). Clinician items focused on clarity of the consolidated view, interpretability of NTE guidance, and usefulness of cry/jaundice indicators as supporting information. Parent items focused on ease of understanding, reassurance provided by the live video, and usefulness of messaging. Given the small number of respondents, the results should be interpreted as descriptive only.

Figure 4.12 provides a compact visual summary of the average ratings for selected items. The purpose of the figure is to communicate the direction of responses (generally positive) rather than to claim statistical significance.

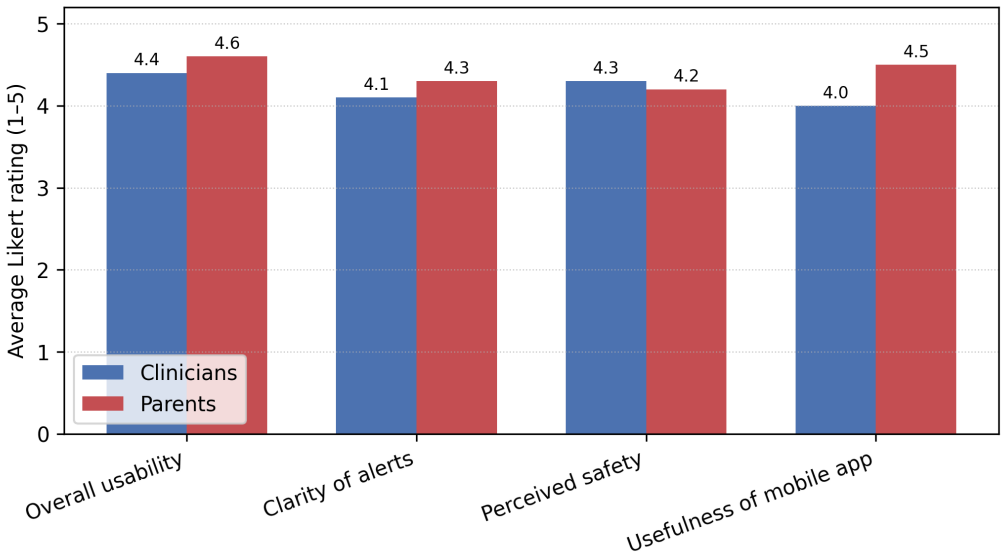


Figure 4.12: Indicative Likert scale averages.

4.5.2 Observed Interaction Behaviour

In addition to ratings, we observed how participants navigated the interfaces during demonstrations. Clinicians typically needed only a short explanation before they could locate an infant, interpret the status indicators, and open the live video stream. The main delays were caused by network/video buffering rather than by navigation complexity. Parents were generally able to complete the QR based onboarding and reach the live video view with limited guidance; the most common minor confusion was locating account/logout actions in the web view, which can be addressed with small UI refinements.

Overall, the combination of qualitative feedback and simple rating results suggests that the interfaces are usable for the intended roles, with the strongest perceived value being (i) consolidated clinician monitoring (including NTE guidance) and (ii) parent live video plus messaging. At the same time, the limitations noted by users especially the desire to avoid over reliance on alerts and the need for stable network/video performance reinforce the prototype nature of the current deployment and the need for broader evaluation in future work.

4.6 Discussion

The results in this chapter indicate that the prototype meets several of the project objectives, while also making clear what remains incomplete before any broader clinical deployment could be considered. In terms of the edge intelligence, the three key components (jaundice, cry, and LCD reading) achieved performance levels that are reasonable for an advisory system running on a Raspberry Pi. The jaundice model, built on MobileNetV3 Small [41, 48], achieved 0.930 accuracy on the held out evaluation split, with high specificity (0.976) and moderate sensitivity (0.813). The high specificity is practically important because it reduces false alerts and avoids continuous disruption to staff workflow, while the moderate sensitivity highlights that some true jaundice cases (especially mild or borderline cases) may still be missed. The cry pipeline [42, 51, 52] showed strong binary detection performance (accuracy 0.978 with PR-AUC 0.998) and produced usable but imperfect cry type classification (accuracy 0.929 with macro F1 0.80), reflecting the expected difficulty of minority cry types and the limits of available labelled audio. The LCD reader [40, 55, 57] achieved very strong detection performance (validation precision/recall around 0.993 and mAP@0.5 around 0.99), supporting the feasibility of non invasive capture of incubator vitals through a camera based approach when the display layout is known and the camera setup is stable.

End to end validation further suggests that the overall architecture works as designed. The Raspberry Pi microservices successfully publish telemetry to ThingsBoard CE over MQTT [50], and the cloud services and dashboards retrieve and present the same state through the deployed back-end stack [45, 53]. The secured reverse proxy path (Tailscale + Nginx) remained stable in the tested configuration, allowing dashboards and mobile clients to access edge APIs and live video without exposing the Pi directly to the public internet [45, 46]. Importantly, the system was also demonstrated

in a limited real environment test run at the National Hospital Galle NICU under permission, which supports the practical feasibility of deploying the edge device near an incubator and operating the complete pipeline in the intended setting.

At the same time, the evaluation underlines limitations that must be stated explicitly. First, the machine learning components are not trained on large, locally representative clinical datasets. The jaundice training data were taken from a public Kaggle dataset [49], and the cry datasets remain limited and imbalanced for certain classes. Consequently, the models should be treated strictly as decision-support signals and not as diagnostic outputs. This view aligns with clinician feedback: staff were interested in the additional indicators, but they emphasised that clinical judgement and conventional assessment (including laboratory confirmation for jaundice where relevant) remain primary. The current implementation reflects that stance by presenting outputs as flags, confidence/quality indicators, and recommendations rather than as automatic decisions.

Second, the LCD reader depends strongly on physical installation quality. Even with strong detection metrics, OCR can degrade due to glare, dust, camera focus drift, or partial occlusion, and while range checks and caching reduce the impact of single-frame failures, periodic human verification remains necessary. Third, although the cry detector performed well on the repository test set, the acoustic variability of a busy NICU (multiple infants, staff speech, alarms, and equipment noise) is broader than typical curated datasets and extended real environment validation is still required before relying on cry type classification in routine operation.

From a usability perspective, feedback from clinicians and parents was broadly encouraging, clinicians valued the consolidated view and NTE guidance, while parents valued the live video and messaging features. However, the evaluation sample was small and the sessions were informal, so the results should not be interpreted as a formal usability study. Practical engineering aspects such as robustness to network dropouts, clear error reporting when the edge device goes offline, and video buffering under constrained connectivity also remain important for future work, particularly when scaling beyond a single incubator installation.

Overall, the prototype demonstrates that a low cost edge device, a cloud IoT platform, and role based dashboards can be integrated into a coherent neonatal incubator monitoring system. The measured performance of the main ML components is acceptable for advisory use, and the end to end deployment operated as intended in both controlled tests and a limited NICU test run. Nevertheless, the results also make it clear that broader validation, richer local datasets, and more structured evaluation are required before the system could be considered for routine clinical use.

Chapter 5

Timeline

5.1 Timeline

The work on the system was organised into four main phases, largely following the structure outlined in the original project proposal. Figure 5.1 shows the overall time plan as a Gantt style chart. In this section we briefly describe what was planned for each phase and what was actually achieved by the end of the project period.

Phase 1: In depth research and system design (Months 1–2) – Completed

The first phase concentrated on background study and high level design. During the initial month the team carried out a literature review on neonatal care requirements, incubator design, sensor technologies and related AI applications in healthcare. These findings, summarised in Chapters 1 and 2, were used to identify the main functional requirements which physiological and environmental variables should be monitored, what accuracy and update rates would be acceptable (for example, aiming for roughly ± 0.5 °C stability in air temperature and a 1 Hz logging rate for environmental data), and what capabilities the user interfaces should provide (multi user logins, live streaming, mobile access).

By the end of Month 2, the team had produced a first version of the system architecture, defining the split between the Raspberry Pi edge device, the ThingsBoard CE instance, and the cloud backend services (later refined into the architecture described in Chapter 3). The initial hardware set was also procured at this stage, including a Raspberry Pi 3B+, two webcams, a USB microphone and basic networking equipment. Conceptual UI mockups were created in Figma, which later guided the React implementation.

Phase 2: Focused development and implementation (Months 3–5) – Completed

The second phase was devoted to implementing the individual components. At the start of Month 3, the Raspberry Pi was configured with a clean OS image and base software stack, and code was written to read from the temperature/humidity sensor, capture images from the cameras and log data to disk. Calibration routines were

implemented for the environmental sensors to ensure that the recorded values were reasonably accurate.

Machine learning development proceeded in parallel. An initial cry detection model based on MFCC features and a decision tree was implemented and tested, followed by experiments with more advanced classifiers and the adoption of the YAMNet+Logistic Regression and ensemble pipeline [42, 51, 52]. For jaundice detection, the team initially tried a simple colour based classifier but quickly moved to a CNN, by the end of Month 4, a MobileNetV3 Small model [41, 48] had been trained and successfully exported to ONNX for on-device inference. The NTE recommendation engine was implemented based on the guideline table (Table 1.1), and an early version of the LCD reader, using YOLO and Tesseract, was integrated.

On the UI side, a basic local web interface running directly on the Pi was built to display sensor values and a simple live video. By the end of Phase 2, the system supported remote access to this local dashboard over the LAN, live camera streaming from the Pi, and preliminary alerting when environmental thresholds were exceeded. Although some pieces were still rough around the edges at this point, most of the core modules described in Chapter 3 existed in an initial form.

It was also during this phase that the limitations of the Raspberry Pi 3B+ became apparent. As more services were added, it became clear that the 3B+ could not reliably handle all camera and ML workloads simultaneously. In response, the hardware platform was upgraded to a Raspberry Pi 4B+ with 4 GB RAM, and the microservices were reorganised so that each function ran in its own process. This change laid the groundwork for the stable operation described later.

Phase 3: System integration, testing, and validation (Months 6–8) – Completed

Phase three focused on the integration of various subsystems into a seamless operational stack and the validation of end to end functionality when in continuous operation. At the start of Month 6, the edge services encompassing LCD reader, jaundice inference, cry pipeline, and NTE recommendation client were integrated into a single deploy on the Raspberry Pi 4B+ and linked with the cloud layer using MQTT to a ThingsBoard CE running on a Google Cloud Platform virtual machine. Concomitant with these operations, the Cloud Run services associated with the React dashboards and Node.js servers went live [45], and a secure entry point to the Raspberry Pi services (Tailscale router VM and Nginx reverse proxy) was setup to allow distant operation of the dashboards with video and edge API operation without making the Raspberry Pi directly accessible through the internet [45, 46].

Also, by the midpoint of this phase, an integrated prototype (Prototype v1) reaches operational functionality. The edge node has the capability of detecting incubator vital signs displayed on the LCD, the implementation of the jaundice and cry models, the computation of the NTE guidance, the publishing of the telemetry data to ThingsBoard, and the display of the same information on the clinical dashboard and the mobile app. Towards the end of the phase, functional testing and stability analysis

were considered. Functional testing entailed the validation of the consistent publishing of telemetry fields, the updating of the dashboard, and the stability of the proxy routes of the endpoints (and live video). Long running continuous testing (about 48 hours) were considered in the search for the existence of memory leaks, camera dropout patterns, thermal throttling, as well as the stability of the application under intermittent connectivity.

Importantly, this phase also included a limited real environment test run at the NICU of the National Hospital Galle, conducted under permission and supervision. During this deployment style run, the full edge cloud pipeline was operated next to an incubator, and the dashboards were used to observe the live system state. This activity served as a feasibility check that the integrated stack can operate in the intended setting (lighting, noise, and workflow constraints), and it provided practical feedback on camera placement, network stability, and how clinicians interpret the displayed indicators.

Towards the end of Phase 3, informal usability sessions were carried out with doctors, nurses, and parents to gather qualitative feedback on the dashboards and the mobile app (Section 4.4). This feedback informed small refinements to UI layout and wording, and it helped prioritise which features should be emphasised in the final system demonstration.

Phase 4: Documentation and project reporting (Months 9–10) – Completed

The final stage was concerned with documentation, analysis, and reporting. The team was tasked with documenting their designs and tests in months 9 and 10. The report included hardware schematics, software architecture diagrams, descriptions for the training and deployment of models, and integration and stress test results. A user manual for the prototype was developed. The operations and shutdown processes for services, explanation of dashboard views and NTE, and procedures for responding to alerts were included in the user manuals for nurses and technicians.

At the same time, the results for the quantitative and qualitative aspects of Phase 3 were compiled for Chapter 4 of this thesis, whereas the timeline, difficulties, and future perspectives were noted for the remaining chapters in 5 and 6. Demo resources in the form of videos were created for the dashboard application as well as the mobile app for the defense of this project. While some final touches were still required towards the end of the timeline of the entire project, the main process of development within the planned ten months was completed. Phases of the projects had some overlapping elements in terms of model development or testing, but there were no setbacks.

Milestones	Feb				March				April				May				June				July				Aug				Sep				Oct				Nov			
	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
Grouping and Selection of the Project Topic																																								
Studying the Background & Literature Review																																								
Project Proposal Submission																																								
Project Proposal Presentation and VIVA																																								
Phase 1: Research & System Design																																								
Phase 2: Development & Implementation																																								
Project Progress Report Submission																																								
Project Progress Presentation and VIVA																																								
Phase 3: System Integration & Testing																																								
Phase 4: Documentation & Reporting																																								
Submission of Final Report																																								
Final Presentation and VIVA																																								

Figure 5.1: Project time plan.

Chapter 6

Challenges and Limitations

Even though the prototype achieved most of its technical goals, a number of challenges had to be addressed along the way, and several limitations remain. This chapter summarises the most important of these issues. The emphasis is on practical constraints we encountered during development and testing, rather than on hypothetical problems.

6.1 Edge Device Constraints and Performance

The edge device proved to be one of the main bottlenecks in the project. Early work was carried out on a Raspberry Pi 3B+. While this platform was adequate for basic sensor reading and simple web serving, it struggled once attempted to run two camera streams, multiple machine learning services and HTTP APIs simultaneously. CPU utilisation was frequently close to 100 %, and under load the camera streams stuttered and inference latencies became unpredictable. This experience ultimately led to migrate to a Raspberry Pi 4B+ with 4 GB RAM, as described in Chapter 3. The Pi 4B+ handled the workload much better, but the migration also revealed additional hardware level issues.

One such problem was related to power delivery. When both cameras were active at once with the Pi and other peripherals, the 5 V supply at the Pi would occasionally sag below a safe level. This typically manifested as the cameras dropping out or the Pi reporting under voltage warnings, especially when the system had been running for some time. After some investigation, traced that the issue with the combination of the original power supply and the additional current draw from the USB devices. The eventual solution was to move to a 12 V power brick feeding a buck converter, and to set the buck output slightly above 5 V (around 5.2 to 5.3 V) at the Pi input. With this arrangement the cameras were able to run continuously without power drops. Although the fix was straightforward, it underlines how sensitive the setup is to power quality, particularly when multiple peripherals are attached.

A second issue concerned heat build-up inside the enclosure that housed the Pi, the buck converter and the local power supply. When run for an extended period, especially with both cameras and all microservices active, the internal temperature of the case increased, and the Pi CPU temperature approached the throttling threshold. This led to reduced performance and, in some cases, instability. To address

this, added a small DC fan and improved the airflow through the case by adding vents. This reduced the Pi's operating temperature by several degrees and prevented throttling under normal conditions. Nevertheless, the experience reinforced the point that running a full stack of services on a Pi inside a closed box is not trivial, careful attention needs to be paid to both power and thermal design, not just software.

Even with the upgraded hardware, the system remains resource constrained. CPU time and memory must be budgeted, and it is unlikely that much more intensive models could be added without either optimising or offloading some of the existing workload. The current design therefore leaves some headroom and is conservative in how often certain tasks (such as LCD OCR or cry classification) are performed.

6.2 Machine Learning Data and Model Generalisation

The machine learning models used in this project were trained under relatively tight data constraints. The jaundice detector relies primarily on the Kaggle Jaundice Image Data set, which, while useful, does not fully capture the diversity of skin tones, lighting conditions, camera angles, and incubator environments encountered in practice. Similarly, the cry detection and classification models were trained on a set of infant cry and non-cry recordings that is limited in size and scope [42]. As a result, the models' ability to generalise to different hospitals, cameras and populations is not guaranteed.

In addition, the conditions under which the models were trained are not identical to those at deployment. For example, many of the jaundice images in the training set are "clean" photographs, whereas in the incubator the baby may be partially covered, lit from unusual angles or viewed through plastic. The cry recordings used for training were not all captured in a NICU environment, so background noise patterns such as multiple infants crying at once or overlapping alarms are under represented. Although augmentation and mixing were used to mitigate this to some extent, the risk of domain shift remains.

Given these limitations, the models should be considered as decision support tools rather than as definitive diagnostic instruments. This view was shared by clinicians who tested the system, who were comfortable treating the outputs as additional signals but not as replacements for their own judgement or laboratory tests. A proper clinical validation study with larger, carefully curated datasets would be required before stronger claims about generalisation could be made.

6.3 OCR Robustness and Physical Setup Dependencies

The incubator LCD reader performs well in controlled tests, but its reliability depends heavily on the physical setup. The YOLOv8n detector and EasyOCR engine

can only work with what the camera sees. If the camera is mounted at a poor angle, too far away, or if the display window is affected by glare, dust or reflections, OCR performance will be degraded. Tests in the NICU environment confirmed that reflections on the incubator's plastic cover and small changes in camera position can make the digits harder to read. Some of these issues were addressed by adjusting the camera mount, using anti glare film, and tuning the cropping and preprocessing, but they cannot be fully eliminated.

The system compensates for occasional misreads by checking values against plausible physiological ranges and by keeping a cache of the last valid reading. This approach works well for filtering obvious outliers, but it only reduces the effect of isolated errors. It cannot help if the camera is misaligned for an extended period or if the display is obscured. In other words, the LCD reader remains dependent on correct installation and periodic visual inspection. For large scale deployment, a more robust mounting solution and possibly a calibration or alignment procedure would likely be necessary.

6.4 Network Architecture and Operational Complexity

The network architecture adopted in this project is intentionally conservative from a security standpoint but introduces some operational complexity. Using a Tailscale mesh network [46] and a dedicated router VM with Nginx means that the Raspberry Pi devices do not need to expose any services directly to the internet. This is desirable in a hospital context, where direct inbound connections to bedside devices are often not allowed. However, it also means that correct operation depends on several layers being configured and maintained, the Tailscale control plane, the router VM, the reverse proxy configuration, and the connectivity between Cloud Run and the VPC.

In experience, once these layers were correctly set up, the system was stable. Nevertheless, misconfiguration at any point (for example, a Tailscale route not being approved, or an Nginx location block being mis typed) could break access to edge services in ways that are not immediately obvious to end users. Debugging such problems usually requires familiarity with both cloud networking and VPN configuration, which may not always be available in a clinical IT team. Moving towards a more automated or "infrastructure as code" approach could help, but that was beyond the scope of the current project.

Another consideration is network reliability and bandwidth. While MQTT telemetry is lightweight, live MJPEG video streams are relatively heavy. In some test runs over slower networks, it observed visible buffering in the video component of the dashboard and mobile app. In a real NICU, where WiFi conditions and backhaul may vary, this could become a usability issue. Alternatives such as more efficient video codecs or adaptive streaming were not explored in this prototype and remain potential improvements.

6.5 Security, Privacy, and Regulatory Considerations

Security and privacy were kept in mind during design, all external communication with the cloud back end uses HTTPS, the Raspberry Pi is shielded behind the Tailscale VPN, and access to the dashboards and mobile app is controlled by JWT based authentication [45]. Parents only see their own baby's video and high level status, and do not see raw vitals or alarms. Nevertheless, the current system has not undergone a formal security audit, and a full privacy impact assessment has not been conducted.

From a regulatory perspective, the prototype is not a certified medical device. It reads existing incubator displays, makes recommendations based on published guidelines [47], and highlights potential issues, but it does not directly control any life support hardware. This reduces the regulatory burden somewhat, but not entirely. Any deployment in a real NICU would still need to consider local regulations, hospital policies, and standards such as IEC 60601-2-19 [65]. A more thorough treatment of security and regulatory compliance would be necessary before moving beyond research and demonstration.

6.6 Usability Evaluation Limitations

The usability and user feedback results reported in Chapter 4 are based on a relatively small number of sessions with clinicians and parents, and on informal observation and short questionnaires rather than on a large, controlled user study. Participants were generally positive about the system, but the sample is too small to draw firm conclusions about usability across different hospitals or user populations. In addition, all demonstrations were conducted with the research team present, which may have influenced participant behaviour and responses.

Another limitation is that the prototype has not yet been evaluated under real workload conditions in a busy NICU. The test scenarios were realistic in terms of data and workflows, but they took place in controlled settings rather than in the middle of a full shift. Factors such as alarm fatigue, competing priorities, and interaction with other hospital systems (e.g. electronic medical records) could affect how the system is actually used in practice. Addressing these questions would require a carefully designed pilot study in collaboration with clinical partners, which was outside the scope of this final-year project but would be a logical next step.

Chapter 7

Future Work

The work presented in this report demonstrates that it is technically feasible to build an edge centric, cloud connected monitoring system around existing infant incubators. At the same time, the prototype is clearly only a first step. This chapter outlines several directions in which the system could be developed further, grouped under model and data improvements, scaling and deployment, clinical integration, control and safety, and user centred evaluation.

7.1 Model and Data Improvements

A natural next step is to improve the robustness and generality of the machine learning components. For the jaundice detector, this primarily means acquiring more clinically representative data. The current model is trained mainly on the Kaggle jaundice image dataset [49] and a small number of additional samples, which does not fully capture the range of skin tones, lighting conditions and camera viewpoints encountered in practice. Collaborations with hospital partners could, subject to appropriate ethical approvals and anonymisation, provide a larger set of images with associated bilirubin measurements. Such data would allow more careful calibration of the model and potentially enable progression from simple binary labels ("jaundiced / non jaundiced") to more graded risk estimates.

For the cry subsystem, additional recordings from real NICU environments would help address the gap between the somewhat idealised training conditions and the complex acoustic reality of intensive care units. With a larger corpus of labelled cries and background sounds, the YAMNet based detection and ensemble classifier could be retrained and possibly extended with sequence models that capture longer temporal patterns (for example, recurrent or transformer architectures). As with the jaundice model, the intent would be to retain a computationally efficient pipeline on the Pi while improving discrimination between different cry types and reducing false positives from non cry noises.

The LCD reader could also benefit from additional data. At present, the YOLOv8n detector and EasyOCR are trained or tuned on a limited set of incubator display images and simulator outputs. Collecting images from a wider range of incubator models and display technologies, and augmenting them under more varied lighting and camera angles, would allow retraining or finetuning of the detector and more

thorough validation of OCR performance. In parallel, model compression techniques such as pruning and quantization could be explored to reduce the computational load of the ML components without sacrificing accuracy, which would make the system more scalable on resource limited hardware.

7.2 Scaling and Multi Incubator Deployment

The current prototype has been exercised primarily with one incubator and one Raspberry Pi edge node at a time. In a real unit, many incubators may be present, and it is not obvious that simply duplicating the current configuration will be manageable without further tooling. Future work should therefore address multi incubator scaling explicitly.

At the edge level, one straightforward approach is to assign one Raspberry Pi per incubator, each configured as described in Chapters 3 and 6. To make this practical at scale, tools are needed to automate registration and provisioning, new Pis should be able to join the Tailscale network, register themselves with ThingsBoard CE (or be registered by a central service), and receive their device tokens and configuration files without manual editing on every unit. On the cloud side, the React dashboard and backends already support many devices logically, but additional views could be added to group incubators by ward, room or nurse assignment.

As the number of monitored incubators increases, the load on the ThingsBoard instance and the Cloud SQL database will also grow. It may become necessary to allocate more resources to the VM hosting ThingsBoard CE, to adjust retention policies for telemetry, and to consider sharding or separate instances for different units. Similarly, the router VM carrying the Tailscale and Nginx proxy may need scaling or redundancy to avoid becoming a single point of failure. These concerns suggest that a more formal infrastructure-as-code setup (using tools such as Terraform or Ansible) would be beneficial in future, even though the current project relied mainly on manually maintained scripts.

7.3 Clinical Integration and Interoperability

At present, the system operates as a standalone monitoring platform. In many hospitals, however, it would be advantageous to integrate with existing clinical information systems, such as electronic medical records (EMRs) or central monitoring stations. Standards such as HL7 and FHIR could be used to encode vitals and alerts in a form that other systems can consume. For instance, the backends could expose FHIR Observation resources corresponding to heart rate, SpO_2, temperature and NTE recommendations, and these could be ingested by an EMR for longitudinal record keeping, or used to generate entries in clinical dashboards that the staff already use.

Interoperability also extends to identity and access management. While the current system manages its own user accounts in Cloud SQL, a more integrated deploy-

ment would likely need to authenticate clinicians against the hospital's directory (for example via LDAP or OAuth with the institution's identity provider) and enforce role-based access based on existing groups. Parent identity verification and consent workflows would need to be agreed with the unit and documented. Addressing these issues would move the system closer to actual clinical deployment but was outside the scope of this final year project.

7.4 Towards Closed Loop Control and Safety

The prototype is intentionally advisory and it reads incubator displays, computes NTE recommendations, detects cries and jaundice, and presents this information, but it does not directly actuate any hardware. In the longer term, one may consider extending the system towards closed loop control of incubator settings, at least in limited ways. For example, the NTE engine could drive small adjustments to incubator air temperature within configured bounds, subject to clinician oversight.

Any move in this direction would raise the safety and regulatory stakes considerably. It would require robust hardware interfaces to the incubator (whether via manufacturer APIs or via standardised control interfaces), rigorous hazard analysis, and a formalised safety case that demonstrates compliance with standards such as IEC 60601-2-19. Redundant sensors and plausibility checks would be necessary to detect sensor failures or inconsistencies. While these topics are beyond the scope of the current project, the modular architecture implemented here (where the NTE engine is already a separate service) should make it easier to explore closed loop strategies in a controlled research environment, for instance using a test incubator rather than a clinical device.

7.5 Usability and Clinical Evaluation

The usability results in Chapter 4 are encouraging, but they are based on a small number of participants and informal observations. As a next step, more structured user studies would be valuable. For clinicians, this could take the form of controlled trials in which tasks (such as identifying unstable infants or adjusting incubator settings in response to NTE changes) are performed with and without the system, and measures such as time to decision, error rates and subjective workload are compared. For parents, one could design a study that measures anxiety levels and perceived involvement in care with and without the video and messaging features.

In parallel, the mobile application could be extended with more robust notification handling and user preferences (for example, allowing clinicians to specify which types of alerts should trigger mobile notifications and during which hours). Longer-term pilot deployments, even if limited to a small number of beds, would provide valuable insights into how the system fits into real workflows, where it is most helpful and where it risks adding noise or distraction.

7.6 Long Term Vision

Looking further ahead, one can imagine the system evolving into a more comprehensive neonatal monitoring platform. The architecture could be adapted for different settings, such as step down units or post discharge home monitoring for high risk infants. The same combination of a low cost edge node, a cloud IoT back end, and role specific dashboards and apps could be applied to other neonatal devices or to resource constrained hospitals in different regions. The work described in this report is therefore best seen as a foundation and it shows that the pieces can be made to work together and that clinicians and parents are open to such tools, but many technical, clinical and organisational questions remain to be answered in future work.

Chapter 8

Conclusion

This project designed and implemented NeoCare, an automated incubator monitoring and decision support system that augments existing incubators using a Raspberry Pi device edge node, a self hosted ThingsBoard CE back end, GCP services, and role specific web and mobile dashboards. On the edge device, three main components were deployed: (i) a MobileNetV3 Small based jaundice model with quality-aware gating, (ii) a two stage cry pipeline (YAMNet based detection followed by ensemble cry type classification), and (iii) a YOLOv8n + OCR LCD reader for non invasive extraction of incubator vitals. In addition, an NTE recommendation engine encoded the national neutral thermal environment table and produced explicit age and weight dependent temperature guidance.

End to end validation confirmed that data produced at the incubator can be captured and processed at the edge, forwarded via MQTT to the cloud, and presented in a unified clinical view and a restricted parent view with acceptable delay. The secured reverse proxy approach using Tailscale and Nginx enabled remote access to live video and edge APIs without directly exposing the Pi to the public network. A limited NICU test run at the National Hospital Galle, conducted under permission and supervision, further demonstrated feasibility of operating the integrated system in the intended environment.

The work has some limitations. The machine learning models that were used in this work were trained on datasets that are not very big and do not really show what you would see in real life. The LCD reader in this work can be affected by where you put the camera, glare and focus. The way this work classifies cry types has not been tested in a NICU with all the noises you would normally hear. The cloud part of this work needs to be set up with things, like VPN and proxy layers. This work has not been checked for security problems. Looked at to see if it meets the rules. It also has not been validated in a setting to see if it really works. So NeoCare is really a research project and it helps doctors make decisions it is not something you would use on patients now. What we found out is that using normal computers and open source tools can be a good way to build a system for monitoring babies and this is something that could be used in hospitals in the future NeoCare is a good starting point for this kind of thing and it shows that a system like this can be built using normal parts and still work well and that is what NeoCare is all, about systems.

Appendix A

Ethical Considerations

The work described in this report touches on several ethically sensitive areas, including neonatal care, the recording and processing of images and audio of infants, and the handling of health related data. Although the prototype has not yet been deployed in routine clinical use, and no identifiable patient data from real NICU admissions were recorded or stored as part of this final year project, it is important to outline the ethical considerations that guided the design and to note what additional steps would be required for future clinical studies.

First, no real patient identifiers were used in the development of the system. All model training was performed on public or synthetic datasets, jaundice models were trained using the Kaggle Jaundice Image Data set, cry models were trained on publicly available or team-curated audio recordings that did not contain personally identifying information, and the incubator LCD reader relied on images of displays that did not include patient names or IDs. For demonstrations and internal testing of the web and mobile interfaces, dummy baby names and identifiers were used. No actual NICU patient records, identifiers or clinical notes were accessed.

Second, the system was deliberately designed so that, at this stage, it operates as an advisory tool rather than as a device that makes or enforces clinical decisions. The NTE engine encodes published guidelines and presents recommendations, the jaundice and cry models provide flags and probabilities and the dashboards display these outputs alongside existing vitals, but the software does not automatically adjust incubator settings or override staff decisions. This design respects the principle that clinical judgement remains with qualified professionals and reduces the risk of harm due to misclassifications or technical failures in an immature prototype.

Third, privacy and data protection concerns were taken into account from the outset. The architecture avoids exposing the Raspberry Pi edge devices directly to the public internet and they are reachable only over an encrypted Tailscale VPN through a router VM and reverse proxy [45, 46]. All external access to the web dashboards and mobile APIs is conducted over HTTPS, and access is gated by authentication and role based authorisation. The parent interfaces intentionally avoid showing raw vital signs or alarm states, presenting only live video and high level status, to reduce the risk of misunderstanding and undue anxiety. Nevertheless, a more complete privacy impact assessment and threat analysis would be necessary before any clinical deployment, especially to ensure compliance with local health-data regulations.

In the context of this student project, formal institutional ethics approval was not required because no real patient data were collected or analysed, and all demonstrations to clinicians and parents involved fictional or simulated data. For future work that would involve recording real infant images or audio, logging real vitals, or running the system on live incubators in a NICU, appropriate approvals from an institutional ethics review board (IRB) or equivalent body would be essential. Such studies would need clear protocols covering informed consent (including for parents or guardians), data storage and retention, access control, and procedures for handling incidental findings or system failures.

Finally, it is worth noting that the system has potential implications for staff workloads and parent experience. While the intent is to support clinicians and reassure parents, poorly designed or over sensitive alerts could contribute to alarm fatigue, and continuous video access could increase parental anxiety if not accompanied by proper explanation and support. These human factors considerations underline the need for careful, iterative evaluation with all stakeholders not only from a usability standpoint, but also from an ethical perspective to ensure that the technology is used in ways that genuinely benefit infants, families and staff.

In summary, the current prototype was developed using synthetic and public data, with advisory only functions and basic technical measures to protect privacy and security. Any move towards clinical deployment will require formal ethical review, stronger guarantees around data protection, and close collaboration with clinical and ethics committees to ensure that the system is introduced in a safe, transparent and ethically sound manner.

Bibliography

[1] "Preterm birth." https://www.who.int/news-room/fact-sheets/detail/preterm-birth, 2023. [Online; accessed 7-March-2025].

[2] "Why premature babies' survival often depends on where they were born." https://www.doctorswithoutborders.org/latest/why-premature-babies-survival-often-depends-where-they-were-born, 2024. [Online; accessed 7-March-2025].

[3] "Barriers and enablers of quality high-acuity neonatal care in sub-Saharan Africa: protocol for a synthesis of qualitative evidence," *BMJ Open*, vol. 14, no. 3, p. e081904, 2023.

[4] "Challenges posed by infant incubators and their potential mitigation," *ResearchGate*, 2023. [Online; accessed 8-March-2025].

[5] "Risk factors of preterm birth in Sri Lanka: case-control study," *ResearchGate*, 2023. [Online; accessed 8-March-2025].

[6] "Premature birth - Symptoms and causes." https://www.mayoclinic.org/diseases-conditions/premature-birth/symptoms-causes/syc-20376730, 2023. [Online; accessed 8-March-2025].

[7] "Infant Incubators." http://www.frankshospitalworkshop.com/equipment/infant_incubators_equipment.html, 2008. [Online; accessed 10-March-2025].

[8] "The Infant mortality rate in Sri Lanka (2021 - 2029, per 1000 live births)." https://www.globaldata.com/data-insights/macroeconomic/the-infant-mortality-rate-in-sri-lanka-218166/, 2025. [Online; accessed 10-March-2025].

[9] "Infant Mortality Rate for Sri Lanka (SPDYNIMRTINLKA)." https://fred.stlouisfed.org/series/SPDYNIMRTINLKA, 2024. [Online; accessed 11-March-2025].

[10] G. B. Merenstein and S. L. Gardner, *Handbook of Neonatal Intensive Care*. Philadelphia, PA, USA: Mosby Elsevier, 6th ed., 2006.

[11] "Contactless heart rate measurement in newborn infants using a multimodal 3D camera system," *PubMed Central*, 2022. [Online; accessed 12-March-2025].

[12] "Advancements and Innovations in Thermodynamics for Infant Incubators: A Review," *International Journal of Heat and Technology - IIETA*, 2023. [Online; accessed 15-March-2025].

[13] "Baby Incubator Explained: Features, Uses, and Benefits." https://eureka.patsnap.com/blog/what-is-baby-incubator/, 2024. [Online; accessed 12-March-2025].

[14] "NeoBeat - Newborn Heart Rate Meter." https://shop.laerdalglobalhealth.com/product/neobeat/, 2025. [Online; accessed 12-March-2025].

[15] "Optimizing Impedance Respiration Rate Monitoring for Neonates." https://clinicalview.gehealthcare.com/quick-guide/optimizing-impedance-respiration-rate-monitoring-neonates, 2024. [Online; accessed 14-March-2025].

[16] "Respiration Monitor — Apnea Monitor." https://www.niceneotech.com/neonatal-care/respiration-monitor/, 2018. [Online; accessed 13-March-2025].

[17] "A Smart Incubator System for Monitoring and Controlling Premature Babies' Environment Using Machine Learning," *IIETA*, 2024. [Online; accessed 13-March-2025].

[18] "Good Signals from Microcontroller Market," *Medical Device Network*, 2024. [Online; accessed 13-March-2025].

[19] "Microcontroller Applications in Medical Devices." https://www.vemeko.com/blog/microcontroller-applications-in-medical-devices.html, 2024. [Online; accessed 13-March-2025].

[20] "DEVELOPMENT AND CONTROL OF SMART INCUBATOR SYSTEM FOR PREMATURE BABIES," *ResearchGate*, 2024. [Online; accessed 14-March-2025].

[21] "SMART IOT BASED INFANT INCUBATOR SYSTEM," *IJESR*, 2024. [Online; accessed 14-March-2025].

[22] "BG29 Small Bluetooth Microcontroller Ideal for Medical Devices." https://www.silabs.com/blog/bg29-small-bluetooth-microcontroller-ideal-for-medical-devices, 2025. [Online; accessed 14-March-2025].

[23] "SMART INFANT INCUBATOR MONITORING AND CONTROL SYSTEM USING IoT," *ResearchGate*, 2024. [Online; accessed 14-March-2025].

[24] "Use a Portfolio of Low-Power Microcontrollers to Simplify Healthcare and Industrial IoT Design." https://www.digikey.com/en/articles/use-a-portfolio-of-microcontrollers-for-healthcare-industrial-iot-design, 2025. [Online; accessed 14-March-2025].

[25] "C8051F96x Ultra-Low Power Microcontroller." https://www.silabs.com/mcu/8-bit-microcontrollers/c8051f96x, 2024. [Online; accessed 14-March-2025].

[26] L. Spahić, U. Sredović, Z. Kurpejović, E. Mrdanović, G. Pokvić, and A. Badnjević, "Machine learning for improved medical device management: A focus on infant incubators," *Sage Journals*, 2025. [Online; accessed 15-March-2025].

[27] "Developing a smart system for infant incubators using the internet of things and artificial intelligence." https://www.slideshare.net/slideshow/developing-a-smart-system-for-infant-incubators-using-the-internet-of-things-a268311519, 2024. [Online; accessed 15-March-2025].

[28] "Smart Infant Incubator," *The International Undergraduate Research Conference*, 2025. [Online; accessed 13-March-2025].

[29] "Sri Lanka Medical Device Registration." https://omcmedical.com/sri-lanka-medical-device-registration/, 2025. [Online; accessed 15-March-2025].

[30] "Medical Devices." https://www.nmra.gov.lk/pages/medical-devices, 2024. [Online; accessed 16-March-2025].

[31] "NMRA and MDEC: Safeguarding Medical Device Safety in Sri Lanka." https://sjhospital.lk/ensuring-medical-device-safety-in-sri-lanka-the-critical-role-of-nmra-and-mde 2022. [Online; accessed 16-March-2025].

[32] "Medical Devices Registration in Sri Lanka." https://mavenprofserv.com/medical-devices-registration-in-sri-lanka/, 2025. [Online; accessed 16-March-2025].

[33] "MEDICAL DEVICE REGISTRATION AND APPROVAL IN Sri Lanka." https://arazygroup.com/ivd-medical-device-registration-sri-lanka/, 2022. [Online; accessed 16-March-2025].

[34] "GUIDE TO NEONATAL INTENSIVE," tech. rep., Perinatal Society of Sri Lanka, 2022. [Online; accessed 17-March-2025].

[35] "Adherence to infection control practices in relation to neonatal care in major hospitals in a district of Sri Lanka," *ResearchGate*, 2020. [Online; accessed 17-March-2025].

[36] "RESUSCITATION OF THE NEWBORN," tech. rep., Sri Lanka College of Paediatricians, 2019. [Online; accessed 17-March-2025].

[37] "10 Best Practices for infant incubator and radiant warmer testing." https://a.flukebiomedical.com/Infant_Incubator_and_Radiant_Warmer_Testing, 2024. [Online; accessed 17-March-2025].

[38] J. Liam, "mjpg-streamer: A video streaming application for linux-uvc." https://github.com/jacksonliam/mjpg-streamer, 2013. [Online; accessed 08-09-2025].

[39] e. S. R. Lelwala, "Neonatal incubator edge device pi services." https://github.com/sahanrashmikaslk/Neonatal_incubator_edgeDevice_PI-services.git, 2025. [Online; accessed 03-04-2025].

[40] e. S. R. Lelwala, "Neonatal incubator display reader (yolov8 + ocr)." https://github.com/sahanrashmikaslk/Neonatal_incubator_displayReader, 2025. [Online; accessed 05-05-2023].

[41] e. S. R. Lelwala, "Advanced neonatal jaundice detection system." https://github.com/sahanrashmikaslk/Neonatal_jaundice_detection.git, 2025. [Online; accessed 05-05-2023].

[42] e. H. P. Madhushani, "Cry-detection-classification-model." https://github.com/HasiniPrasadika/Cry-Detection-Classification-Model/tree/main/cry_project, 2024. [Online; accessed 05-05-2023].

[43] e. S. R. Lelwala, "Nte recommendation engine for infant incubators." https://github.com/sahanrashmikaslk/NTE_recommendation_engine.git, 2025. [Online; accessed 05-05-2023].

[44] e. S. R. Lelwala, "Neonatal incubator monitoring system – thingsboard configurations." https://github.com/sahanrashmikaslk/Neonatal_incubator_monitoring_system_TB, 2025. [Online; accessed 05-05-2023].

[45] e. S. R. Lelwala, "Nicu incubator monitoring with thingsboard integration." https://github.com/sahanrashmikaslk/incubator_monitoring_with_thingsboard_integration, 2025. [Online; accessed 05-05-2023].

[46] T. Inc., "Tailscale documentation." https://tailscale.com/kb, 2024. [Online; accessed 12-08-2025].

[47] S. L. Ministry of Health, *National Guidelines for Newborn Care*. Family Health Bureau, Colombo, Sri Lanka, 2014.

[48] A. Howard, M. Sandler, G. Chu, *et al.*, "Searching for mobilenetv3," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 1314–1324, 2019.

[49] A. Olapo, "Jaundice image data." https://www.kaggle.com/datasets/aiolapo/jaundice-image-data, 2020. [Online; accessed 08-05-2025].

[50] ThingsBoard, "Thingsboard documentation." https://thingsboard.io/docs, 2024. [Online; accessed 24-06-2025].

[51] Google, "Yamnet: Audio event classification model." https://tfhub.dev/google/yamnet/1, 2019. [Online; accessed 14-06-2025].

[52] B. McFee, C. Raffel, D. Liang, *et al.*, "librosa: Audio and music signal analysis in python," in *Proceedings of the 14th Python in Science Conference (SciPy)*, pp. 18–25, 2015.

[53] e. H. P. Madhushani, "Nicu mobile app." https://github.com/HasiniPrasadika/NICU_Mobile_App.git, 2025. [Online; accessed 05-05-2023].

[54] e. S. R. Lelwala, "Neonatal incubator display simulator." https://github.com/sahanrashmikaslk/Neonatal_incubator_displaySimulator.git, 2025. [Online; accessed 04-07-2025].

[55] Ultralytics, "Yolov8: Real-time object detection." https://docs.ultralytics.com, 2023. [Online; accessed 21-07-2025].

[56] R. Smith, "An overview of the tesseract ocr engine," in *Proc. Int. Conf. Document Analysis and Recognition (ICDAR)*, (Curitiba, Brazil), pp. 629–633, 2007.

[57] J. AI, "Easyocr: Ready-to-use ocr with 80+ languages supported." https://github.com/JaidedAI/EasyOCR, 2020. [Online; accessed 12-07-2025].

[58] R. Team, "React – a javascript library for building user interfaces." https://react.dev, 2024. [Online; accessed 01-04-2025].

[59] C. Contributors, "Chart.js documentation." https://www.chartjs.org/docs/latest, 2024. [Online; accessed 16-06-2025].

[60] Google, "Flutter documentation." https://docs.flutter.dev, 2024. [Online; accessed 17-05-2025].

[61] Google, "Firebase cloud messaging documentation." https://firebase.google.com/docs/cloud-messaging, 2024. [Online; accessed 18-05-2025].

[62] G. Cloud, "Google cloud documentation." https://cloud.google.com/docs, 2024. [Online; accessed 05-05-2023].

[63] G. Cloud, "Cloud run documentation." https://cloud.google.com/run/docs, 2024. [Online; accessed 05-05-2023].

[64] G. Cloud, "Cloud sql documentation." https://cloud.google.com/sql/docs, 2024. [Online; accessed 05-05-2023].

[65] I. E. Commission, *IEC 60601-2-19: Medical Electrical Equipment – Part 2-19: Particular Requirements for the Basic Safety and Essential Performance of Infant Incubators*. IEC, Geneva, Switzerland, 2009.