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

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. Gayathri Thilakarathne	Mr. Neel Karunasena
(Supervisor)	(Co-Supervisor)

Abstract

This project fulfills the basic need for improved neonatal care, particularly for premature infants in Sri Lanka and other developing countries, where access to advanced medical technology and trained personnel could be limited. Traditional infant incubators, though essential, will often need to be constantly manned to closely regulate critical environmental parameters such as temperature, humidity, and oxygen concentration, placing an enormous strain on healthcare resources. This project recommends the use of a sophisticated automated system not only to keep track of and control such critical environmental parameters but also to continuously monitor the baby's vital signs, thus offering a more pro-active and responsive care. With a cutting-edge set of sensor technology for real-time data acquisition, a smart, high-performance microcontroller-based platform for processing, and advanced machine learning algorithms for predictive analytics and pattern recognition, the incubator system proposed here will dynamically and automatically control environmental parameters according to the infant's physiological needs. Besides, the system will continuously monitor key vital signs, such as heart rate, respiratory rate, and oxygen level, and alert healthcare workers automatically to any extreme deviations or health risks so that immediate intervention can be made. This autonomous solution will immensely enhance precision, consistency, and effectiveness of neonatal care, reduce tremendously the workload of uninterrupted manual intervention on physicians and nurses, and ultimately enhance health outcomes and survival rates of vulnerable preterm babies, especially in resource-limited settings where such technological innovation can be the deciding factor in saving young lives and promoting long-term health.

Acknowledgments

We are pleased to provide our sincere gratitude to our supervisor, Ms. GCW Thilakarathne, for her valuable guidance, inspiration, and dedication towards this undergraduate project proposal, "Development of an Automated Condition Controlling and Monitoring System for an Infant Incubator". Her expertise and insightful suggestions were instrumental in developing this project. We ~~also want to~~ thank our co-supervisor, Mr. Neel Karunasena, for his useful guidance and advice in preparing this proposal. Also, we ~~want to~~ express our appreciation to Dr. Iromi Ranaweera, the module Coordinator of module EE7802 Undergraduate Project, for the guidance given. We also extend our sincere thanks to the Director of the National Hospital, Galle, Dr. Nirmala Waththuhewa and all the dedicated Neonatal Intensive Care Unit (NICU) staff members for providing the essential resources (incubators) and valuable knowledge, which will be crucial for the practical implementation and testing of our project. Lastly, our gratitude extends to everyone who ~~were~~ contribute their support and guidance towards the success of this project.

Contents

List of Figures

List of Tables

Acronyms

AC	-	Alternating Current
DC	-	Direct Current
LV	-	Low Voltage
MV	-	Medium Voltage

Chapter 1

Introduction

- Critical Challenges in Neonatal Care for Premature Infants

  Premature birth is a leading killer of children below 5 years globally. An estimated 13.4 million premature babies were born in 2020, prior to 37 weeks of gestation. Preterm complications are the most frequent cause of more death in people of this age, totaling around 900,000 people in 2019.Many infants who survive preterm birth face a lifetime of disabilities, including learning impairments, as well as visual and hearing problems. Globally, prematurity stands as the leading cause of death in children under the age of five years [1, 2].

  Survival rates for preterm babies are drastically different between high-income and low-income countries. Inequalities in survival are stark, with half of the babies born at or below 32 weeks dying in low-income settings due to a lack of feasible and cost-effective care, such as warmth, breastfeeding support, and basic infection and breathing difficulty management. In contrast, almost all these babies survive in high-income countries. The suboptimal use of technology in middle-income settings is also contributing to an increased burden of disability among preterm babies who survive the neonatal period. For instance, over 90% of extremely premature babies (less than 28 weeks) born in low-income countries die within the first few days of life, whereas less than 10% die in high-income settings. This significant disparity underscores the urgent need for solutions tailored to the specific challenges of resource-limited environments [1, 2].

  Developing nations have great difficulty in ensuring proper neonatal care owing to a lack of resources and infrastructure. Neonatal intensive care units in sub-Saharan Africa, for instance, frequently struggle with constrained resources and systemic issues, resulting in lower quality of care and increased infant mortality. Such barriers include poor availability of equipment, supplies, clinical guidelines, and an adequate number of trained and motivated health care professionals. In addition, inadequate infrastructure and challenges in obtaining crucial medicines and commodities exacerbate these difficulties [3].

  Such extreme inequality in survival underlines the urgency of solutions adapted to resource-scarce settings.The stark contrast in fate for preterm babies depending on the birth location is indicative of one of the key global health disparities. Bridging this disparity requires thinking outside the box and solutions that can deliver proper care under resource-poor circumstances. This project is motivated by the potential to close this gap through a robotic and intelligent infant incubator system with the constraints and requirements of developing countries

as a special focus.

- Limitations of Current Infant Incubation Technologies

  Traditional incubators are classically dependent on continuous human monitoring for parameter adjustment. Healthcare workers must continuously observe and manually adjust factors like temperature, humidity, and oxygen levels to ensure the health of the infant. Continuous intervention places a huge burden on already overwhelmed medical staff, particularly in under-equipped institutions common in developing countries.

  Existing incubators may have defective thermoregulation, wasted energy, and potential electromagnetic field (EMF) hazards. Certain infant incubators fail to complete thermoregulation appropriately and fail to consider all radioactive, conductive, convective, and evaporative exchanges of the ambient environment. This can lead to temperatures differing in the hood, and differences of ±0.8°C have been observed in some research. Also, a significant amount of the electrical energy consumed by existing incubators, which usually averages between 85%, is directed towards resistive heating components and leads to inefficiency in energy. Some incubators also cannot lower temperature if ambient temperature is elevated and can expose the infant to the risk of hyperthermia. Also, there are worries regarding the potential risks of EMF emissions from incubators, with emissions greater than 200mG ~~having the potential to~~ interfere with melatonin production or vagal tone in infants and caregivers [4].

  ~~They cannot offer sufficient~~ protection from the high noise levels ~~of~~ NICUs and ~~can be sources~~ of noise themselves. Newborn incubators rarely offer sufficient protection from the high noise levels of neonatal intensive care units. The fan structure within the incubator itself has the potential to produce noise in the frequency range 1.3–1.5 kHz with a weighted sound pressure level of 40.5 dB. Although the incubator design may limit the frequency range of externally transmitted sound to lower frequencies, it will not be effective in attenuating transient sounds such as alarms or opening and closing of cabinet doors. This continuous exposure to excessive levels of noise can ~~be~~ disrupt~~ive~~ of sleep and ~~recreation and~~ can result in long-term hearing impairment and unfavorable neurological development in preterm neonates [5].

  ~~Others lack features like~~ vital sign monitoring and connectivity. While most traditional incubators offer features like thermoregulation, relative humidity control, and oxygen level control, ~~they~~ lack embedded systems for continuous vital sign monitoring and connectivity for remote access and data analysis. Lacking these features limits the ~~potential for proactive response to and detection of~~ changes in the infant's status, once more pointing out the need for constant human monitoring [4].

  Though incubators play the critical function of maintaining a controlled environment for susceptible newborns, their lack of full automation, ~~environmental control,~~ noise reduction, and monitoring features ~~identifies key areas of substantial improvement needed to advance the quality of care, particularly in resource-poor environments. It is~~ these gaps this project aims to fill with novel design and technology development.

- Proposed Project: Automated Infant Incubator Controlling & Monitoring System

  The project ~~will~~ design an automated monitoring and controlling system for infant vital signs and incubator environments. The new system will automatically control important environmental variables in the infant incubator, such as temperature, humidity, and oxygen concentration. At the same time, it will measure important infant vital signs like heart rate and respiratory rate through embedded sensors.

  Machine learning will be implemented on predictive health analysis to enable more personalization of care. The system will, ~~by activating the use of~~ machine learning algorithms, analyze trends in the baby's vital signs ~~for the prediction of~~ probable distress or abnormality. The predictive mode will enable intervention ~~beforehand~~, ~~enabling more personalization and possible avoidance of the need for~~ continuous human observation.

  The autonomous nature of the system can be of immense use when it comes to resource limited environments. By reducing the ~~amount of constant~~ human intervention to fine-tune pa-rameter settings and by providing early indication of potential health issues, the computer-based system offers a pragmatic solution for the improvement of neonatal care in arid areas where medical facilities and staff may be in short supply. This project has great potential to significantly improve preterm infants' survival and overall health condition, particularly in the Third World.

  A specially crafted mobile app and user interface will provide real-time incubator status and baby vital signs to caregivers. The push notification and remote control features in the app will enable ~~better~~ responsiveness and convenience. ~~L~~ive camera integration ~~will~~ enable parents to see their baby in the incubator through the web application or mobile application. Access to the ~~camera~~ stream ~~will be~~ securely controlled by the health care professionals with the authority to deny or grant access according to their choice. Not only ~~does this feature~~ ~~offer~~ greater transparency, but ~~it also gives~~ parents ~~peace of mind while maintaining stringent security and privacy measures.~~

1.1 Background

1.1.1 High risks faced by premature infants in developing countries

Approximately 13.4 million preterm infants were born in 2020, before 37 completed weeks of gestation. It is a significant worldwide health problem, and preterm birth rates vary from 5% to 18% all over the globe. The majority of these preterm births, approximately 90%, occur in African and Asian countries [1, 2].

Preterm birth complications account for more child deaths under the age of 5. ~~During~~ 2019, an estimated 900,000 died due to complications from being born prematurely. Premature birth is the major cause of death in ~~children under the age of five years old worldwide~~. The sheer number of acute and chronic complications of prematurity

places a heavy burden on health systems, communities, and families [1, 2].

Half of those born at or before 32 weeks die in low-income settings due to the lack of basic care. The age-old issue of poor access to low-cost resources such as provision of heat, breastfeeding care, and infection and breathing disease treatment makes a major contribution to this over-mortality. For example, in poor countries, more than 90% of extremely premature babies (below 28 weeks) die within a few days of birth. This contrasts starkly with the high-income countries in which almost all such babies survive [1].

Premature babies have short-term complications like breathing problems, heart problems, brain problems, temperature control problems, gut problems, blood problems, metabolism problems, and immune system problems. Premature babies can have breathing problems within the first few weeks of life since their lungs are not fully developed, leading to respiratory distress syndrome and apnea. Heart problems such as patent ductus arteriosus and hypotension are also felt. The risk of intraventricular hemorrhage, or cerebral bleeding, rises in earlier pregnancies. Premature babies lose body heat easily as they have no fat stores and are unable to generate sufficient heat to replace it, leading to hypothermia. The guts of premature babies can be immature, increasing the risk of necrotizing enterocolitis. Disorders of the blood like anemia and jaundice in newborns also occur. Besides, preterm infants normally experience metabolism and immune problems, making them vulnerable to low blood sugar and infections like sepsis [3].

Long-term consequences can include neuro-developmental delays, vision and hearing impairment, chronic respiratory conditions, and psychological and behavioral effects. Preterm birth can lead to long-term mental and developmental disability, affecting physical growth, learning, communication, self-care, and social interaction. Preterm birth is linked with cerebral palsy, attention deficit hyperactivity disorder, anxiety, and depression. Neurological disorders and lung problems, such as bronchopulmonary dysplasia, may also persist. Preterm babies may also develop dental problems, hearing loss, vision problems such as retinopathy of prematurity, and intestinal problems [4].

The severe morbidity and mortality of preterm delivery, especially in low-income countries, provide strong rationale for the undertaking of this project. The associated high risk and long-term health problems of preterm babies, along with differential survival rates between low- and high-income countries, make it essential to find novel solutions. This project aims to address such significant issues through the development of an automated infant incubation system which is capable of providing maximum care, particularly in situations where access to specialist health care personnel and technology may be limited.

1.1.2 Challenges of traditional infant incubators

Traditional incubators require constant human monitoring. Medical professionals are tasked with continuous observation and manual regulation of many parameters in the

incubator, including temperature, humidity, and oxygen concentration, to ~~provide~~ a stable and appropriate environment for the premature infant. This ~~requires~~ constant observation and intervention, ~~which~~ places a heavy workload on nursing staff and can lead to delays in adjustment ~~if staff are attending to other critical needs.~~ They can have inaccurate thermo-regulation, with fluctuation in temperatures. ~~Most~~ existing infant incubators ~~cannot~~ maintain an accurate and stable temperature within the incubator hood. ~~These systems~~ do not sufficiently account for the complex interaction of radiative, conductive, convective, and evaporative heat transfer between infant and environment. This may produce temperature fluctuations, which have the potential to induce hypothermia or hyperthermia in the infant with negative health consequences. Temperatures of as high as ±0.8°C ~~are~~ reported ~~with some commercial resistance incubators~~ [5].

~~Most of them possess a heating unit and lack a cooling feature if the ambient temperature is high. Most commercially available infant incubators nowadays are commercial resistance incubators that primarily employ a resistive heating element to increase the temperature within the incubator.~~ These models are typically designed to operate when the ambient temperature is below the desired temperature for the infant. However, they lack a cooling mechanism to reduce the temperature if the ambient temperature rises above the level required, which can occur in various geographical locations, particularly during transportation in sunny weather. Lack of cooling is a high risk of hyperthermia for the premature infant [5, 6].

EMF radiation ~~from some incubators can be dangerous. Risks have been voiced with regard to the potential harmful effect of EMF radiation from infant incubators on both the preterm infants and the health care workers in close contact with them.~~ EMF levels above 200mG have been suggested to interfere with melatonin release and vagal tone. ~~While~~ the entire range of such effects remains to be explored, reduction of exposure to EMF in infants is advised by redesigning incubator components with plastic instead of metal and increasing the gap between the bed and the emission source, and using ferro-absorbing panels for protection [5].

Internal incubator noise and external noise from the NICU environment are not adequately minimized and tend to have a negative impact on the health of the infant. The incubators for newborn infants ~~have a tendency to offer~~ insufficient protection from the high levels of noise encountered within the NICU. The incubators themselves also generate noise, primarily from the internal fan for air circulation, able to ~~generate~~ sound in the 1.3–1.5 kHz frequency ~~band~~. Transient alarm and incubator operating sounds, for example, opening and closing doors are also not well attenuated. These high levels of noise ~~in the NICU~~ ~~have~~ been shown to ~~cause a number of harmful health problems in~~ preterm infants, ~~including~~ sleep disturbance, stress, and changes in physiological reactions such as heart and respiratory rates [5, 6].

Standardization of relative humidity levels is lacking, and ~~variable~~ practices ~~are the outcome~~. An adequate ~~relative humidity (RH)~~ level in the infant incubator is ~~necessary~~ for the health of preterm infants ~~and has an effect on parameters such as~~ trans-epidermal water loss, hypothermia, electrolyte balance, oxygen consumption, risk of infection, and skin integrity. ~~There is a noted lack of standardization, though,~~

in the handling of RH levels in clinical practice. This variation in the use of RH levels, along gaps in nursing practice in incubator handling are dangerous to the premature infant [5].

Temperature distribution within the incubator may be slow and non-uniform. To prevent health risks to the premature infant, it is required that the infant incubator should be able to maintain the temperature in the hood at a uniform and constant level. Incubators available today may suffer from slow and non-uniform temperature distribution, which can lead to localized areas of raised or lowered temperature, potentially causing discomfort or physiological distress to the infant. Ideally, the incubator should have the ability to quickly reach the desired temperature and maintain it evenly across the hood [5].

Special care and procedures may be limited with closed incubator designs. Although the closed construction of traditional incubators guarantees a controlled environment and protection from infection, sometimes it can become a hindrance to quick and easy access to the infant for special treatment or procedures such as controlled ventilation, pulse oximetry, or ECG monitoring. In those situations, open system in which the infant is heated with a radiant warmer hung above the cradle would be preferred for allowing Enhanced accessibility for doctors and nurses [7].

The resolved limitations of the traditional infant incubators, ranging from primitive environmental control to security and functionality issues, indicate the need for a sophisticated and automatic system. Overcoming these defects can bring significant improvements in neonatal therapy, particularly where efficiency and reduced dependency on constant manual intervention are crucial, i.e., resource-limited environments.

1.1.3 Neonatal Care in Sri Lanka

Sub-Saharan Africa and other poor regions have limited resources and structural problems in NICUs. Their neonatal intensive care units have inferior quality of care, increased infant mortality, and patient and worker discontent due to the same. Quality neonatal care is only possible if sufficient equipment, materials, clinical protocols, trained and motivated health workers, and supportive supervision and client satisfaction are ensured.

Barriers include inadequate equipment, inadequate staff training, inadequate infrastructures, and drug shortages. Most healthcare workers in developing countries usually cite inadequate equipment, inadequate staff training, and inadequate infrastructure as key hindrances to delivering adequate neonatal care. Parents also complain about poor availability of medicines and services, poor cleanliness in facilities, and issues about the expense of care. These are compounded by factors like proximity to hospitals, lack of transportation, and poor parental education.

Prematurity in Sri Lanka was 10.9% in a recent study. A Sri Lankan study involving 9130 live-born neonates found 10.9% were pre-term (prior to 37 weeks of gestation). This is a high rate of preterm deliveries that require critical care in the country [8].

Infant mortality in Sri Lanka decreased significantly throughout the decades but remains still an issue 15. Even though Sri Lanka has developed significantly in restraining infant deaths since the mid-20th century, from approximately 100 deaths per 1000 live births in 1950 down to 5.6 deaths per 1000 live births in 2022, the There is still a demand for continuous improvement in neonatal care. The infant mortality rate in 2024 was 6.80 per 1000 live births [9, 10].

Shortage of equipment, training of staff, and infrastructure are the problems in neonatal care in Sri Lanka. Similar to other developing countries, healthcare professionals in Sri Lanka also point towards the lack of necessary equipment, inadequate staff training in neonatal care, and poor infrastructure as key obstacles in the delivery of optimal care to newborns. These problems can impact the ability to provide timely and effective interventions to preterm and sick babies.

To have Sri Lanka as a context provides a specific environment and allows for consideration of local issues and health infrastructure. To target the project on the needs and within the confines of a lower-middle-income Asian country like Sri Lanka, the advanced automated incubator system can be specifically tailored for managing the prevalent problem in this region. Having the knowledge of the regional healthcare facilities, prevalence of preterm births, and the existing limitations in neonatal care in Sri Lanka will render the solution proposed relevant, feasible, and able to generate significant large-scale positive impact on infant health outcomes within this specific setting.

Some of the important parameters such as temperature, humidity, weight of the infant are used to develop this automatic infant incubator system. These parameters are significant as they provide full details of a baby's physiological state and each one of them is directly associated with infant health in important ways.

The Neutral Thermal Environment (NTE) Temperature Guidelines provide precise temperature ranges based on the infant's age and weight, ensuring that they remain in a safe and controlled environment and the data is given below in Table **??**.

Neonatal incubators that are used in Sri Lanka do help in keeping the right temperature for infants, which is really important for growth and survival. Especially for newborns, particularly those who weigh under 1500 g, warmer temperatures need to be maintained, typically between 33.0-35.4°C. Larger infants, on the other hand, actually need those slightly lower temperatures since they do not require as much warmth. When an infant gets older, which happens gradually, the temperature that is needed is set to decrease. Infants older than two weeks usually require around 29.0-34.0°C, which depends on their weight, of course. By the time they reach five to six weeks, the temperature in the incubators tends to go down even more as the infant's ability to handle temperature changes improves. This type of controlled environment is supposed to help avoid issues like hypothermia, and it is meant to ensure that proper care for neonates is provided, which ultimately, in the best-case scenario, improves survival rates [**?**].

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		
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

Age and Weight	Starting Temperature (°C)	Range of Temperature (°C)
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 is generally 36.5°C to 37.5°C (97.7°F to 99.5°F) ~~in newborns.~~ Hypothermia (below 36.5°C) or hyperthermia (above 37.5°C) can be life-threatening and these temperature values are ~~indicated in the reference Book and the data is given~~ below in Figure 1.1. ~~(National Guidelines for Newborn Care published by the Ministry of Health)~~

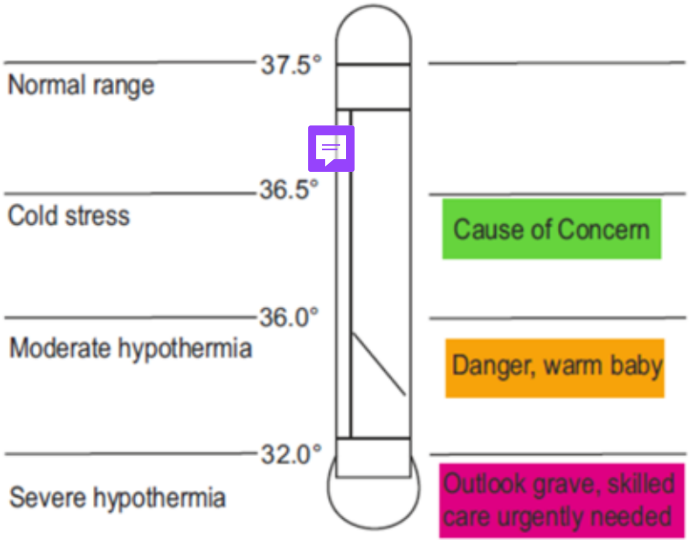

Figure 1.1: Grading of hypothermia

A ~~child~~'s weight, especially ~~for~~ premature or low birth weight babies, is a ~~good~~ indicator of ~~the child~~'s growth ~~and there are special~~ standard treatments for low birth weight babies ~~in Standard Treatment Protocols reference Book.~~ The humidity level

is a very important parameter to keep an infant moist. Excess or low humidity can cause respiratory issues, dehydration or skin illness in infants. To prevent excess loss of water and make the environment pleasant for the infant, the humidity level in incubators is maintained at around 50-60% [11, 12].

Due to the above reasons, the automated system is designed on the basis of those parameters.

1.2 Problem Statement

The need for minimal human intervention in the regulation of incubator parameters. Traditional infant incubators need high-level manual regulation of environmental parameters by medical personnel, which may be time-consuming and prone to delays, especially in under-staffed facilities. An automatic system that can regulate these parameters according to real-time information would significantly reduce this need.

The demand for precise and automated temperature, humidity, and oxygen control. Consistent and ideal environmental parameters are vital to the well-being and development of premature infants. Traditional incubators tend to be beset with inaccuracy and inconsistency in the form of these parameters. An automatic system based on accurate sensor feed and control systems is a must that supplies a stabilizing supporting microenvironment.

The need for round-the-clock real-time monitoring of infant vital parameters (heart rate, respiratory rate). Early changes in physiology should be detected so that delayed intervention is not necessary in neonatal care. Most standard incubators do not have inbuilt provisions for long-term monitoring of vital parameters. Automated real-time monitoring of these parameters would give an early warning of a possible deterioration.

The inability of conventional incubators to anticipate infant distress. Conventional incubators primarily react to present circumstances and not anticipating potential issues that can occur. Adding machine learning to examine vital sign data and predict future distress could potentially make anticipatory intervention possible and possibly improve outcomes.

The constraints of prolonging optimal neonatal care in poor-resource settings due to the constraints of dominant technologies. Lacking specialist doctors and advanced medical technology, many developing nations share this fact. The constraints brought about by standard incubators additionally contribute to the issue. A cost-efficient, automated system with enhanced monitoring and forecasting has to be applied in order to enhance neonatal care in such poor-resource settings.

An infant incubator is a sophisticated machine, which requires trained staff to run the machine properly. Having trained staff to run the machine and correct mistakes is a problem with today's infant incubators. A computer-based system, which would instantly accomplish all these functions, would make it extremely simple without re-

quiring special training and trained staff.

No remote monitoring exists in present incubators. It merely allows watching babies on behalf of parents and inhibiting the chance for doctors and nurses to check the condition of the baby without interference. There isn't any camera system for monitoring the baby's vital signs for parents and medical staff to make any decision by distant watching. An Automated System can offer online monitoring of the baby's motion inside the incubator and parents can check the status and activity of the baby at a distance, remotely.

1.3 Objectives and Scope

1.3.1 Objectives

- To design and develop an intelligent neonatal incubator system to automatically regulate temperature, humidity, and oxygen levels based on real-time sensor feedback and adaptive control algorithms.
- To integrate safe and reliable sensors for online detection of infant vital signs like heart rate and respiratory rate to offer continuous and reliable data acquisition for neonatal treatment.
- To use a microcontroller-based system for real-time data acquisition and processing to enable the seamless inclusion of sensor readings into control systems to regulate optimal incubator conditions.
- To integrate and deploy machine learning algorithms to detect early infant distress using predictive models trained on historical or simulated data to provide warnings for potential health risk.
- For designing an affordable, simple-to-use incubator system that is tailored to neonatal care for use in low-resource environments and is safe and meets regulatory needs and most importantly simple to use, reliable, and maintainable.
- To design a web and mobile-based secure user interface for remote monitoring and control to enable real-time access by medical personnel to incubator status and baby health information as well as for purposes like push messages, remote tuning, and parental live webcam control.

1.3.2 Scope

- The project will focus on the development of a functional prototype of the automated incubator system. This includes the design, fabrication and first test of a functional model demonstrating the main function of automated environmental control and significant signal monitoring.
- This will include selection and integration of an appropriate sensor and microcontroller. The project will entail careful examination and selection of temperature, humidity, oxygen, heart rate and respiratory rate sensors and a mi-

~~crocontroller platform that will accommodate the required data processing and control capabilities.~~

- Algorithm design for automatic control and future analysis will be within its purview. This encompasses the creation of a software logic to maintain the desired incubator conditions and to analyze significant signals, and to analyze potential health issues.

- Design will also consider ~~safety requirements and regulation compliance, and this is especially relevant to Sri Lankan medical devices. While the whole regulator approval process is beyond immediate scope, design will consider relevant~~ international security standards (eg, IEC 60601) and Sri Lankan National Drug Regulatory Authority (NMRA) regulations for medical ~~equipment.~~

- ~~Scope may possibly exclude manufacturing, clinical trials or complete regulatory approval at this stage. These aspects would most likely be addressed in future stages after successful prototyping of the system and after the first trial.~~

Chapter 2

Literature Review

2.1 Previous Work

2.1.1 Sensor Integration and Real-time Data Monitoring

- Sensor Technologies for Monitoring Key Parameters

  Incubators use thermistors as temperature controls in conventional units. These temperature-~~stable~~ resistors are used in both air mode and skin temperature mode, typically at a predetermined range of 23-37°C for air mode and 34-37°C for skin mode, with an accuracy of approximately +/-0.8°C and +/-0.5°C, respectively [13].

  Humidity is typically controlled by blowing warm air over water. In most incubators, warm air is blown over a water reservoir, and humidity can be controlled manually by adjusting a deflector plate above the reservoir or by changing the water temperature. The optimal humidity level is typically between 40% and 90%. Some advanced systems may use active humidifiers with humidity control features and digital displays [8, 14].

  Oxygen may be supplied by means of a control valve. Incubators have the provision of supplying supplementary oxygen to the infants who require it, with the ability to adjust and maintain the proper concentration of oxygen. A few high-end models consist of an oxygen concentration display unit and the provision of entering the oxygen concentration as a prescribed range, say 21% to 60% [8, 14].

  Advanced incubators can utilize non-contact measurement in terms of temperature. ~~Non-contact~~ sensors are utilized in certain modern incubators to monitor the baby's physiological parameters, like temperature, reducing infection ~~potential~~ and ~~providing~~ true readings ~~without making contact with the skin.~~

- Sensor technologies for neonatal heart and respiratory monitoring

  ~~ECG~~ is a reliable ~~means of heart rate measurement.~~ ~~Electrocardiography (ECG) uses~~ ~~the chest~~ electrodes ~~of the infant~~ ~~to~~ capture the electrical ~~current~~ of the heart ~~and take~~ an objective and accurate measurement of the heart rate. It is the most reliable method of newborn heart rate measurement and is recommended in international resuscitation guidelines. ~~Contactless ECG-based heart rate monitoring using dry electrodes is also explored~~ [14].

Pulse oximetry is widely used for the monitoring of heart rate and oxygen saturation [14]. Pulse oximetry uses a sensor, most often a soft wrap, on the baby's hand or foot to monitor, without intruding, the amount of oxygen in the blood and heart rate. It measures with light the changes in blood flow and oxygenation. New pulse oximeter sensors, particularly those utilizing high technologies like Masimo SET, are designed to provide accurate measurements under the circumstances of motion and low perfusion.

Contactless heart rate estimation by camera-based photoplethysmography ~~is a fresh method.~~ ~~Camera-based photoplethysmography~~ (cbPPG) is a contact-free method founded on a camera quantifying the minute variations in skin color depending on the heartbeat and ~~predicting the~~ heart rate. Studies have validated promising correlations of cbPPG with other traditional methods, including pulse oximetry and ECG.Multimodal 3D camera devices employing near-infrared cameras are being tested to improve the reliability of this contactless newborn heart rate assessment [11].

Respiratory rate ~~may~~ be measured using impedance pneumography ~~with ECG electrodes.~~ ~~Impedance pneumography is a measurement of~~ the change in electrical impedance over the thorax and upper abdomen, varying with respiration. ~~Impedance pneumography is possible with~~ the same ECG electrodes used for ~~monitoring cardiac rate, with the additional benefit of monitoring respiratory rate. Best location of electrodes, for instance, using a vector that runs across the abdomen, is of vital importance in getting good amplitude of signal and maximum sensitivity in neonates~~ [15].

There are also specific apnea/respiration monitors, particularly for the measurement of respiratory arrest (apnea) in vulnerable infants that have a tendency towards respiratory collapse. These typically employ sensors taped on the infant's head ~~and track~~ chest and abdominal movement to provide visual and acoustic alarms when ~~there is a drop below a suitable respiratory rate or respiratory arrest~~ [16].

- Importance of real-time data acquisition and processing

Real-time data allow real-time regulation of the incubator environment. ~~Continual~~ streams of data delivered by the sensors enable the automatic control system to respond ~~in a timely fashion~~ to any deviation of the infant's physiological parameters or environmental variables of the incubator, preventing and countering destabilizing deviations.Continuous monitoring can identify subtle warning signs of distress. ~~M~~onitoring the vital signs and environmental factors in real-time allows the system to detect minor changes which ~~can~~ indicate the onset of distress or abnormality in the infant, ~~thus~~ facilitating prompt intervention by healthcare professionals [17].

Table 2.1: Comparison of Neonatal Monitoring Sensors

Sensor Type	**Measured Parameter(s)**	**Technology**	**Advantages**	**Disadvantages**
Thermistor	Temperature	Electrical resistance change with temperature	Simple, cost-effective	Contact-based, accuracy can be affected by placement
Non-contact Infrared	Temperature	Detects infrared radiation emitted by the body	Non-invasive, reduces risk of infection	Accuracy can be affected by distance and environmental factors
Electrochemical	Oxygen	Measures current produced by oxygen reaction	Relatively accurate	Sensor lifespan can be limited
Optical	Oxygen	Measures light absorption at specific wavelengths	Non-contact options available	Can be affected by movement and ambient light
ECG	Heart Rate	Measures electrical activity of the heart	Highly accurate, reliable	Contact-based, requires electrode placement
Pulse Oximetry	Heart Rate, Oxygen Saturation	Measures light absorption through tissue	Non-invasive, easy to use	Accuracy can be affected by motion, low perfusion, ambient light
Camera-based PPG	Heart Rate	Detects subtle skin color changes with heartbeat	Contactless	Accuracy can be affected by ambient light and movement
Impedance Pneumography	Respiratory Rate	Measures changes in electrical impedance across the chest	Can use existing ECG electrodes	Susceptible to cardiogenic and motion artifacts
Respiration /Apnea Monitor	Respiratory Rate	Detects chest/abdominal movement	Specifically designed for apnea detection	May require separate sensors
DHT Sensor	Temperature, Humidity	Capacitive humidity sensor and thermistor	Low cost, widely available, relatively easy to interface	Lower accuracy compared to dedicated sensors, limited range
Load Cell Sensor	Weight	Strain gauge	High accuracy, reliable for weight measurement	Sensitive to overloading, may require calibration
Heart Rate Sensor (Optical)	Heart Rate	Photoplethysmography (PPG) - detects changes in light absorption due to blood flow	Non-contact options available, relatively simple to use	Accuracy can be affected by movement, ambient light, and skin tone
O2 Gas Sensor (General)	Oxygen Concentration	Various technologies (e.g., electrochemical, optical, zirconium dioxide)	Provides direct measurement of oxygen concentration	Accuracy and lifespan can vary depending on the technology used

2.1.2 Automation and Microcontroller Systems

- Microcontroller platforms in medical devices

  Microcontrollers ~~are applied~~ in the control of medical devices, sensor data processing, and automation deployment. ~~Microcontrollers~~ integrate a microprocessor, memory, and input/output peripherals onto a single chip for the purpose of delivering precise control and monitoring of medical devices. Microcontrollers are ~~the building block of most of~~ medical devices, ranging from implantable devices like pacemakers to diagnostic devices like blood glucose meters and pulse oximeters. The international market for medical devices' microcontrollers valued US$ 415 million in 2023 and is expected to grow to US$ 675 million by 2030, reflecting their growing significance in healthcare technology [18, 19].

  ~~Examples include~~ Arduino, ARM Cortex-M series, and ~~other~~ low-power microcontrollers. ~~Various microcontroller platforms find applications in the development of medical devices.~~ ~~Arduino Nano is used~~ in some smart incubator designs based on ease of use and Wi-Fi integration. The ESP8266 Wi-Fi module also finds application in internet connectivity for remote monitoring. ARM Cortex-M line microcontrollers like those from STMicroelectronics (STM32) and Analog Devices (MAX326xx) are popular for low power and high performance and suit battery-powered medical devices and wearable sensors. PIC microcontrollers from Microchip Technology are also used in medical devices. Silicon Labs offers ultra-low power microcontrollers like the C8051F96x family, which fit power-sensitive embedded systems. Renesas RA microcontrollers using the ARM Cortex-M cores are known for their performance and reliability characteristics. Frontgrade also offers the ARM Cortex-M0+ microcontrollers to utilize for applications where radiation tolerance is needed [17, 20–25].

- Capabilities for automation, data processing, and interfacing with sensors and actuators.

  Microcontrollers ~~can automatically control~~ temperature, humidity, and oxygen ~~from~~ sensor feedback. By continuous ~~supply of information through~~ temperature, humidity, and oxygen sensors, microcontrollers can ~~apply~~ control algorithms ~~for automatically controlling the operation of~~ heaters, humidifiers, and oxygen regulators ~~to an environmental condition of the incubator as desired~~ [17, 20, 21]. ~~The~~ closed-loop control system reduces human involvement and ensures a ~~constant~~ environment for the baby.

  ~~They~~ are capable of processing data from several sensors in real-time. Contemporary microcontrollers ~~have the capability to process~~ data streams from different sensors simultaneously [17, 19, 23]. Real-time processing of data makes it possible to monitor several parameters continuously, ~~for example~~, temperature, humidity, oxygen saturation, heart rate, and respiratory rate, ~~and thus get a holistic picture of the infant's~~ state and the incubator environment.

  ~~They~~ are able to interface with actuators like heaters, fans, and oxygen controllers. ~~Microcontrollers can give~~ control signals to various actuators in order to manage the incubator conditions [17, 21]. For example, ~~based on the readings from the temperature sensor, the microcontroller may enable or disable the heater or vary the fan speed in order to manage the temperature.~~ ~~Similarly, it may manage oxygen flow from a source based on the oxygen sensor readings.~~

- Microcontroller requirements for medical devices

  Microcontrollers are employed in medical device control, sensor data processing ~~and utilization,~~ and automation application. Microcontrollers are tiny computers that contain a microprocessor, memory, and input/output peripherals integrated into a product and provide precise control and monitoring of medical devices. Microcontrollers are the central component of a wide range of health care applications from implantable devices like pacemakers to diagnostic devices like glucose meters and pulse oximeters. World market for microcontrollers for medical devices was US$ 415 million in 2023 and will be US$ 675 million by 2030, which indicates their increasing use to health technology [18, 19].

  To be used in real-time medical applications, the microcontroller should ~~be able to~~ provide ~~enough~~ processing capability to facilitate functions such as data acquisition, processing, and control. In ~~such applications as automatic~~ infant incubators, this allows ~~the system to respond fast to changes in~~ the infant's condition and incubator environment ~~without wasting energy.~~ The microcontroller should also be able to facilitate wireless communication to allow data transfer and, ~~if~~ possible, remote monitoring. Ample memory space is ~~also necessary~~ to ~~hold~~ program instructions and ~~data acquired~~ [1].

Table 2.2: Comparison of Microcontroller Platforms for Medical Devices

Microcontroller Platform	**Core Architecture**	**Clock Speed**	**Memory (Flash/RAM)**	**Key Features**
Arduino Uno 328p R3	8-bit AVR	16 MHz	32 KB / 2 KB	Simple, widely used, large community support
ESP32 (Typical)	32-bit Dual-Core	Up to 240 MHz	Up to 16 MB / 520 KB	Wi-Fi, Bluetooth, more processing power and low power design
Arduino Mega 2560 R3	8-bit AVR	16 MHz	256 KB / 8 KB	More I/O pins & memory
Arduino Nano 33 IoT	ARM Cortex-M0+	48 MHz	256 KB / 32 KB	Wi-Fi, Bluetooth, Gyroscope
ESP8266	32-bit RISC	80-160 MHz	Up to 16 MB / Varies	Wi-Fi
STM32 (Various Series)	ARM Cortex-M (M0, M3, M4, M7)	Up to 600 MHz (H7 Series)	Up to 2 MB / Up to 1.4 MB (H7 Series)	Wide range of peripherals, low power options, security features
MAX326xx (Analog Devices)	ARM Cortex-M4 with FPU	Up to 100 MHz (MAX32655)	Up to 3 MB / Up to 1 MB (MAX32650)	Ultra-low power, integrated peripherals for medical sensors, Bluetooth
PIC (Microchip Technology)	Proprietary	Varies	Varies	Wide range of options for different applications
C8051F96x (Silicon Labs)	8051	25 MHz	Up to 128 KB / 8 KB	Ultra-low power, integrated LCD controller, AES encryption
Renesas RA Family	ARM Cortex-M (M23, M33, M4, M85)	Varies	Varies	Strong security, high performance, ultra-low power
UT32M0R500 (Frontgrade)	ARM Cortex-M0+	50 MHz	128 KB / 16 KB	Radiation tolerant, analog signal chain, CAN controllers

2.1.3 Machine Learning for Predictive Health Analysis

- Machine learning in neonatal vital signs analysis

  Machine learning ~~can scan through available data and identify~~ early indicators of ~~worsening health~~. By analyzing continuous streams of sensor data from physiologic sensors, machine learning algorithms can identify subtle patterns and deviations that can occur before overt changes in the status of an infant are seen, allowing them to intervene early [17].

  Linear Regression, Decision Tree, Naive Bayes, Random Forest, and Deep Neural Networks are some of the algorithms explored for incubator control and performance estimation. Such algorithms can be used to predict the complex relationships among environmental factors, infant vital signs, and infant health

status. For instance, machine learning models can predict infant incubator status of function according to post-market surveillance measurement so that defective equipment can be detected. They can also control the incubator environment by tracking real-time ~~temperature, humidity, heart rate, and oxygen levels~~ data and dynamically adjust the settings to deliver individualized and optimal environments for every infant [17, 26]. ~~Machine learning can predict infant mortality risk~~ in incubators.

Research has explored the use of various machine learning algorithms, such as XG Boost and ~~Random Forest~~, to predict infant mortality risk in incubators based on variables determined by medical professionals. These predictive models can assist medical professionals in making early diagnoses and interventions, which can possibly improve the survival chances of neonatal infants. Smote and ADAYSN are most commonly employed as data imbalancing methods to enhance these forecast models. ~~LSTM~~ networks have been utilized for time-series incubator data analysis. ~~Long Short-Term Memory~~ (LSTM) networks, a type of recurrent neural network, are well fit for time-series analysis, e.g., the continuous stream of vital signs being monitored for infants in incubators.

These networks are able to learn temporal relationships within the data and can be used to predict future values or identify unusual patterns over a time interval. Experiments have shown that LSTM models are able to achieve high prediction accuracy of parameters in an infant incubator [27].

- Potential for personalized care and reduced oversight

Machine learning is able to dynamically regulate incubator parameters based on the individual infant parameters of weight and gestational age. By analyzing the data relevant to the individual infant's characteristics, such as health status, birth weight, and gestational age, machine learning algorithms can ~~dynamically~~ and automatically control the environmental conditions in the incubator to provide the most appropriate and personalized care to the individual infant [17]. This approach moves ~~from a one-hat policy~~ toward neonatal care, tailoring the environment to meet the ~~particular~~ needs of the individual infant.

Prediction analytics ~~can help health care professionals to intervene early, which can potentially reduce the need for ongoing manual monitoring.The ability of machine learning algorithms to predict potential health complications or deterioration in a baby's condition can help medical workers take early action, normally before the condition becomes critical. Early action has better health benefits and can also reduce the workload of the doctor since it allows him/her to focus all his/hers attentions on weaker babies, rather than having to constantly watch all of them.~~

2.1.4 Medical Device Design for Neonatal Care Systems

- Literature review on infant incubator design

Smart incubator systems are built to provide the optimal care through controlling and monitoring environmental conditions. Smart incubator systems employ advanced technology to continually monitor and automatically regulate significant environmental parameters in the incubator, such as temperature and

humidity, to provide a healthy and stable environment for preterm and critically ill newborns. Some monitor the body temperature of the baby [20].

They typically incorporate features like remote monitoring, alarm, and data transfer. Internet of things (IoT) is a common feature in most smart incubators, which supports remote monitoring of the incubator setup as well as the health status of the infant from smartphones or computers. The devices can also send alerts to medical staff and parents in the event of any variation in temperature, humidity, or other parameters being monitored. Medical information can be transmitted in real time to the medical staff and parents through IoT apps, and a few systems have been designed to store medical data via a central network [20].

Low-cost ~~double~~-powered ~~wise~~ incubators ~~have also been proposed for poor countries. Keeping in mind the financial constraints of healthcare facilities in poor countries that may not be able to purchase regular neonatal incubators, researchers have focused on developing low-cost and efficient alternatives.~~ Dual-power facility is envisaged in some designs to ensure dependability even when there is unstable power supply. The vision is to design better-performing incubators that can be produced more for distribution to health centers in resource-poor settings [5, 20, 21, 28].

Other designs are targeting the replication of the womb of the mother environment. ~~Simulating the mother's womb environment is one of the primary objectives of infant incubator design, particularly smart incubators. It consists of providing a controlled temperature, adequate humidity, and in some designs, a controlled oxygen concentration, all being crucial for healthy premature baby development~~ [12, 17, 20].

- Safety standards and regulatory requirements in Sri Lanka

  Medical devices in Sri Lanka are regulated by the ~~NMRA. The~~ NMRA, established in 2015, is responsible for ensuring that medical devices marketed ~~in Sri Lanka are of~~ acceptable ~~quality,~~ safety, and ~~fitness for their intended use~~. This includes regulating the registration, licensing, importation, sale, and distribution of medical devices [29–31].

  ~~Registration is a multi-step process that entails sample import license, device registration, and import license. The importation and registration of medical devices into Sri Lanka are achieved through the grant of a sample import license, device registration, NMRA approval, and finally an import license. The foreign manufacturers are first required to obtain approval of their manufacturing facilities from the NMRA.~~ Registration is a minimum of ~~6~~ months and ~~is usually accompanied by the filing of a number of documents, including business registration certificates, letters of authorization, free sale certificates, product samples, labels, user manuals, and technical specifications.~~ ~~L~~ocal testing may be required for certain types of devices [29, 32, 33].

  ~~Foreign manufacturers need to appoint a local agent (Marketing Authorization Holder). All foreign medical device manufacturers are required to submit their application for registration through a Marketing Authorization Holder (MAH), who acts as their local agent in Sri Lanka.~~ The MAH is responsible for the

registration, licensing, importation, sale, distribution, ~~handling of~~ quality failures, and all ~~other aspects pertaining to the specific medical device in Sri Lanka~~ [30, 32].

ISO 13485 compliance and CE accreditation is compulsory. ISO 13485 certification by a recognized notified body for the design, development, manufacturing, and post-marketing monitoring of medical devices is compulsory for Sri Lankan registration. Further, CE accreditation by a recognized notified body and also an EC design examination certificate if required might be necessary [29].

Sri Lankan neonatal equipment safety standards place emphasis on proper preparation, infection control, and adherence to guidelines. ~~Sri Lankan~~ neonatal care guidelines place more emphasis on antenatal counseling, proper preparation of NICU and delivery room, prevention of hypothermia, and adherence to infection control guidelines. ~~The Sri Lankan~~ guidelines ~~of neonatal care establish~~ hand hygiene practices, use of personal protective equipment, sharps management, and blood and body fluid handling [34–36].

International standards like IEC 60601-2-19 are also ~~used. The IEC 60601-2-19 standard~~ specifies basic safety and essential performance requirements for the testing of infant incubators and is an international preferred practice. While voluntary, most manufacturers recognize the importance of adhering to such international standards to ensure the effectiveness and safety of their medical devices. Neonatal transport guidelines also refer to European Standards EN 13976-1 and EN 13976-2 for neonatal transport equipment [37].

Chapter 3

Methodology

3.1 Overall System Architecture

At the high level view, our system is a pipeline that would start with the incubator and the infant, go through a smart edge device that does sensing and machine learning, send the results to the cloud, and then expose information and alerts to clinicians and parents. We're not going to wrap around existing incubators, but rather provide a layer of sensing, analysis and connectivity that is feasible in a constrained NICU.

On the input side, we can see the infant and the incubator: camera on baby face and skin, camera on incubator LCD, and microphone on baby cries. In the middle, a Raspberry Pi 4B+ is executing multiple microservices reading the signals, applying the machine learning (for jaundice and cry), parsing the vitals out of the LCD, and computing NTE-based recommendations. On the output side, those signals are being sent via MQTT to a ThingsBoard Community Edition instance on Google Cloud Platform (GCP), and from there to Cloud Run backends and a React dashboard, and finally to a Flutter mobile application used by clinicians and parents.

Figure 3.1 shows this simple inputs–processing–outputs view before the more detailed architecture is discussed.

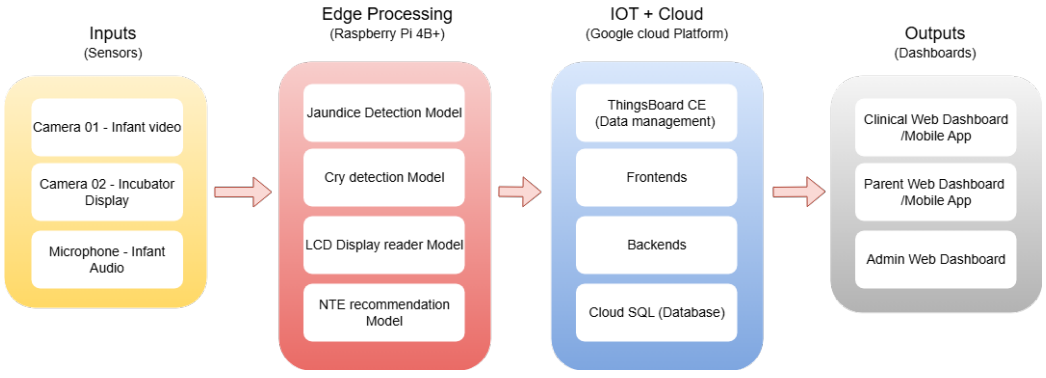

Figure 3.1: High-level input–processing–output pipeline of the incubator monitoring system.

In the remainder of this section we describe the same architecture in greater detail. In subsection 3.1.1 we outline the skeleton structure of the edge and cloud components,

including the use of Tailscale and Nginx as a sort of secure bridge, while subsection 3.1.2 shows how data actually flows through the system, from frames and audio all the way to dashboards and notifications.

3.1.1 Edge–Cloud Architecture with Raspberry Pi 4B+

The detailed architecture is realized in a number of repositories, each hosting one subsystem. All the Raspberry Pi services (camera streaming, LCD OCR, jaundice and cry detection, NTE recommendations — a very small snapshot UI) are in a separate edge-device repository for Pi microservices [38]. The incubator display OCR models, as well as the training code, are in a Neonatal Incubator Display Reader repository [39]. The jaundice model is in a Neonatal Jaundice Detection repository [40]. The rule-based NTE logic is in its own NTE recommendation engine repository [41]. The logic used for the cry classification models and the feature-extraction logic is based on the Cry-Detection-Classification-Model project [42]. On the cloud side, the React dashboard is in a repository, as are the Node.js admin backend and parent/clinician backend (in separate repositories [43]). The self-hosted ThingsBoard CE configuration is stored in a separate ThingsBoard repository [44]. A Flutter NICU mobil app [45] implements mobile access for parents and clinicians; the code in the repository is behind the production version, but the architecture is the same.

On the edge, there is a Raspberry Pi 4B+ with 4 GB RAM located next to the incubator and connected to two USB cameras and a microphone. The “baby” camera is pointed at the face and upper body of the infant and used for both jaundice and live video streaming. The “LCD” camera is pointed at the incubator LCD display so that the heart rate, SpO_2, skin temperature, humidity, and air temperature can be measured non-invasively from the screen. Both are streamed to a Python application that makes them available via `mjpg-streamer`, a simple MJPEG streaming server [46] and then consumed by Python microservices in the Pi edge repository [38]. The LCD reading service loads a YOLOv8 detector and EasyOCR model from the OCR repository [39] and exposes a simple HTTP endpoint `/readings` on port 9001 that returns the latest validated vitals as JSON. The jaundice service wraps the MobileNetV3-Small model exported to ONNX from the jaundice repository [40], gates on brightness and baby presence and exposes an HTTP endpoint on port 8887 for on-demand classification. The cry service uses the feature-extraction and ensemble models from the cry project [42] and listens on port 8888. The NTE server calls into the FastAPI-based NTE recommendation engine [41] and provides a local API that takes an infant age, weight, and current readings and returns recommended incubator temperature ranges and advisory messages.

Once this edge-side processing is done, telemetry is packaged and sent over MQTT to a ThingsBoard Community Edition instance running on a GCP virtual machine. The ThingsBoard instance is configured using dashboards, rule chains and device profiles maintained in the ThingsBoard configuration repository [44]. The MQTT messages carry structured fields such as `heart_rate`, `spo2`, `skin_temp`, `air_temp`, `humidity`, `jaundice_status`, `cry_status`, and `nte_advice`, and are associated with a particular device corresponding to the incubator. ThingsBoard stores these data and offers

REST and WebSocket APIs for them, which are used later by the web dashboard and the backends.

The application layer in GCP is built from the integration repository [43]. The React dashboard (under `react_dashboard/`) is deployed as a Cloud Run service and provides three role-based interfaces: a clinical dashboard for doctors and nurses, a parent dashboard, and an admin panel. The admin backend and the parent/clinician backend are two Node.js microservices also deployed on Cloud Run. They provide REST APIs for login, JWT-based authentication, admin user management, baby and parent management, invitation and PIN workflows, and messaging. Both backends connect to a Cloud SQL PostgreSQL instance (called `incubator-db`) and use separate logical databases (`admin_db`, `parent_db`, and `incubator_system`) to store admin accounts, parent accounts, invitations, message threads, and any additional metadata required. Detailed deployment instructions, including Dockerfiles, GCP `cloudbuild.yaml` files, and environment variable templates, are described in the documentation files in this repository.

~~This is somewhat awkward because Cloud Run exposes services only on a private Tailscale network, and is not publicly routable. Our chosen solution (following the Tailscale documentation[47]) is to use a small GCP virtual machine as a Tailscale subnet router, and also running Nginx as a reverse proxy.~~ The router VM is configured to join the same Tailscale network as the Raspberry Pi and is allowed to route traffic to the Pi's `100.89.x.x` address. Nginx is configured (in `nginx-pi-proxy.conf` and `tailscale-nginx.conf` at the root of the integration repo) so that when a HTTPS request comes in to the appropriate path on the React dashboard or backends (e.g. `/api/pi/camera/`, `/api/pi/lcd/`, `/api/pi/jaundice/`, or `/api/pi/snapshot/`), it is forwarded over the Tailscale VPN to the appropriate service running on the Pi. In the other direction, the Pi itself only needs outbound connectivity to the Tailscale control plane, and to the ThingsBoard and GCP endpoints; it never needs to open inbound ports to the public internet. As such, live video streaming and all low-level APIs are effectively hidden behind the Cloud Run URL and the Nginx reverse proxy, but are still encrypted end-to-end by Tailscale.

Figure 3.2 shows this detailed edge–cloud structure, including the Raspberry Pi, the Tailscale router and Nginx proxy, the ThingsBoard CE instance, Cloud Run services, Cloud SQL, and the main classes of clients. For clarity, internal port numbers are omitted in the diagram and only the logical roles are shown.

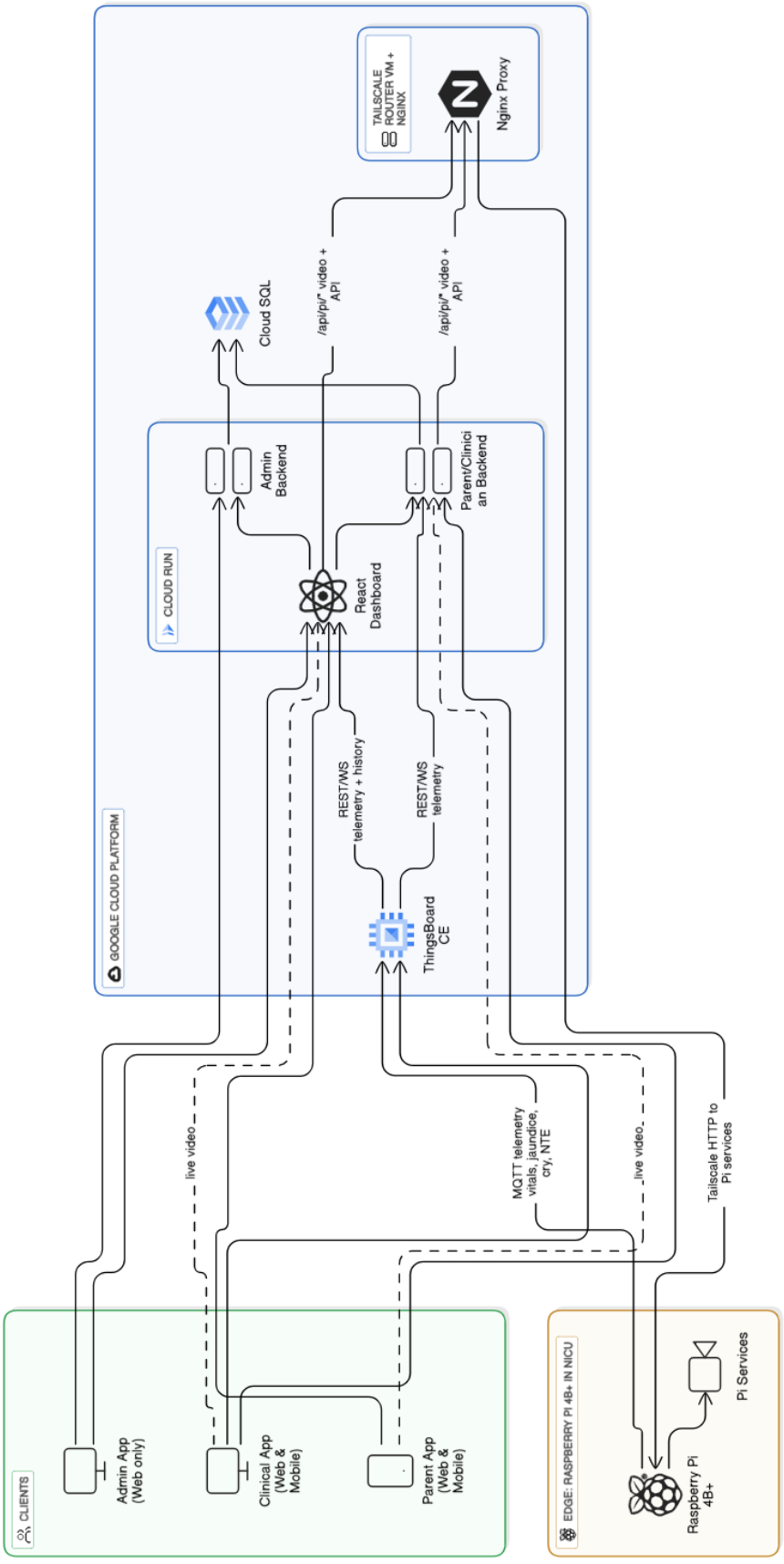

Figure 3.2: Edge–cloud architecture with Raspberry Pi 4B+, ThingsBoard CE, GCP services, and client applications.

3.1.2 End-to-End Data Flow

Beyond the static structure, it is useful to understand how data actually travels through the system during typical operation. The flow always begins at the incubator and the infant, where the Raspberry Pi 4B+ is continuously collecting data from the two cameras and the microphone. The infant-facing camera produces frames that are fed into the jaundice detection pipeline from the Neonatal Jaundice Detection repository [40]. Before classification, each frame is analysed by a brightness gate and a baby-presence detector, both implemented in the same repository, to ensure that only clear, well-lit images with a visible baby are passed to the MobileNetV3-Small model. The LCD-facing camera delivers frames into the Neonatal Incubator Display Reader pipeline [39], where YOLOv8 localises each segment of the display and EasyOCR is used to convert the digits into numeric vitals, followed by range checks based on clinical guidelines [48]. Audio from the microphone is segmented into one-second windows and fed into the cry detection pipeline built on top of the Cry-Detection-Classification-Model [42]. In parallel, the NTE service periodically calls the NTE recommendation engine [41] with the current incubator air temperature, the infant's age in hours, and weight in grams, and receives back an NTE temperature band and a short textual recommendation.

All of these outputs are combined into telemetry messages that are published via MQTT to the ThingsBoard CE instance. ThingsBoard stores the incoming time series and triggers rule chains when certain thresholds are crossed, for example when vitals leave their normal ranges or when the NTE engine reports that the incubator air temperature is too low. The React dashboard running on Cloud Run periodically queries ThingsBoard's REST API for the latest telemetry and chart data, and the parent/clinician backend may also subscribe to telemetry updates to generate notifications for the mobile app. When a clinician opens the clinical dashboard, the React front-end uses the parent/clinician backend to discover the list of babies and incubators from the Cloud SQL database, then uses the ThingsBoard API to fetch detailed vitals and history for the selected device. The user can then click on a camera icon in the dashboard, which causes the browser to request live video from `/api/pi/camera/`. This request is routed through the Tailscale router VM and Nginx proxy, which forwards it over the VPN to the Pi's infant camera stream. The resulting MJPEG stream is returned to the browser and displayed in a video component on the clinical or parent dashboard. For parents using the mobile app, an analogous flow occurs: the mobile app uses the backend to obtain a proxy URL to the camera stream, which again goes through Nginx and Tailscale but appears to the app as a single HTTPS resource.

In summary, the end-to-end data flow ties together the edge processing on the Raspberry Pi, the MQTT and REST interfaces of ThingsBoard CE, the Cloud Run backends and React dashboard, the Cloud SQL database, and the Flutter mobile app into a coherent pipeline. Figure 3.3 sketches the major steps in sequence form. The Mermaid source is included for reproducibility and to make it easy to adjust the diagram as the system evolves.

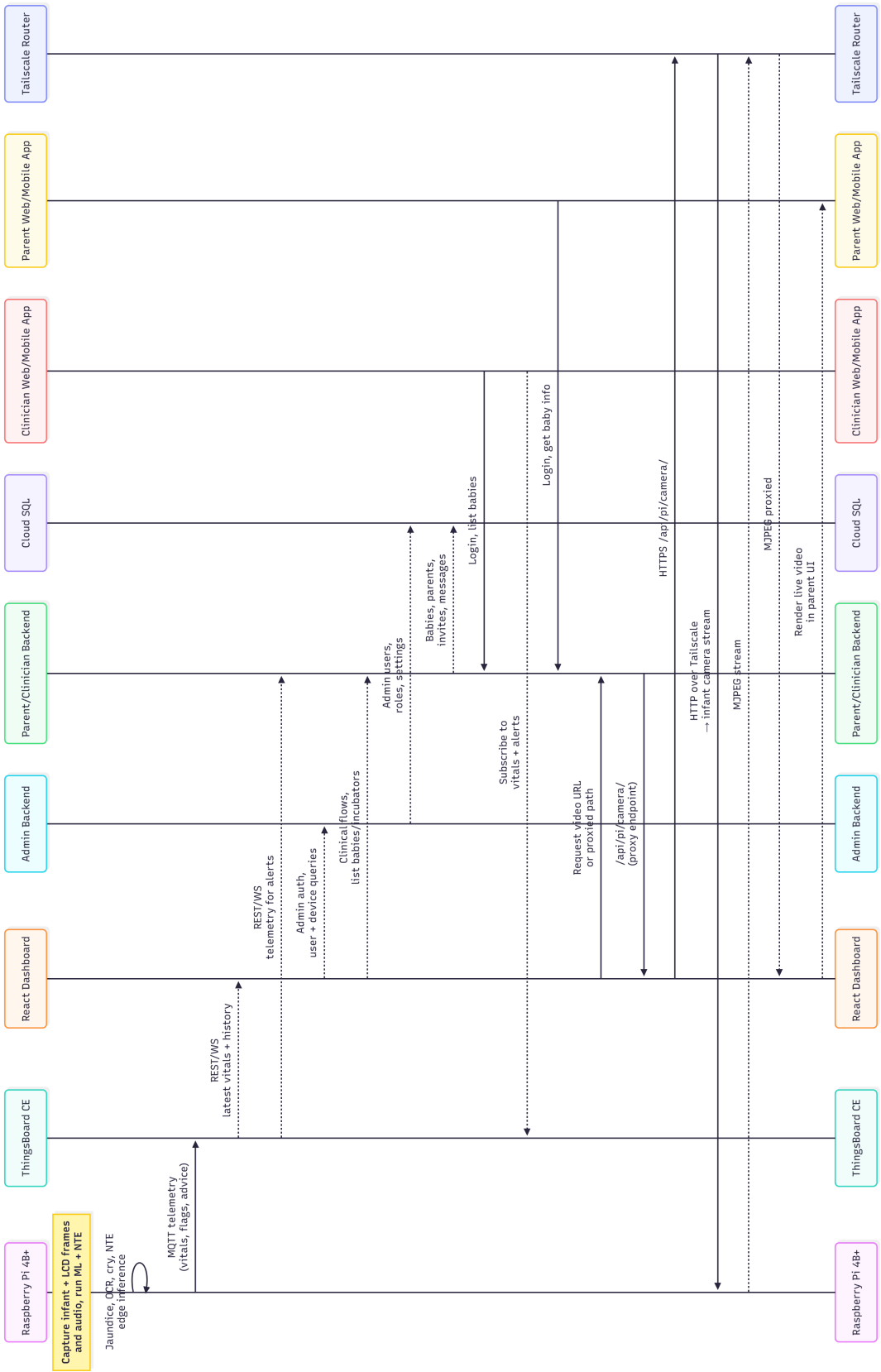

Figure 3.3: End-to-end data flow from edge inference to dashboards and mobile apps.

Note: The concrete code and configuration for these flows are documented across the edge-device repository [38], the display reader [39], jaundice [40], NTE [41], cry detection [42], the ThingsBoard configuration repo [44], and the Cloud integration repository [43], as well as the official documentation for Raspberry Pi 4 Model B [49], ThingsBoard [50], Tailscale [47], and the tools used for OCR and streaming [46, 51].

3.2 Machine Learning Models and Integration

The proposed system employs three principal machine learning components: (i) a convolutional model for neonatal jaundice detection, (ii) an audio model for infant cry detection and classification, and (iii) a computer-vision model for incubator LCD reading. In all three cases the development process followed a similar pattern. Models were first designed and trained in dedicated repositories using offline data and interactive notebooks, then exported as light-weight artefacts (e.g. ONNX, PyTorch, or scikit-learn pickles), and finally deployed on the Raspberry Pi 4B+ as microservices. These microservices expose narrow HTTP APIs and/or publish telemetry to the IoT platform, which simplifies integration and isolation. Figure 3.4 illustrates this common workflow at a high level.

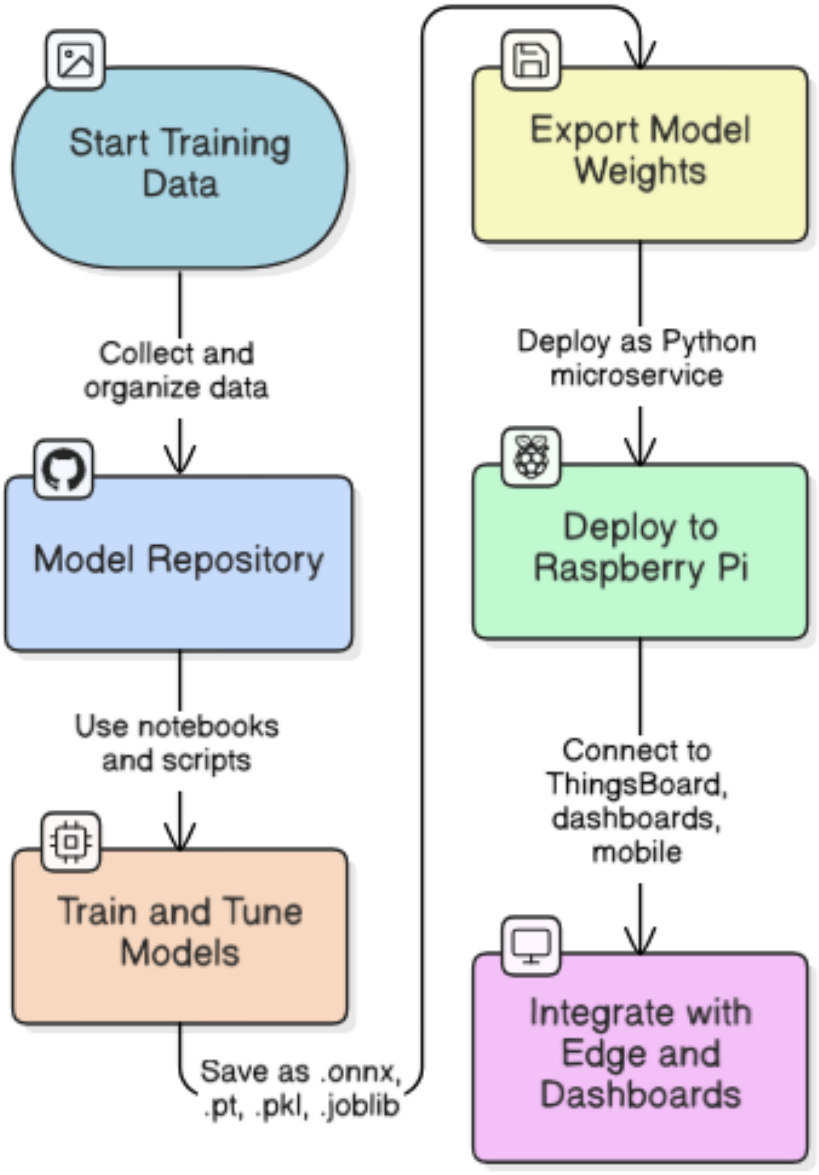

Figure 3.4: Common development process for machine learning components: offline training in a dedicated repository, export of trained weights, and deployment as a microservice on the Raspberry Pi 4B+.

The subsequent subsections describe each model in turn. For each, the model architecture and training procedure are presented first, followed by a description of how the trained model is integrated into the edge device and the wider system.

3.2.1 Model 1– Neonatal Jaundice Detection

The neonatal jaundice detector is implemented in the `Neonatal_jaundice_detection` repository [40]. The design objective was to obtain a classifier that is computationally feasible on a Raspberry Pi 4B+, yet sufficiently expressive to capture colour and luminance patterns associated with jaundice. After empirical comparisons with alternative backbones, MobileNetV3-Small [52] was selected as the feature extractor

because of its favourable accuracy/efficiency trade-off and its mature implementations in PyTorch.

Model architecture and training procedure. Training is conducted in a Jupyter notebook (`jaundice-detection.ipynb`) within the jaundice repository [40]. The model is fine-tuned on the Kaggle Jaundice Image Data set [53], which contains colour images labelled as jaundiced or normal. All images are resized to 224×224 pixels, converted to RGB, and subjected to data augmentation including small rotations, horizontal flips, and brightness/contrast perturbations. This augmentation scheme aims to approximate the variability of camera positioning and illumination around incubators. The dataset is divided into training and validation subsets in an 85/15 proportion with a fixed random seed for reproducibility.

On the modelling side, the `mobilenet_v3_small` network from `torchvision` is initialised with ImageNet weights and its classification head is replaced by a single linear unit yielding one logit for binary classification. The network is trained using the `BCEWithLogitsLoss` function, AdamW optimiser, and a `ReduceLROnPlateau` scheduler acting on the validation loss. Training is run for approximately ten epochs; in practice, the validation accuracy, sensitivity, and specificity stabilised within this range. At the end of training, the best weights are exported both as a PyTorch checkpoint (`.pt`) and as an ONNX model (`.onnx`) to support different deployment targets.

Figure 3.5 summarises this training pipeline.

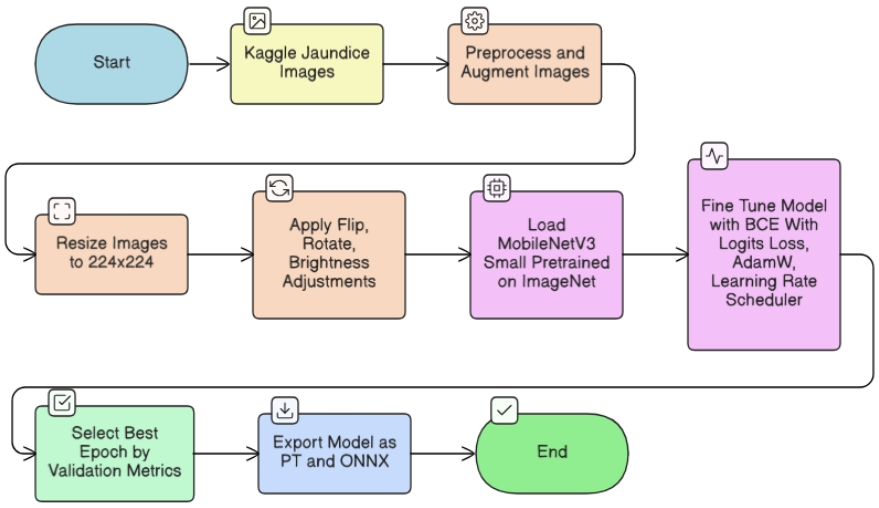

Figure 3.5: Neonatal jaundice model training: MobileNetV3-Small fine-tuned on the Kaggle jaundice dataset and exported for deployment.

Edge deployment and system integration. On the Raspberry Pi, the ONNX variant of the model is loaded via ONNX Runtime (`app.py` in [40]) and wrapped as a microservice in the Pi services repository [38]. Before invoking the CNN, two gating mechanisms are applied to each frame from the infant-facing camera. The first is

a brightness gate that computes the mean grayscale intensity of the frame; if this value falls below a defined threshold, the system returns a "too dark" status and does not proceed to classification. For moderately dim frames, contrast-limited adaptive histogram equalisation (CLAHE) is applied to improve visibility and the result is marked as lower reliability.

The second gate checks for the presence of a baby in the frame. This is implemented in `baby_presence_detector.py` using a combination of a Haar cascade face detector and a YCrCb-based skin mask. The detector measures face area as a fraction of the frame and the ratio of skin pixels to total pixels, and only if both exceed configured thresholds is the frame accepted as containing sufficient infant skin. If no acceptable face/skin combination is found, the system returns a "no baby detected" status and skips inference.

For frames that pass both gates, the image is resized to 224×224, normalised with ImageNet statistics, and forwarded to the MobileNetV3-Small ONNX model. The logit is converted to a probability via a sigmoid, thresholded to derive a binary label, and returned along with the probability and a reliability flag. The microservice publishes these outputs to ThingsBoard CE [50] as telemetry fields (e.g. `jaundice_flag`, `jaundice_prob`), which are then consumed by the React dashboard [43]. In the clinical dashboard, each incubator row includes a jaundice indicator that reflects the most recent classification.

The edge inference pipeline for jaundice detection is depicted in Figure 3.6.

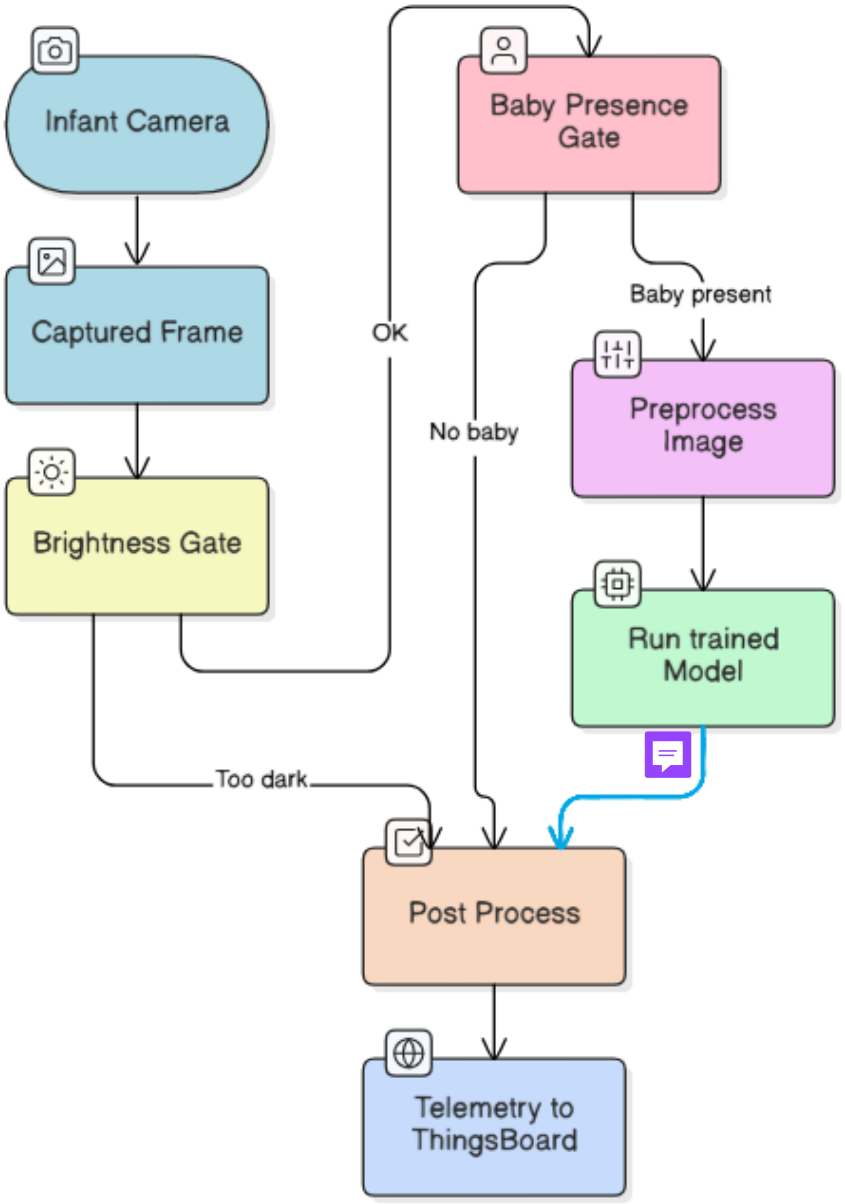

Figure 3.6: Jaundice inference pipeline on the Raspberry Pi: image gating followed by MobileNetV3-Small inference and telemetry publication.

3.2.2 Model 2– Infant Cry Detection and Classification

The cry detection and classification subsystem is based on a combination of signal processing and supervised learning. Model training and evaluation are carried out in the `Cry-Detection-Classification-Model` repository [42], while the real-time components on the Raspberry Pi consist of two services: `cry_detector.py`, which performs continuous detection, and `cry_classification_service.py`, which classifies short cry segments.

Model architectures and training procedure. Two related tasks are addressed. The first is binary detection: deciding whether a given audio segment contains a cry. The second is multi-class classification: given a segment identified as a cry, predicting its type (e.g. hunger-related, discomfort, etc.). For detection, the trained pipeline uses YAMNet [54] as a fixed embedding extractor followed by a Logistic Regression model. For classification, a feature-engineering approach based on `librosa` [55] is combined with an ensemble classifier.

For detection, audio clips are resampled to 16 kHz and passed through the YAMNet model, which yields a sequence of 1024-dimensional embeddings. These embeddings are aggregated across time by computing their mean and standard deviation, producing a 2048-dimensional feature vector. A standard scaler and PCA transformation are applied, and the resulting features are used to train a Logistic Regression classifier to discriminate cry versus non-cry. The detection threshold (denoted `DETECTION_THRESHOLD` in the code) is chosen empirically on the validation set to balance sensitivity and false positives.

For classification, the pipeline computes classical audio descriptors using `librosa` [55]: MFCCs, chroma features, Mel-spectrogram energies, spectral contrast, tonnetz, and other spectral shape statistics. These features are averaged over time to form a compact vector. Several base classifiers, including ~~Random Forests and Gradient Boosting~~ machines, are evaluated; the final classifier is a voting ensemble that combines the most promising models, accompanied by a scaler, feature selector, and label encoder. These objects are persisted using `joblib` as files such as `yamnet_lr_model.joblib`, `scaler_yamnet.pkl`, `babycry_ensemble.pkl` and others [42].

Figure 3.7 summarises the training pipeline.

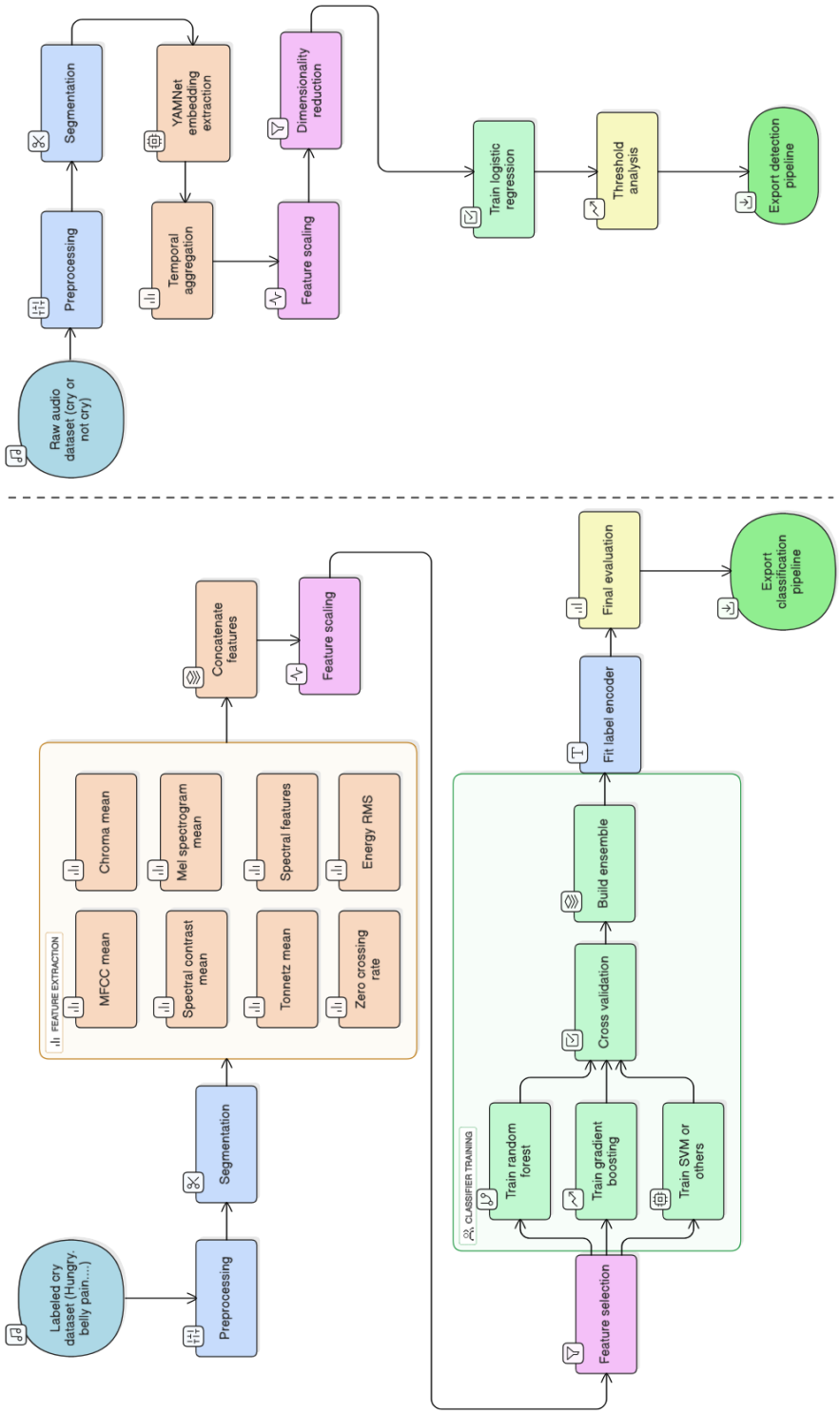

Figure 3.7: Cry model training: YAMNet embeddings with Logistic Regression for detection, and an ensemble classifier with librosa features for cry type.

Edge deployment and system integration. On the Raspberry Pi, continuous detection is handled by `cry_detector.py` in the Pi services repository [38]. Audio is captured from a USB microphone at 48 kHz using `PyAudio`. The detector maintains a sliding buffer of approximately two seconds and, for each buffer, computes the root-mean-square (RMS) amplitude and a short-time Fourier transform. It then

estimates the proportion of spectral energy lying in a "cry band" between 300 Hz and 2000 Hz. If both the RMS amplitude and the energy ratio in this band exceed thresholds (which are tuned via `sensitivity`, `cry_threshold` and `noise_threshold`), the buffer is marked as indicating a cry event.

Upon detecting the onset of a cry (i.e. a transition from "no cry" to "cry detected") and provided that no recording is in progress, the detector begins to store raw audio samples into a recording buffer. Recording continues until roughly five seconds of audio have been accumulated, as specified by the constant `RECORD_DURATION = 5`. This five-second segment is then resampled to 16 kHz and passed to the classification service `cry_classification_service.py` running locally on the same Raspberry Pi (port 8890). The classifier service loads the YAMNet and ensemble models exported from the cry repository [42], extracts YAMNet embeddings and librosa features, and produces a JSON response containing a binary decision (`is_cry`), a detection confidence, a cry type label (if the classification confidence exceeds a threshold), and the probability distribution over cry types.

Back in `cry_detector.py`, this information is used to update the cry state, which records whether a cry is currently detected, the last cry time, total detections, and any cry type information. A cooldown interval, controlled by `classification_cooldown` (10 s in the current configuration), prevents repeated classification of overlapping segments during prolonged episodes.

The detector also exposes its state via a local HTTP API (`/cry/status`, `/cry/start`, `/cry/stop`) and publishes a summary to ThingsBoard CE [50] via MQTT using the `ThingsBoardClient` helper. The telemetry includes fields such as `cry_detected`, `cry_audio_level`, `cry_sensitivity`, and `cry_total_detections`. The React dashboard [43] displays this information as an indicator on each incubator card and can plot cry events over time. The clinical dashboard web/mobile app [45] consumes the same telemetry on the clinician side and can raise notifications when repeated cry episodes are observed.

The deployment pipeline for the cry subsystem is summarised in Figure 3.8.

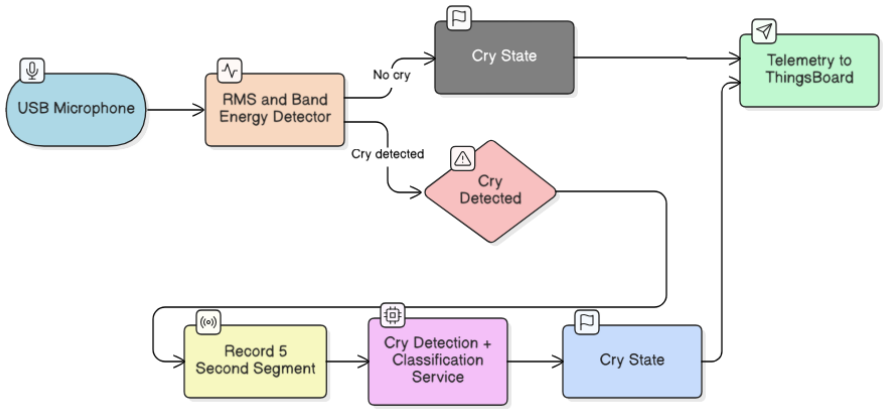

Figure 3.8: Cry subsystem on the Raspberry Pi: a real-time detector triggers five-second recordings that are then classified locally, with the resulting state exported as telemetry.

3.2.3 Model 3– Incubator LCD Display Reader

The incubator LCD reader allows the system to reuse the incubator's existing display rather than installing additional wired sensors. The majority of the development for this component is done in the `Neonatal_incubator_displayReader` repository [39], with the production service `lcd_reading_server.py` located in the Pi services repository [38]. An auxiliary repository [56] provides an HTML-based display simulator that was used to generate additional test images and to reproduce specific display layouts during development.

Model and training procedure. The LCD reader has two main tasks: detection of the display regions containing numeric values, and recognition of the digits within those regions. For the first task, a YOLOv8n detector [57] is trained on images of incubator displays annotated with bounding boxes around each parameter region (heart rate, SpO_2, skin temperature, humidity, and air temperature). The annotations are converted into YOLO format and organised into training, validation and test splits; the class mapping and split configuration are recorded in `artifacts/yolo/splits/incubator.yml`. Training is carried out with the standard Ultralytics implementation, using an input resolution of 640, default augmentation settings, and monitoring mean average precision on the held-out test data. The resulting weights (e.g. `incubator_yolov8n_v4.pt`) are exported and later loaded on the Raspberry Pi.

For OCR, two approaches were evaluated. The initial pipeline relied on Tesseract OCR [58], with a "fast" mode tuned for real-time use and a more exhaustive "accurate" mode intended for batch processing. The logic in the display-reader repository [39] includes ROI caching, frame skipping and resolution scaling to make Tesseract feasible. However, in on-device tests with real incubator displays, Tesseract exhibited sensitivity to glare and low contrast. A second pipeline using EasyOCR [51] was therefore implemented. Although EasyOCR is itself a learned model, its pre-trained

weights handled the incubator digit patterns and lighting variations more robustly in our experiments. Based on systematic tests using the display simulator [56] and NICU photographs, EasyOCR was chosen as the default OCR backend for the edge device, while Tesseract remains available for offline or alternative workflows.

Figure 3.9 provides a compact overview of the LCD reader model development.

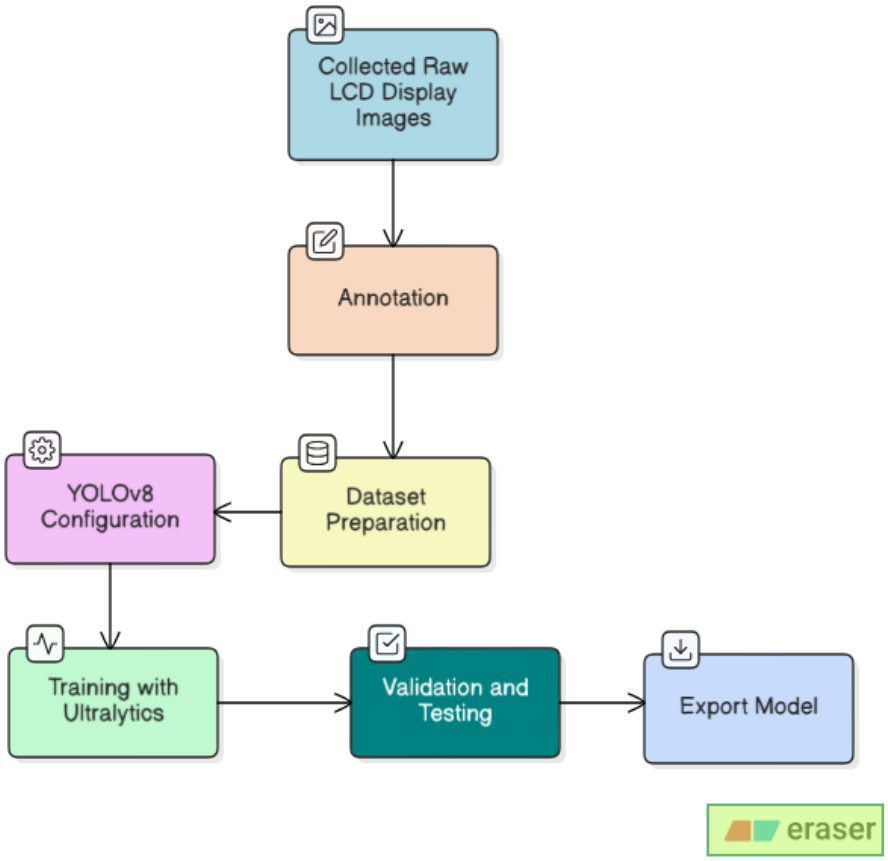

Figure 3.9: LCD reader development: YOLOv8n trained for display-region detection, combined with a pre-trained OCR engine (EasyOCR as default, Tesseract as alternative).

Edge deployment and system integration. On the Raspberry Pi, the pipeline is implemented in `lcd_reading_server.py` [38]. The LCD camera is connected to `mjpg-streamer`, which exposes a MJPEG stream on port 8081. The server attaches to this HTTP stream, reads a frame, decodes the JPEG using OpenCV, and passes the resulting image to the YOLOv8n detector to obtain bounding boxes for each parameter. For each detected box, the corresponding region is cropped and, if necessary, pre-processed (e.g. converted to grayscale or normalised). These ROIs are then OCR-processed by EasyOCR. The raw OCR outputs are post-processed by stripping non-digit characters, enforcing expected formats (for instance, fixed numbers of digits, optional decimal point positions), and converting the result into numerical values.

Each parameter is checked against a pre-defined range derived from clinical guidelines [48], such as plausible heart rate, SpO_2, skin temperature, air temperature, and humidity values. If a reading falls outside the acceptable range, it is flagged as invalid. The service maintains a cache of the last valid reading for each parameter and can either retain those values or mark a field as missing when new readings are invalid. A background loop executes this capture–detect–OCR–validate cycle at a fixed interval (e.g. every five seconds), updating the cache accordingly. The current cached readings are exposed via a `/readings` HTTP endpoint on port 9001, and may also be relayed to ThingsBoard CE as telemetry fields if desired.

From the dashboard's point of view, the LCD reader is accessed through the Nginx reverse proxy on the Tailscale router VM, via the path `/api/pi/lcd/`, which is mapped to `/readings` on the Pi [43, 47]. The React clinical view consumes this endpoint through a small helper and displays the resulting vitals in the main panel, while the NTE widget uses the air temperature reading as one of its inputs. During development, the incubator display simulator [56] allowed for repeated testing by rendering synthetic displays and feeding them through the same pipeline.

The on-device LCD reader pipeline is summarised in Figure 3.10.

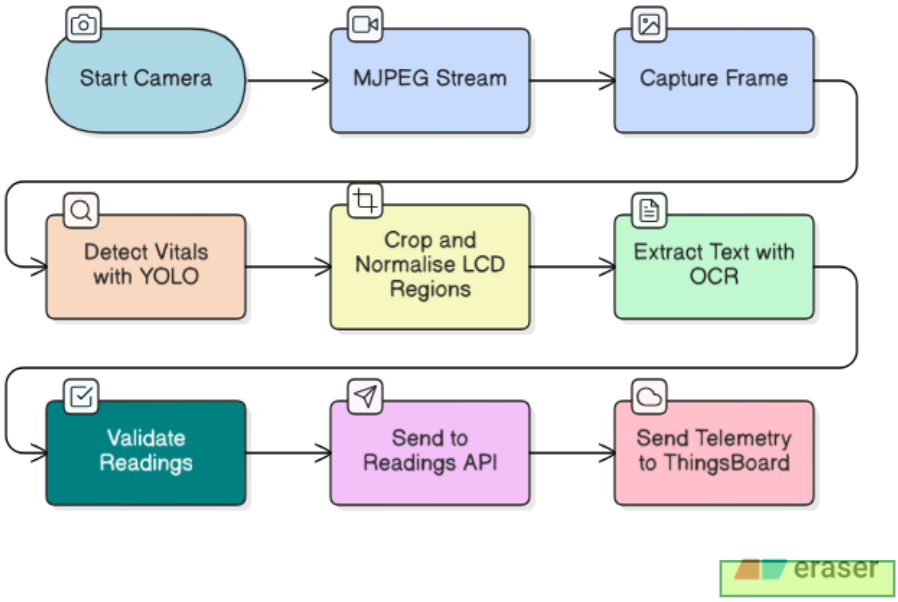

Figure 3.10: LCD reader pipeline on the Raspberry Pi: MJPEG frames from the LCD camera are processed by YOLOv8n and EasyOCR, validated and served as numerical vitals.

3.3 NTE Recommendation Engine

The Neutral Thermal Environment (NTE) recommendation engine provides a rule-based decision layer that links the raw temperature readings from the incubator to clinically motivated target ranges. Rather than predicting anything from scratch, this

component encodes the NTE table from the Sri Lankan neonatal guidelines [48] (introduced earlier in Table 1.1) into a small, well-tested rules engine and exposes it via a simple API. In that sense it is not a machine learning model, but it fits naturally into the "intelligent processing" layer of the system alongside the learned models described in Sections 3.2.1 and 3.2.2. The implementation is maintained in a dedicated repository [41] and then wrapped on the Raspberry Pi to drive temperature-related recommendations and alerts.

3.3.1 Rule Formulation and Implementation

The starting point for the NTE engine is the neutral thermal environment table in the national newborn care guidelines [48] (reproduced as Table 1.1 in Chapter 1). This table specifies, for a range of age bands (in hours or days) and weight bands (e.g. $< 1200\,\mathrm{g}$, 1200–1500 g, 1501–2500 g, $> 2500\,\mathrm{g}$), the recommended incubator air temperature range to maintain thermal neutrality. The table is quite detailed, with separate entries for the first hours after birth, daily intervals in the first weeks, and somewhat broader categories thereafter. Rather than repeatedly looking up this table by hand, the engine encodes it into a data structure and provides a function that, given age in hours and weight in grams, returns the appropriate temperature band.

In practice, this logic is implemented in `nte_rules.py` in the NTE recommendation engine repository [41]. The file begins by defining a list of rows, each row containing an `age_min_h`, `age_max_h` interval and a dictionary of weight bands to temperature ranges. A helper function, conceptually called `find_nte_range(age_hours, weight_g)`, loops over the rows, finds the one whose age interval contains the given age, and then selects the correct weight band entry based on the infant's weight. If no exact match is found (for example, for very large ages), the engine falls back to the last row in the table. Alongside the air-temperature band, the file defines a normal core body temperature interval (36.5–37.5 °C) and a target humidity range (roughly 50–60 %), consistent with the guidelines [48].

Once the appropriate NTE range has been identified, a second function `recommend()` compares the infant's current environment readings to that target. The function accepts the infant's age and weight, along with a dictionary of current measurements (air temperature, skin temperature, humidity), plus optional tolerances. It then constructs a list of recommendation objects, each containing a type (for example "temperature" or "humidity"), a severity label ("info" or "warning"), a short message (e.g. "Increase incubator air temperature toward 35.0 °C"), and a more detailed explanatory string. The air temperature is compared to the NTE band; values below the lower bound minus a tolerance lead to "increase temperature" recommendations, whereas values above the upper bound trigger "decrease temperature" messages. Similar comparisons are made for humidity and, optionally, skin temperature. The output of the function is a structured JSON-like object which can be logged, displayed, or sent to the cloud.

Figure 3.11 sketches how the clinical table from Table 1.1 is transformed into the code-level representation in the NTE engine.

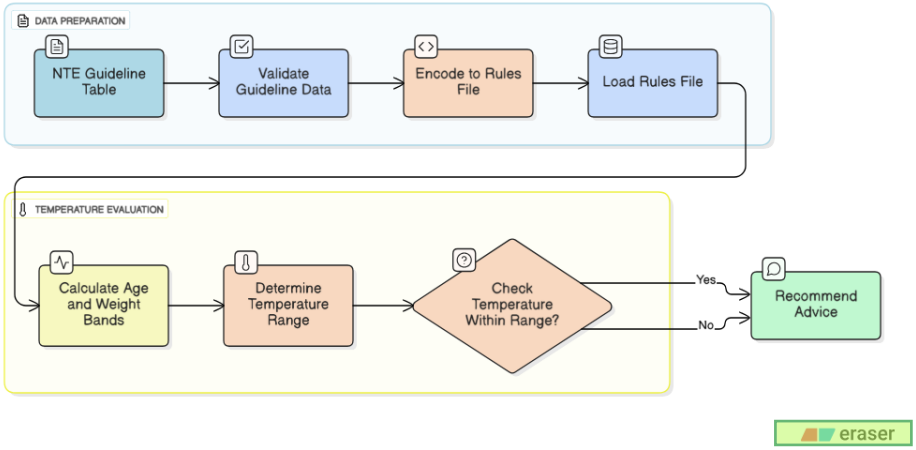

Figure 3.11: Encoding the NTE guideline table into a rule engine: age and weight bands from Table 1.1 are converted into a lookup function and a recommendation generator.

The advantage of this approach is that any future updates to Table 1.1 can be applied by modifying the data structure in one place, without changing the rest of the system. It also ensures that the recommendations are directly traceable back to the published guidelines, which is important for clinical acceptance.

3.3.2 Edge Integration and Cloud Connectivity

To make this rule engine usable in practice, a small web API is defined in `recommend_api.py` in the same repository [41]. This file uses FastAPI to expose a `POST /recommendations` endpoint. The endpoint accepts a JSON payload containing the infant's age in hours, weight in grams, and a nested structure of current readings (air temperature, skin temperature, humidity). On receipt of a request, it calls the `recommend()` function described above and returns the resulting list of advice items as JSON. Because the API is defined with standard OpenAPI semantics, it can be called both from the Raspberry Pi and from external tools, which was convenient during development and testing.

On the Raspberry Pi, this API is wrapped by the NTE microservice in the edge-device repository [38]. The NTE service periodically gathers the current environment readings from the LCD reader (air temperature, humidity) and the latest infant metadata (age in hours, weight), and sends them to the FastAPI endpoint (or directly calls the rule functions when everything is running locally). The returned advice items are then processed in two ways. Firstly, compressed versions of the recommendations (for example a simple severity flag and a summary message) are added to the MQTT telemetry that the Pi sends to ThingsBoard CE [50]. This allows the React dashboard [43] to show a concise NTE widget: a green indicator when the incubator air temperature lies within the recommended band, and amber or red messages when it is below or above the target range. Secondly, the full textual advice is made available through local HTTP APIs on the Pi for debugging or for consumption by future

edge-side user interfaces.

Figure 3.12 shows the integration flow between the edge services, the NTE engine, and the cloud layer.

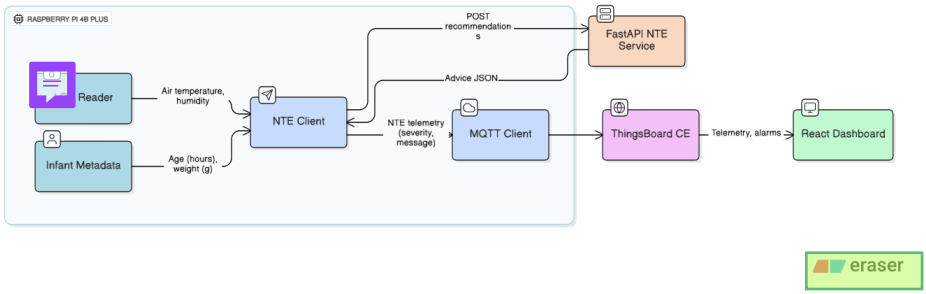

Figure 3.12: Integration of the NTE engine: readings and infant metadata from the Raspberry Pi are sent to the NTE API; the resulting advice is forwarded as telemetry to ThingsBoard CE and displayed in the dashboard.

From the user's perspective, the NTE widget appears in the clinical dashboard as a small panel showing the recommended air-temperature band for the given infant (according to Table 1.1), the current measured air temperature from the LCD reader, and a short recommendation such as "Increase incubator air temperature toward 35.0 °C" or "Within target NTE range". This information is intended to complement the raw vitals and to make it easier for staff to translate the national guidelines [48] into day-to-day incubator settings without having to consult printed tables.

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 [43], using React [59] for the front-end 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 [43]. While clinicians and parents also have corresponding Flutter mobile clients [45], 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.

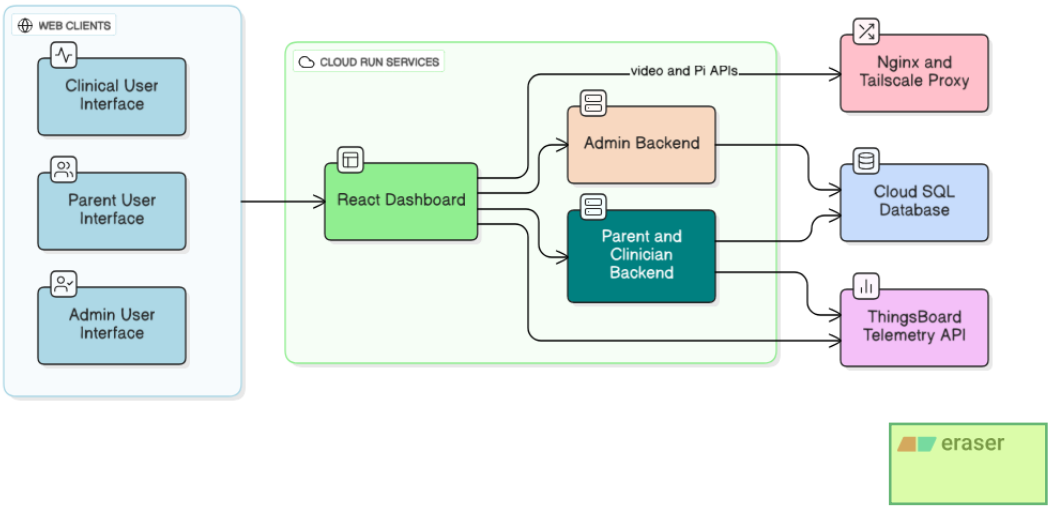

Figure 3.13: Architecture of the web dashboards: React SPA communicating with Node.js backends, ThingsBoard CE, and edge services via a Tailscale+Nginx proxy.

3.4.1 Clinical Dashboard

The clinical dashboard is designed as the primary working view for ~~doctors and nurses in the NICU~~. Its layout ~~is built around a main table of incubators~~ and ~~a detail view for each infant~~. On the left, a table lists all active incubators known to the system, using the incubator IDs and baby metadata stored in Cloud SQL [43]. Each row shows a concise summary: the incubator identifier, infant name or code, current heart rate, SpO_2, skin temperature, humidity, and air temperature as read by the LCD reader, along with small indicators for cry status and jaundice status. These vitals are not polled directly from the Pi; instead, a React service wraps the ThingsBoard REST API [50] and fetches the latest telemetry fields for each device.

When a clinician clicks on a row in the table, a detail panel for that infant is opened. This panel includes a set of "cards" for each vital, using colour coding and thresholds, and time-series charts rendered using Chart.js [60] to show trends over the recent hours. The NTE widget is embedded alongside these charts; it displays the NTE target band derived from the rule engine (Section 3.3), the current incubator air temperature, and a short textual recommendation. Beneath the vitals, the dashboard embeds the live infant camera stream, which is obtained by making an HTTPS request to `/api/pi/camera/` on the Cloud Run endpoint. This request is passed through the Tailscale+Nginx proxy (see Figure 3.2) and ultimately connects to the `mjpg-streamer` service on the Pi [46]. To the React component, however, this looks like a standard MJPEG URL and can be displayed in a regular video element.

Figure 3.14 illustrates a typical clinical dashboard screen. In practice, several design iterations were made in Figma before arriving at the final layout, but the core elements have remained the same: tabular overview, per-infant detail with charts, NTE indicator, and live video.

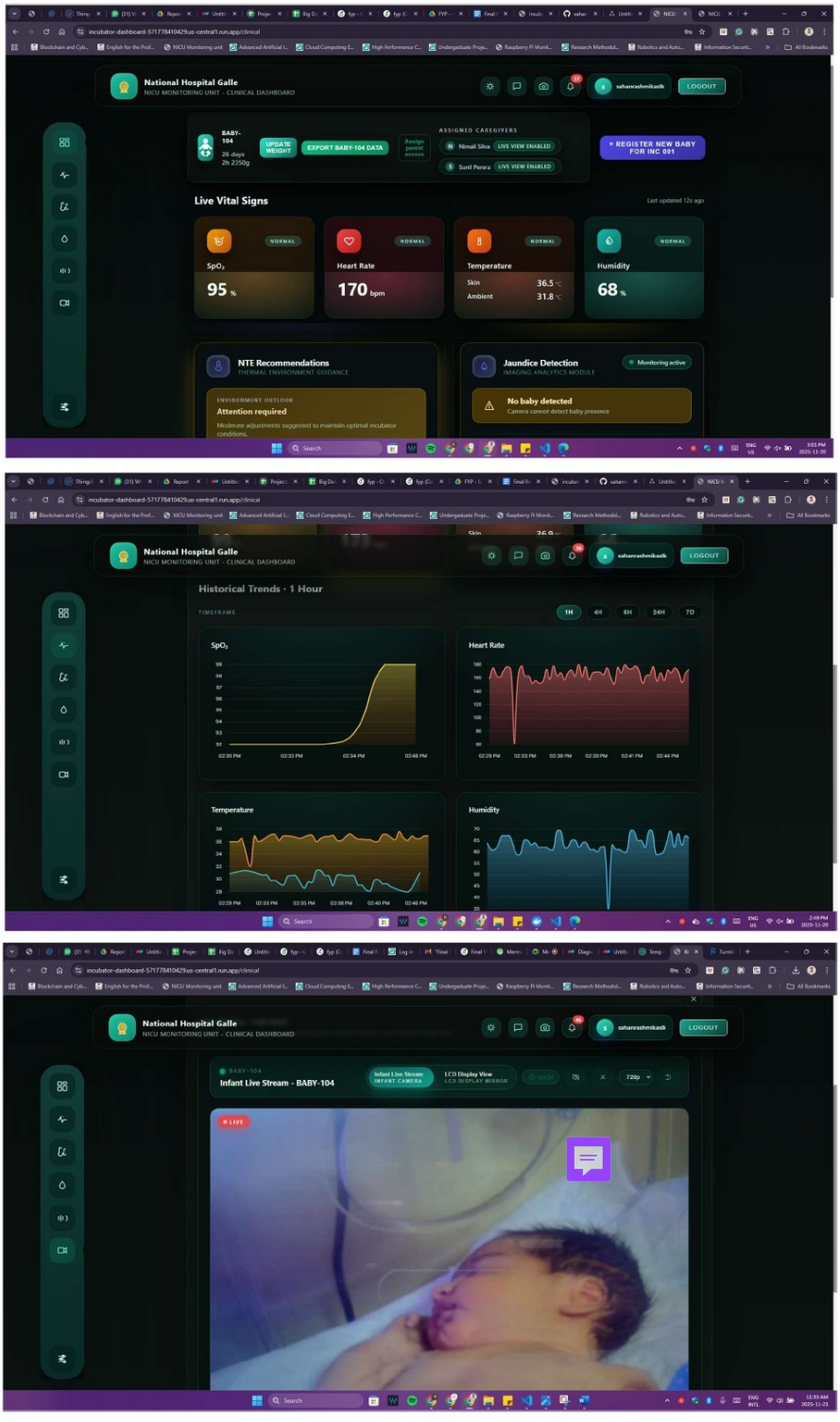

Figure 3.14: Clinical dashboard showing the list of incubators, per-infant vitals and trends, NTE widget, and live camera feed.

3.4.2 Parent Dashboard

The parent dashboard reuses the same React infrastructure but presents a simplified and more cautious view. Parents are authenticated via the parent/clinician backend [43] using invitation codes and PINs that are linked to specific infants. Once

logged in, the parent sees a much shorter list—typically only one baby or a small number if applicable. For each baby, the main element is a large live video panel showing the infant in the incubator, again obtained via the `/api/pi/camera/` endpoint and rendered as an MJPEG stream. In addition to the video, a compact status area summarises whether the infant is "stable" or "under closer observation", but numerical vitals and internal alarm states are not shown. This design choice is deliberate: it gives parents a reassuring window into the NICU without exposing them to continuous numeric fluctuations, which could be misinterpreted without clinical context.

Below the video panel, a messaging section allows for text-based communication between parents and clinical staff. Messages are handled by the parent/clinician backend which stores them in Cloud SQL and enforces appropriate access control [43]. On the clinician side, the same message threads appear in their own dashboard view and in the NICU mobile app [45], so that staff can reply from either a workstation or a mobile device. Figure 3.15 shows a schematic parent dashboard layout.

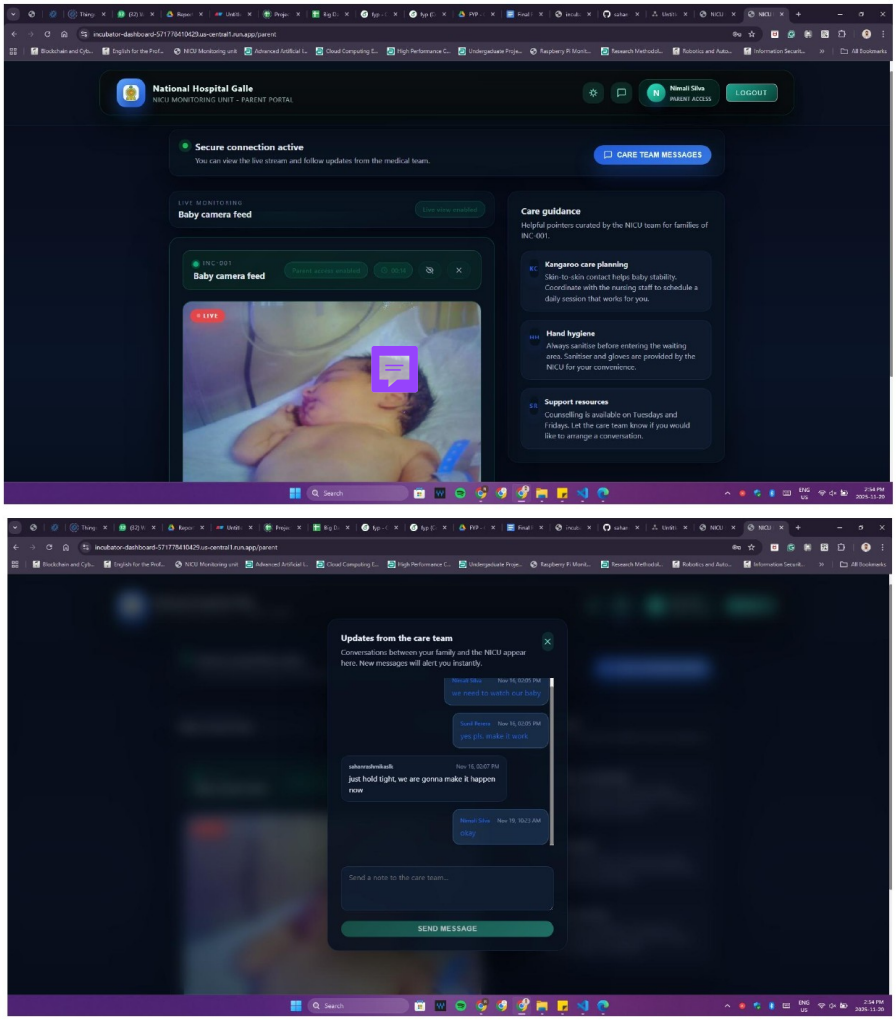

Figure 3.15: Parent dashboard with a focus on secure live video and simple status information, plus a messaging panel for text communication with staff.

3.4.3 Administrator Dashboard

The administrator dashboard is ~~used~~ for system configuration and device management rather than ~~for~~ clinical monitoring. It is implemented as a separate portal within the same React SPA [43] ~~but~~ is only accessible via the web. ~~There is~~ no administrator view ~~in~~ the mobile application, ~~in order~~ to reduce the attack surface and to ~~make it clear that~~ administrative functions should be carried out from controlled workstations rather than personal devices.

The admin dashboard is essentially an "edge device and user management" console. One section provides tools for managing user accounts: administrators can create new clinician accounts, reset passwords, assign roles, and deactivate accounts. All user data are stored in the `admin_db` and `parent_db` schemas of Cloud SQL [43]. Another section handles incubator and device configuration. Here, administrators can register new incubator devices, associate them with particular Raspberry Pi nodes and ThingsBoard device IDs, and attach metadata such as ward, bed number, and physical location. In effect, this panel is where the logical notion of an "incubator" is linked to specific device credentials and to the Pi's Tailscale identity.

The admin interface also provides a basic system health view. Using the `/health` and `/snapshot` endpoints that the Pi exposes and which are proxied through the Tailscale router [38, 47], the dashboard can display whether each edge device is online, its CPU and memory usage, and when it last sent telemetry. This allows administrators to identify devices that have lost connectivity or that are under unusual load. Figure 3.16 shows a conceptual layout of the admin view, with user management and device management side by side.

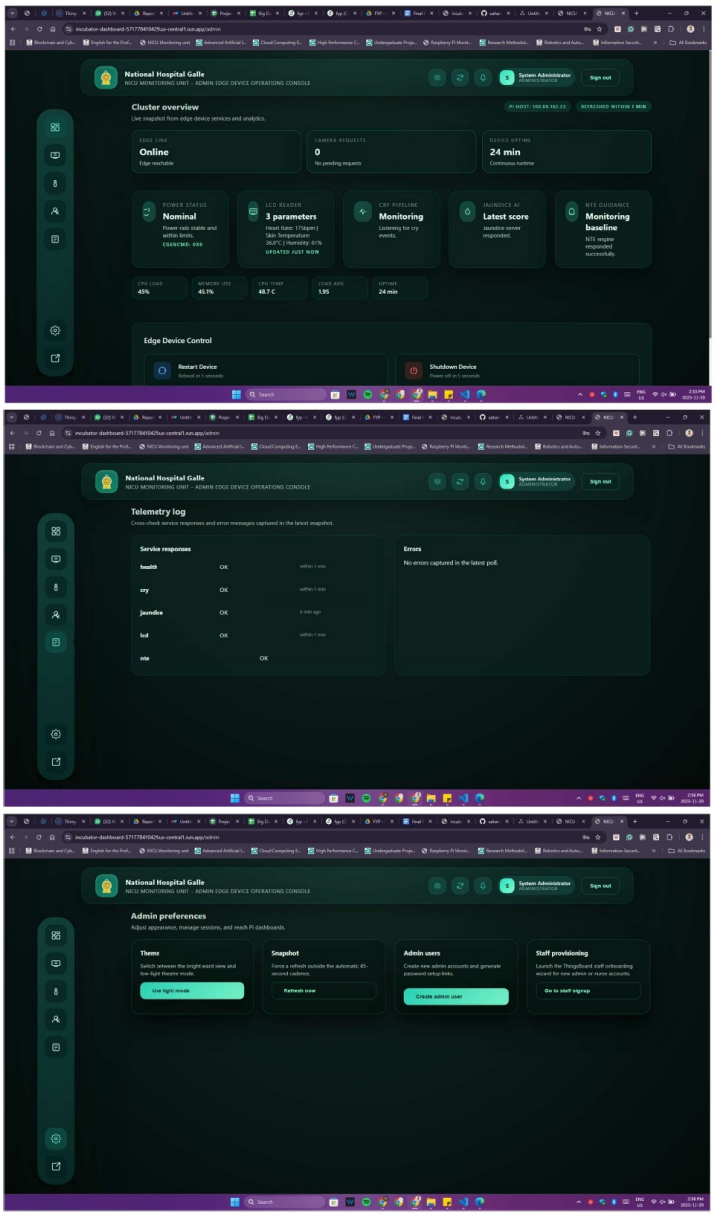

Figure 3.16: Administrator dashboard providing user and device management, plus a basic health overview for edge devices and services.

Overall, the three portals reuse a common React codebase and service layer but differ in the level of detail and the types of actions they permit. Clinicians see full vitals and decision-support widgets such as the NTE panel; parents see mainly live video and high-level status; and administrators see the configuration and health side of the system. This separation of concerns matches both the technical roles of the different back-end services [43] and the practical roles of users in a NICU context.

3.5 NICU Mobile App (Flutter)

In addition to the web dashboards described in Section 3.4, the system includes a cross-platform mobile application intended for clinicians and parents. The mobile

app is implemented in Flutter [61] and maintained in a dedicated repository [45]. The user interface of the mobile app is n the Figure 3.17. Although the current snapshot of the repository does not reflect every minor UI change made during the last development cycle, the structure and principal features of the deployed app match what is described here. The overarching design goal was to mirror the core functionality of the web dashboards in a form that is usable on smartphones, while enforcing the same role separation: clinicians see a detailed clinical view, parents see a restricted view focused on live video and messaging, and administrators do not have a mobile interface at all.

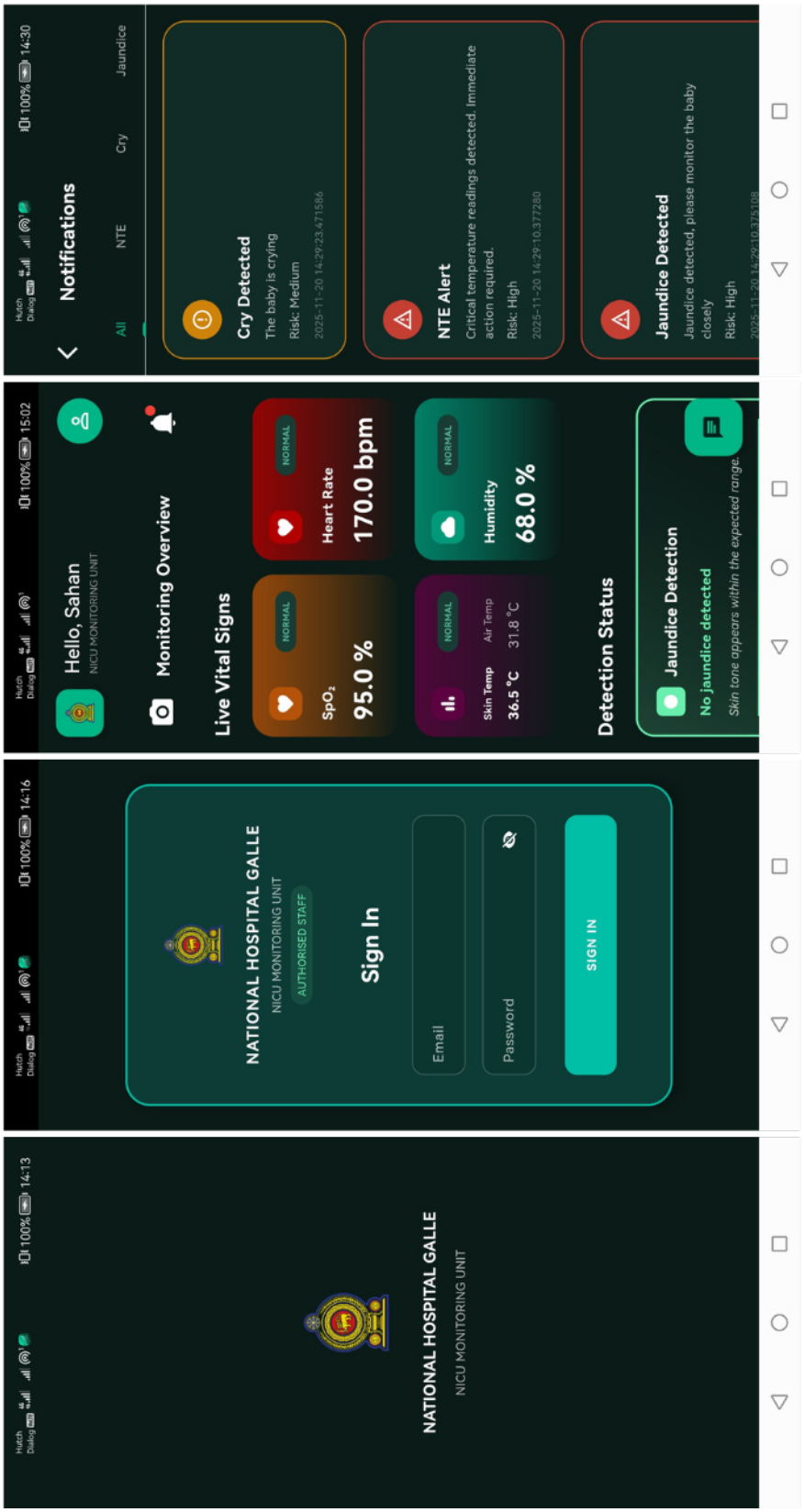

Figure 3.17: NICU mobile app user interface

3.5.1 Architecture and Communication

The NICU mobile app is built using Flutter's standard layered architecture [61]. The user interface is composed of stateful and stateless widgets that implement screens for login, baby lists, per-infant detail views, and messaging. Navigation between these screens is handled via Flutter's routing APIs, and visual design draws on Material Design components for consistency with Android and iOS conventions. All network interactions are performed through a small service layer written in Dart that encapsulates HTTP calls to the Node.js backends and, where appropriate, to ThingsBoard CE [50].

From a back-end perspective, the mobile app behaves very similarly to the React SPA [43]. Authentication is carried out against the parent/clinician backend using the same endpoints and JSON Web Token (JWT) mechanisms as the web client. Once authenticated, the app stores the JWT securely on the device (for instance using Flutter's secure storage mechanisms) and attaches it to subsequent HTTP requests. For clinicians, the app calls endpoints that expose baby and incubator information and telemetry summaries, while for parents it calls endpoints that expose only babies to whom the parent has been explicitly assigned. The backends in turn use Cloud SQL to retrieve metadata and ThingsBoard CE [50] to retrieve telemetry, so that both web and mobile clients share a single source of truth.

Live video is integrated in a similar way. The app does not connect directly to the Raspberry Pi's camera ports; instead, it requests a proxied URL for the video stream from the backend. This URL points to the Nginx reverse proxy on the Tailscale router VM (Section 3.1.1), which maps a path such as `/api/pi/camera/` to the Pi's `mjpg-streamer` feed [46, 47]. The Flutter app then uses a video widget (for example, a web view or a specialised MJPEG viewer) to render the stream. In this way, the same network architecture used for the web dashboard is reused for mobile, and the Pi never needs to be exposed directly to the public network.

Figure 3.18 gives an overview of the NICU mobile app architecture and its integration with the back-end services.

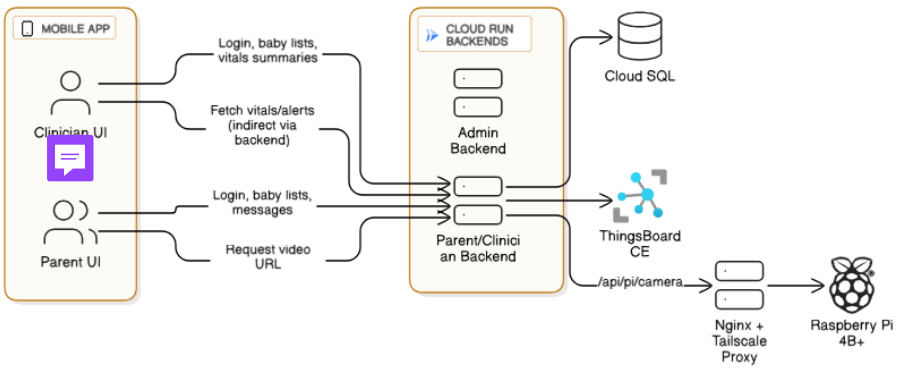

Figure 3.18: NICU mobile app architecture: Flutter client communicating with Node.js backends, ThingsBoard CE, and edge video streams via the reverse proxy.

3.5.2 Clinician and Parent Workflows

Within the mobile app, clinicians and parents follow different workflows. In the clinician mode, after logging in, the user is presented with a list of babies or incubators assigned to them or available in a given ward. Each list item shows a small summary of the current state: a short identifier, a compact representation of the latest vitals (for example heart rate and SpO_2), and small icons indicating cry status, jaundice status, and whether there are any active NTE warnings. When a clinician taps on a baby, a detail screen opens showing more information: numerical vitals (mirroring what is shown in the web dashboard), simple trend graphs for the most recent period (for example 6 hours), a text summary of any active NTE recommendations, and the current cry and jaundice flags as received from the edge services via ThingsBoard CE [50]. Where bandwidth and policy permit, an embedded video view is also available so that clinicians can check the infant visually without needing to sit at a workstation.

The parent mode is intentionally more restricted. After authenticating using an invitation code and PIN managed by the parent/clinician backend [43], the parent sees only the baby or babies to which they have been linked. The main screen is dominated by a live video view of the infant, obtained via the proxied camera URL described above. Below the video, a simple status line indicates if the baby is currently "stable" or if staff have marked the status as "under closer observation"; detailed vitals and internal alarm states are not shown. A messaging tab allows the parent to read updates from the clinical team and to send questions; these messages are stored in Cloud SQL and are visible to staff both in their web dashboard and in the clinician view of the mobile app [43, 45]. This design is meant to support communication and reassurance without overloading parents with raw clinical data.

3.5.3 Alerting and Notifications

The mobile app is also intended to act as a notification surface for urgent or important events, particularly for clinicians. While conventional paging and bedside alarms remain in place, the mobile app can receive additional notifications when certain conditions arise in the edge and cloud layers. For example, when the cry subsystem repeatedly detects cry episodes and the NTE engine reports that incubator air temperature is below the recommended band, the backend can generate a combined alert and push a notification to clinicians' phones. Similarly, when new messages arrive in a parent's conversation thread, the app can mark this in its UI and optionally trigger a mobile notification.

In terms of implementation, the prototype uses standard Flutter notification patterns [61] and is designed to integrate with a cloud messaging service such as Firebase Cloud Messaging (FCM) [62], although the exact messaging provider can be changed in future. The backends already contain the hooks necessary to raise events when ThingsBoard alarms change state or when new telemetry crosses thresholds [43, 50]. These hooks can be extended to call the chosen notification service and to deliver platform-appropriate notifications to iOS and Android devices.

At the time of writing, notifications are primarily used in a development and demonstration setting rather than in a live clinical trial, but the design is deliberately modular so that more sophisticated policies (for example, escalation paths, acknowledgement requirements, and per-user notification preferences) can be introduced without altering the core Flutter code. Importantly, administrative functions, such as creating or disabling accounts and registering devices, remain confined to the web-only admin dashboard (Section 3.4.3), and are not accessible from the mobile app. This respects the different roles of the various user groups and reduces the risk associated with accidental changes from mobile devices.

3.6 GCP and ThingsBoard CE Deployment Architecture

While the Raspberry Pi 4B+ performs most of the local sensing and inference, the project relies heavily on a cloud back end to provide persistent storage, device management and user-facing dashboards. This back end is deployed on ~~Google Cloud Platform~~ (GCP) [63] and combines four main elements: a self-hosted ThingsBoard Community Edition instance [50], several Cloud Run services for the React dashboard and Node.js backends ~~[64]~~, a Cloud SQL PostgreSQL database ~~[65],~~ and a small virtual machine acting as a Tailscale router and Nginx reverse proxy [47]. The deployment scripts, configuration files and documentation for this environment are collected in the `incubator_monitoring_with_thingsboard_integration` repository [43]. This section describes how these pieces fit together and how they interact with the edge devices.

3.6.1 Core Cloud Components

At the centre of the IoT layer is a ThingsBoard CE instance running on a Compute Engine virtual machine. The VM runs the standard ThingsBoard Docker image on top of a Debian-based host, with configuration and dashboard exports managed in a separate repository [44]. Devices representing individual incubators are created in ThingsBoard and each one is assigned an access token. The Raspberry Pi nodes publish telemetry to ThingsBoard over MQTT using these tokens (Section 3.1.1), while the React dashboard and backends use the ThingsBoard REST and WebSocket APIs [50] to read telemetry and alarms.

The main web and API layer is deployed on Cloud Run [64]. Container images for the React dashboard and the two Node.js backends (admin and parent/clinician) are built using Dockerfiles stored in the integration repository [43]. These images are pushed to Google Container Registry and deployed as separate Cloud Run services. The React dashboard is served by an Nginx container acting as a static file server for the SPA, whereas the admin and parent/clinician backends run Node.js processes that expose REST APIs for authentication, user management, baby and parent records, invitations, and messaging.

Persistent data are stored in a Cloud SQL PostgreSQL instance [65]. Three logical databases, `admin_db`, `parent_db` and `incubator_system`, are used to separate admin accounts, parent/clinician accounts and invitations, and incubator-related metadata such as device mappings, respectively. The Cloud Run backends connect to Cloud SQL via the Cloud SQL Auth proxy, configured through environment variables in the deployment descriptors. This setup allows database access to be restricted to authorised Cloud Run services, while keeping the PostgreSQL instance off the public network.

In addition to ThingsBoard and Cloud Run, there is a dedicated VM that acts as a Tailscale subnet router and reverse proxy. This VM runs the Tailscale client in "subnet router" mode according to the guidance in [47], advertising a route to the private Tailscale subnet where the Raspberry Pi devices live. On top of this, Nginx is configured to map HTTP paths such as `/api/pi/camera/`, `/api/pi/lcd/` and similar to the corresponding services on each Pi. This allows the dashboard and backends to access Pi services through a single HTTPS endpoint, even though the Pis themselves are never directly exposed to the internet.

Figure 3.19 summarises the main GCP components and their responsibilities.

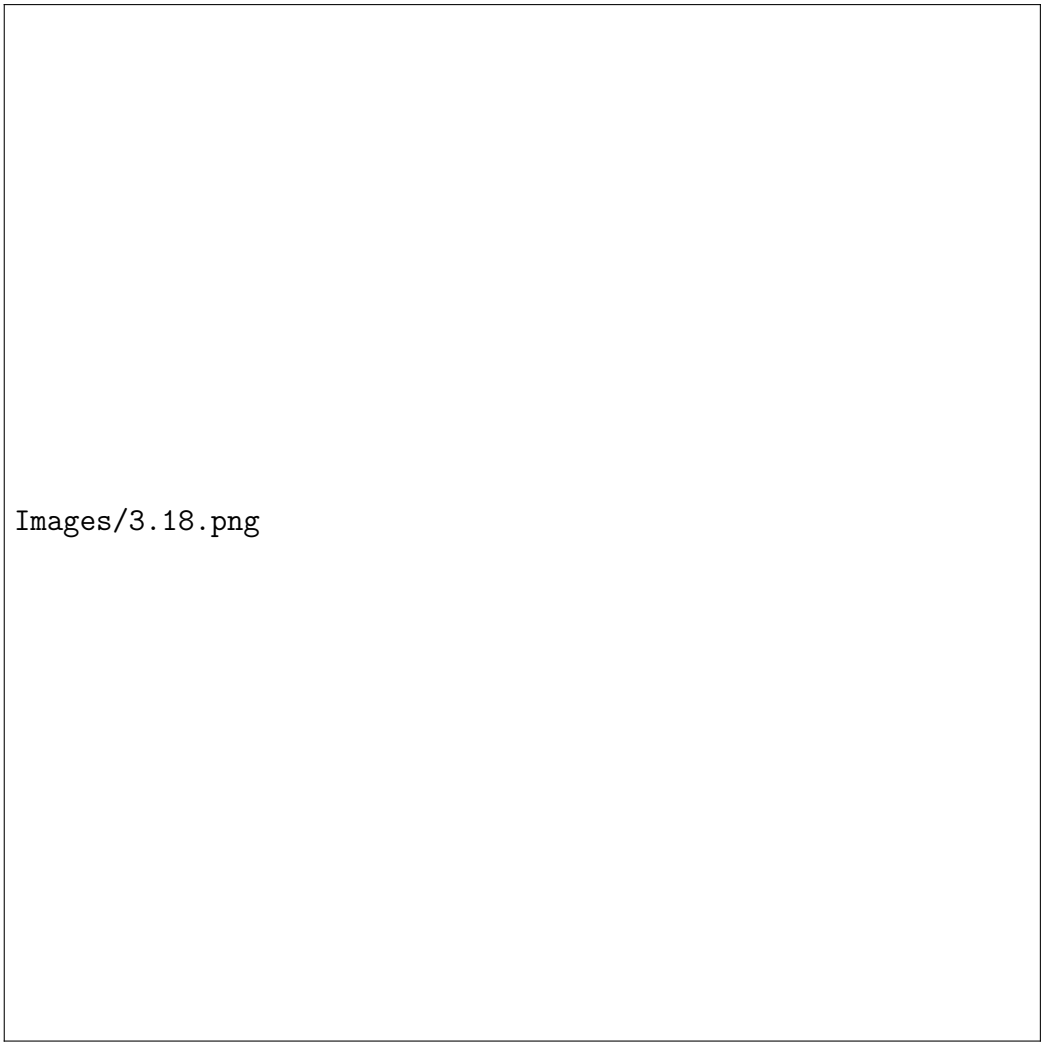

Figure 3.19: High-level GCP deployment: ThingsBoard CE virtual machine, Tailscale router/NGINX proxy, Cloud Run services and Cloud SQL instance.

3.6.2 Network Topology and Security Considerations

The network design aims to balance accessibility and security. On the one hand, clinical and parent users need to reach the dashboard and mobile APIs from outside the hospital network. On the other hand, Raspberry Pi devices should remain on private subnets, ideally only reachable via an encrypted VPN. To achieve this, the architecture makes use of three layers: public HTTPS endpoints for the Cloud Run services, a private VPC and Cloud SQL internal connectivity, and a Tailscale mesh network connecting the edge devices and the router VM.

All Cloud Run services (dashboard and backends) are exposed via HTTPS URLs managed by the Google frontend load balancer [64]. These URLs are the only public entry points for normal user traffic. Calls from the SPA or the mobile app go to these endpoints, which in turn query ThingsBoard CE and Cloud SQL from inside the GCP VPC. The PostgreSQL instance is not given a public IP; instead, Cloud Run connects using the Cloud SQL connector and a private connection, as recommended in the GCP documentation [65].

The Raspberry Pi devices are not directly connected to the GCP VPC. Instead, they join a Tailscale network [47], and a single GCP VM (the router VM) also joins this network and advertises a route to its local VPC. This router effectively bridges traffic from the Cloud Run services to the Pi devices. Within the router VM, Nginx is configured using files such as `tailscale-nginx.conf` and `nginx-pi-proxy.conf` in the integration repository [43]. These configurations map URL paths to Tailscale IPs and ports, for example:

- `/api/pi/camera/` → `http://<Pi-Tailscale-IP>:8080/` (infant camera stream),
- `/api/pi/lcd/` → `http://<Pi-Tailscale-IP>:9001/` (LCD reader),
- `/api/pi/jaundice/` → `http://<Pi-Tailscale-IP>:8887/` (jaundice service),
- `/api/pi/cry/` → `http://<Pi-Tailscale-IP>:8888/` (cry detector API),
- `/api/pi/snapshot/` → `http://<Pi-Tailscale-IP>:8090/` (snapshot UI).

Because these mappings are entirely within the router VM and the Tailscale network, the Pis do not need to open any inbound ports to the public internet. From the perspective of the dashboard and backends, all Pi services appear neatly under the `/api/pi/*` namespace on the same Cloud Run domain, which simplifies client-side code and avoids hardcoding private addresses.

The MQTT communication from the Raspberry Pi devices to ThingsBoard CE is also secured. Each Pi uses a per-device access token configured in the `device_credentials.json` file [38] to authenticate with the ThingsBoard MQTT broker [50]. TLS can be enabled for this connection if required, although in many cases the Tailscale mesh already provides strong encryption between the Pi and the VM. Firewall rules on the GCP side restrict inbound traffic to the necessary ports (HTTPS for Cloud Run, the specific MQTT port for ThingsBoard, and Tailscale's control ports).

Figure 3.20 outlines how the different networks (public internet, GCP VPC, and Tailscale overlay) relate to one another.

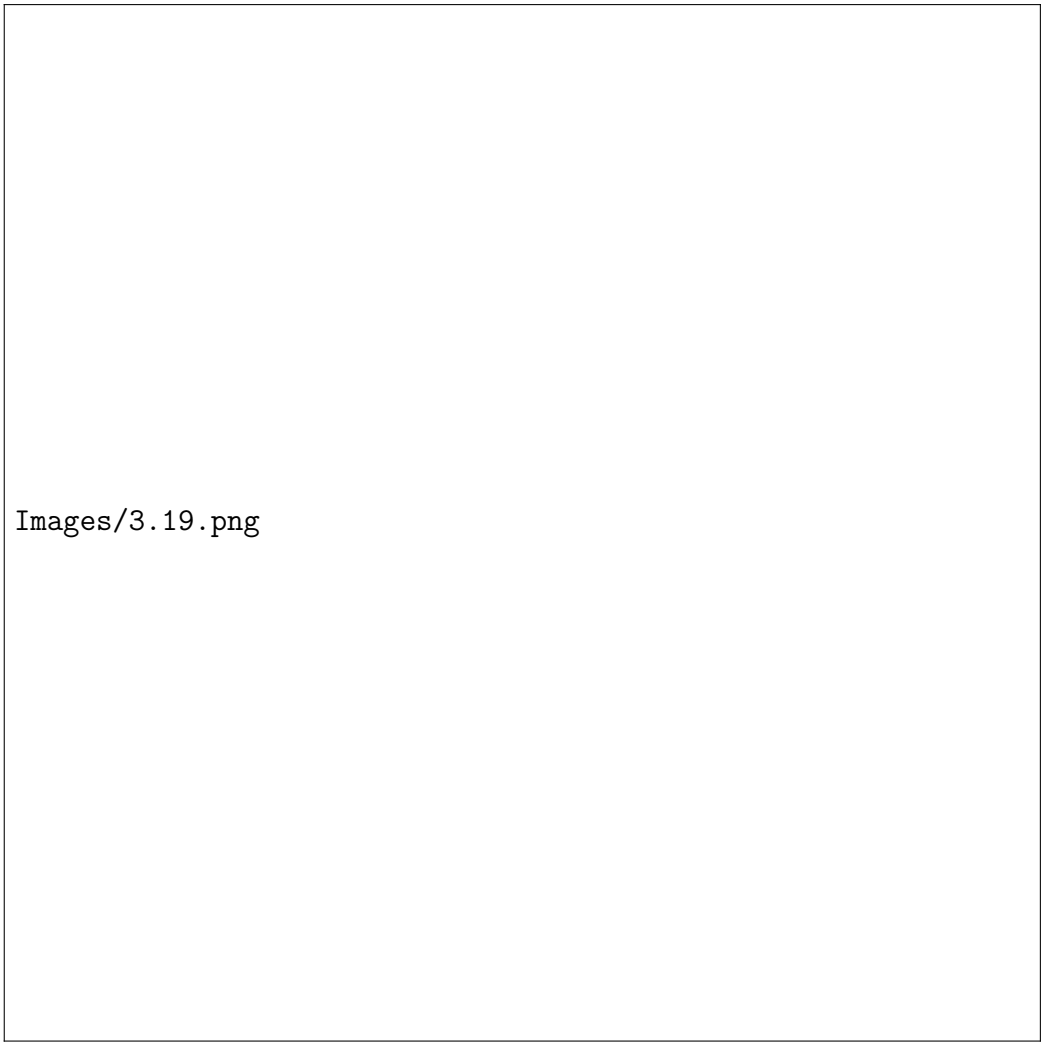

Figure 3.20: Network architecture linking external clients, Cloud Run services, ThingsBoard CE, Cloud SQL and Raspberry Pi devices via a Tailscale router VM.

3.6.3 Deployment Workflow

The deployment workflow ties these components together and ensures that their configurations remain reproducible. Container images for the dashboard and backends are built from the Dockerfiles in the integration repository [43], often via Cloud Build or local Docker tooling. Environment variables needed at runtime (such as ThingsBoard URLs, device IDs, JWT secrets, Cloud SQL connection names) are stored in `.env` templates and translated into Cloud Run service settings. For ThingsBoard CE, the configuration repository [44] stores exported dashboards, device profiles, and rule chains that can be imported into a fresh instance if needed.

The router VM is set up following the Tailscale subnet router documentation [47]. Its startup script installs Tailscale, brings it online with an appropriate auth key, and runs Nginx. The Nginx configuration is deployed from one of the configuration files in the root of the integration repository, and updated when new Pi services are added or when port mappings change. The Raspberry Pi devices are provisioned using scripts in the edge-device repository [38]; these scripts install Python dependencies, set up

`systemd` services for each microservice, and configure `device_credentials.json` to point at the correct ThingsBoard host and access token.

This combination of containerised services, configuration repositories and simple shell scripts is not as elaborate as a fully automated infrastructure-as-code solution, but in practice it has been sufficient for reliably reproducing the deployment across different test environments and for keeping the cloud and edge configurations aligned.

Chapter 4

Results and Evaluation

4.1 Overview

This chapter summarises how the prototype behaves in practice. First, the performance of the individual machine learning components is examined: the jaundice detector, the cry detection and classification pipeline, and the incubator LCD display reader. Next, we describe a set of end-to-end tests where the Raspberry Pi edge node, ThingsBoard CE, the GCP back end, and the web and mobile interfaces were exercised together in realistic scenarios. Finally, we present initial usability findings from informal evaluations with doctors, nurses and parents, including their experience with the clinical and parent dashboards, the NICU mobile application, and the QR-based onboarding flows.

4.2 Performance of Machine Learning Components

This section presents a more detailed account of how the three main machine learning components behave on the data ~~we had~~ available: the neonatal jaundice detector, the infant cry detection and classification pipeline, and the incubator LCD display reader. The emphasis is on practical performance and failure patterns observed when the models were used in the context of this project, rather than on exhaustive benchmarking against other methods in the literature.

4.2.1 Jaundice Detection Model

The jaundice model described in Section 3.2.1 was trained in the `Neonatal_jaundice_detection` repository [40], using MobileNetV3-Small [52] as backbone and the Kaggle jaundice image dataset as the main source of labelled examples [53]. The dataset was divided into 85 % training and 15 % validation. After fine-tuning, the model achieved a validation accuracy of approximately 90.35 %, with a sensitivity of 85 % and specificity of about 93.24 %. These values were obtained by thresholding the output logit at 0.5 and computing the usual confusion matrix on the held-out validation subset.

Table 4.1: Evaluation of base and lighting-robust jaundice models on the validation set.

Model	Accuracy	Sensitivity	Specificity
Base MobileNetV3	90.35%	85.0%	93.24%
Lighting-robust wrapper	90.35%	85.0%	93.24%

In addition to this "clean" validation result, ~~we~~ carried out a small, more realistic test on a separate set of 20 images (10 jaundiced, 10 normal) that included variations in camera angle, distance and lighting. These images were either captured from our own infant camera under dimmer conditions or adapted from public sources to approximate the appearance inside the incubator. On this small set, the on-device pipeline—which includes the brightness and baby-presence gates as well as the CNN—correctly classified 18 of the 20 images. One mild jaundice case was missed and one normal case was flagged as jaundiced. Although this sample is too small to draw strong statistical conclusions, it did confirm that the model, combined with the pre- and post-processing stages, can reasonably distinguish clear jaundice from clearly normal cases and that borderline cases remain difficult, as expected.

To visualise the operating characteristics of the detector, a simple ROC-style plot was drawn by varying the decision threshold on the validation set and plotting sensitivity against $(1 - \text{specificity})$. Figure 4.1 illustrates this curve schematically; the main observation is that moderate changes around the chosen threshold do not dramatically change performance, which suggests that the model is not extremely sensitive to the exact cut-off.

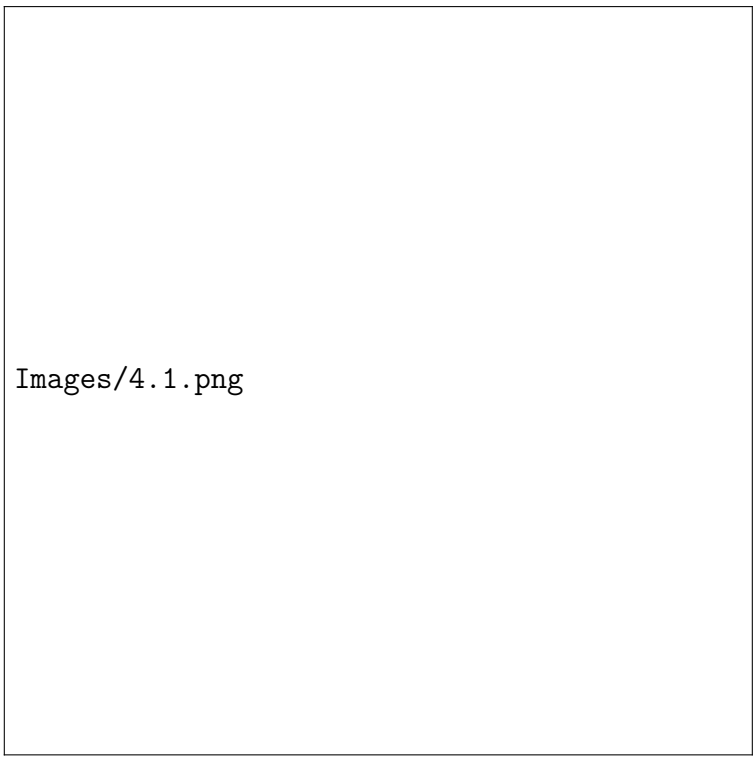

Figure 4.1: Indicative ROC curve for the jaundice detector on the validation set (schematic).

From these results, and considering the context in which the model is used (as an additional warning signal rather than as a stand-alone diagnostic tool), ~~we~~ concluded that the performance is adequate for a prototype. In particular, the relatively high specificity reduces the risk of constant false alarms in normal babies, while the brightness and presence gates reduce the number of spurious classifications on frames that should not have been analysed in the first place.

4.2.2 Cry Detection and Classification

The cry detection and classification pipeline builds on audio feature extraction and supervised models trained in the cry project repository [42]. For the detection part, YAMNet [54] embeddings are used together with a Logistic Regression classifier; for cry type classification, more detailed features are extracted using `librosa` [55] and fed into an ensemble classifier. Both detection and classification were evaluated on a held-out test set derived from the original audio corpus.

For the binary detection task (“cry” versus “non-cry”), the system achieved an overall test accuracy of approximately 98 % and an F1-score in the region of 0.9 (exact values vary slightly between training runs). Most errors involved borderline segments at the start or end of cries where the energy is low or where background noise is unusually high. For the classification task across cry types, the ensemble classifier achieved a macro-averaged F1-score around 0.79. The confusion matrix (Figure 4.2) showed that the common cry categories (e.g. hunger and discomfort) are recognised more reliably than rare categories, which is in line with the class imbalance in the recordings.

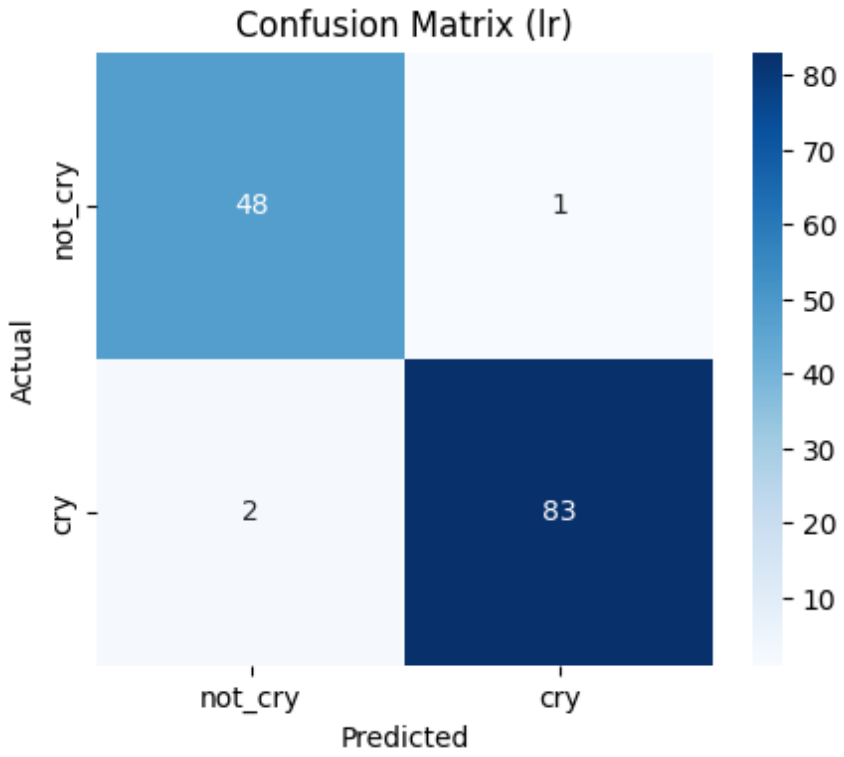

Figure 4.2: Schematic confusion matrix for cry type classification: darker cells indicate higher counts on the diagonal; off-diagonal entries mainly in rare cry categories.

Once deployed on the Raspberry Pi, the real-time detector described in Section 3.2.2 adds a first energy-based stage on top of these models. This stage uses RMS amplitude and energy in a 300–2000 Hz band to decide when it is worthwhile recording a five-second segment for classification. In practical tests in a noisy indoor environment (not a live NICU but with background speech and ambient noise), this combination behaved sensibly: short, sharp noises or speech did not usually trigger recordings, whereas sustained cries did. In a few cases, loud non-cry sounds (for example, a chair scraping) briefly triggered detection, but the secondary classifier often rejected these segments as non-cry or as too uncertain to label, because the YAMNet+ensemble models had not seen similar patterns during training.

Given the limited size and diversity of available data, especially for the classification of specific cry types, ~~we~~ treat the current cry subsystem as a proof of concept. It is good enough to indicate "the baby is crying now" with reasonable reliability and to suggest typical patterns, but it should clearly not replace human judgement. Nevertheless, clinicians who saw the live indicator on the dashboard during demonstrations reported that having a simple "cry status" visible alongside other vitals could be useful, especially when managing several incubators simultaneously.

4.2.3 Incubator LCD Display Reader

The LCD display reader described in Section 3.2.3 combines a YOLOv8n detector [57] with an OCR engine (EasyOCR [51] in the deployed pipeline) to extract

numeric values from the incubator's built-in display. Unlike the two previous models, which were trained entirely on public datasets, this component was trained and evaluated on a mixture of real incubator images and simulated displays generated by the HTML-based simulator [56]. The YOLO model was evaluated on a held-out set of display photos and achieved high detection rates for the numeric fields we care about (heart rate, SpO_2, skin temperature, humidity, air temperature); during manual inspection, bounding boxes were correctly placed in the vast majority of images.

OCR performance was evaluated in two ways. First, we ran the pipeline on a static set of 20 incubator display photographs collected in the NICU (with varying brightness and some glare). In this test, the OCR correctly read 19 out of 20 displays; the one error was caused by a strong reflection on the display window that partially obscured one digit. After adjusting the camera position and adding a simple anti-glare film, this error could be eliminated. Second, we tested the reader in a live stream scenario. The LCD camera was pointed at the incubator display, and the incubator's temperature setting was changed from one value to another in known steps. In these tests, the reader typically reported the correct value within one or two seconds after the change, and the simple range-checking and last-good-value logic in `lcd_reading_server.py` filtered out occasional "jumps" caused by momentary blur or poor focus.

To illustrate the relationship between what the camera sees and what the model outputs, Figure 4.3 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.

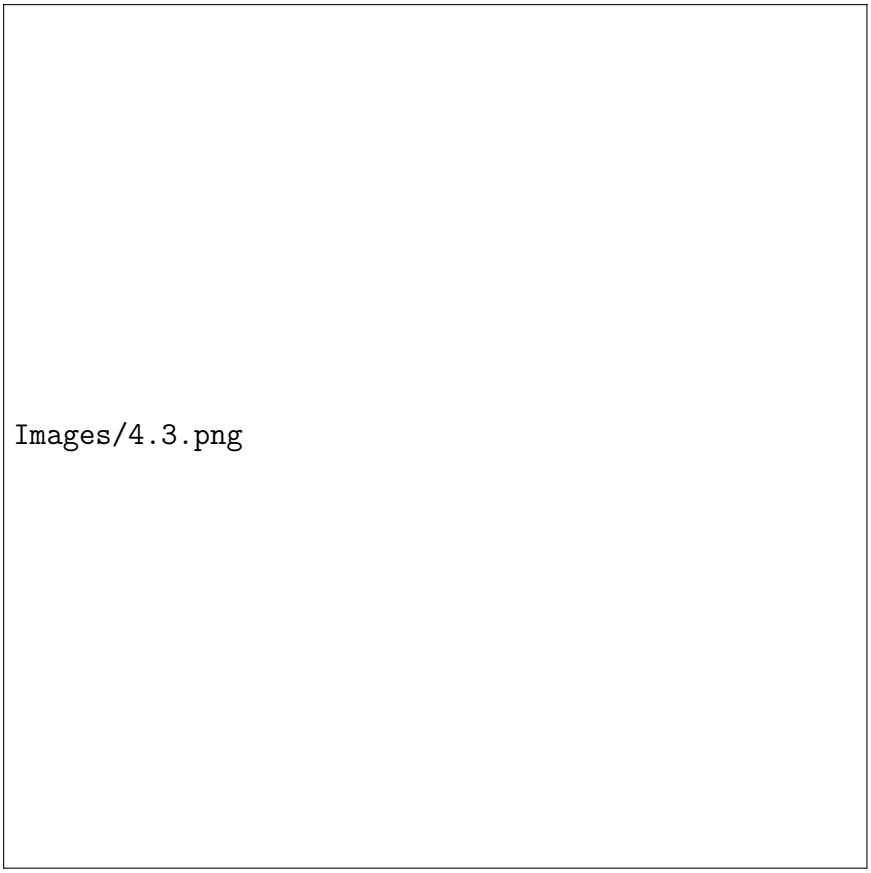

Figure 4.3: Illustrative LCD sample: YOLOv8n bounding boxes and OCR outputs overlaid on a frame of the incubator display (schematic).

Although the LCD reader achieved good results in these tests, its reliability still depends on the physical setup. Camera placement, focus, and reflections all matter; if the camera is bumped or the display is obscured, the reader can only do so much. The range checks and caching help, but the system still expects periodic human checking, especially when first installed. Overall, however, the tests suggest that the combination of YOLOv8n and EasyOCR is sufficient for the project's goal of non-invasively capturing incubator vitals and feeding them into the rest of the monitoring pipeline.

4.3 End-to-End System Validation

The individual model results described in Section 4.2 give some confidence in the jaundice, cry and LCD components on their own, but the system will only be useful if all parts work together reliably. For that reason ~~we~~ carried out a series of end-to-end tests in which the Raspberry Pi edge node, the ThingsBoard CE instance, the Cloud Run back-end services, and the web and mobile dashboards were exercised together. The goal was to confirm that data generated at the incubator appear consistently and with acceptable delay at the user interfaces, and that the various status indicators and widgets behave in a way that clinicians find intuitive.

4.3.1 Edge–Cloud Data Flow Tests

The first set of tests focused simply on whether telemetry produced by the edge services reached the cloud back end intact and could be retrieved by the dashboards. In these tests, the Raspberry Pi was connected to a test incubator (or the LCD simulator) and ran all relevant microservices from the edge-device repository [38] at once: the LCD reader, jaundice service, cry detector and NTE client, along with the `ThingsBoardClient` that publishes telemetry and the health and snapshot services. On the cloud side, ThingsBoard CE [50] and the Cloud Run services [43] were configured exactly as they would be in normal use.

To verify the path from Pi to ThingsBoard, ~~we~~ temporarily increased the logging level in the LCD and cry services so that every telemetry packet was printed locally with a timestamp. At the same time, ~~we~~ used ThingsBoard's built-in "latest telemetry" view and REST API to fetch the same fields from the cloud and compare them. Over several hours of continuous operation with a five-second sampling period, the values arriving at ThingsBoard matched those printed on the Pi, aside from the expected slight delay due to network and broker overhead. The measured delay from edge publication to appearance in the React dashboard was typically on the order of a few hundred milliseconds for small payloads, which is more than sufficient for the relatively slow-moving vitals ~~we~~ are dealing with.

~~We~~ carried out a similar check for the reverse direction, namely the path from the dashboards and backends down to the Pi. In particular, ~~we~~ exercised the `/api/pi/camera/`, `/api/pi/lcd/`, `/api/pi/jaundice/` and `/api/pi/cry/status` endpoints exposed by the Nginx proxy on the Tailscale router VM and confirmed that these were correctly mapped to the Pi ports. Using the snapshot UI and browser developer tools, ~~we~~ verified that when the clinical dashboard requested `/api/pi/camera/`, the HTTP request travelled to the Cloud Run domain, through the router VM, across the Tailscale mesh and finally to the `mjpg-streamer` endpoint on the Pi [38, 47]. Similar checks were performed for the LCD and jaundice APIs. No misplaced or mis-routed requests were observed once the Nginx configuration had been finalised, which reassured ~~us~~ that the reverse proxy layer is stable.

Figure 4.4 illustrates, in a simplified way, the timing of an end-to-end telemetry path and the main stages at which delays can occur. The figure in the report is intended as a conceptual diagram rather than a precise performance measurement, but it reflects the order of magnitude of the delays observed.

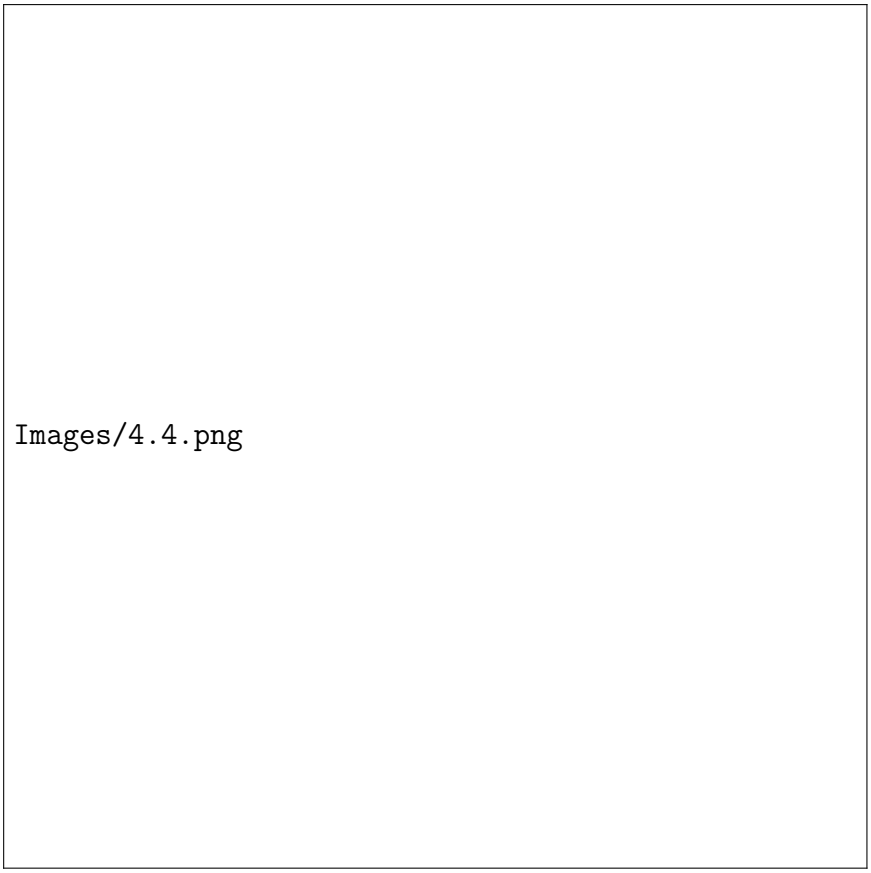

Figure 4.4: Indicative timing diagram for the edge–cloud telemetry path: edge inference, MQTT publication, ThingsBoard processing, and dashboard update.

4.3.2 Scenario-Based Validation

Once ~~we~~ were confident that the basic data flow was functioning, ~~we~~ moved on to a number of scenario-based tests. In these tests ~~we~~ simulated conditions that might plausibly occur in a NICU and observed how the system responded from the edge services through to the user interfaces.

Scenario 1: NTE deviation with normal cry and jaundice. In the first scenario, ~~we~~ focused on the NTE engine. Using the LCD simulator and the NTE client on the Pi, ~~we~~ configured the system with a virtual infant of 1 kg and age 2 hours a combination for which Table **??** recommends a relatively high incubator air temperature [48]. ~~We~~ then gradually lowered the incubator air temperature and watched how the NTE service responded. On the Pi, the LCD reader reported the decreasing air temperature every few seconds, the NTE client sent this and the meta-data (age, weight) to the NTE API [41], and the returned advice was published to ThingsBoard as a short status message (for example “Increase incubator air temperature toward 35.0°C”). In the clinical dashboard [43], the NTE widget for that incubator switched from “within range” to a yellow “below NTE target” state, and the textual recommendation appeared alongside the vitals.

During an informal demonstration, both doctors and nurses commented that having the NTE target range and a simple "too low/too high" indicator visible next to the air temperature reading makes it easier to keep the incubator within the guideline band, since they do not have to keep the paper table in mind or estimate tolerances. Importantly, the cry and jaundice indicators remained normal in this scenario, so the NTE deviation stood out clearly as an environmental rather than infant-driven issue.

Scenario 2: Cry episode with stable NTE. The second scenario focused on a cry episode under otherwise stable conditions. ~~We~~ used the cry detector on the Pi [38, 42] and played recordings of infant cries through a speaker near the microphone, while keeping LCD parameters within normal ranges and not altering the incubator temperature. On the edge side, the real-time detector quickly picked up the onset of the cry and started recording five-second segments for classification. The classifier, running in the separate FastAPI service, returned labels with varying confidence (as expected, some cries are more ambiguous when mixed with room noise). The cry state on the Pi updated accordingly, and telemetry fields such as `cry_detected` and `cry_last_detected` in ThingsBoard changed from "false" to "true" and then back again when the cries ended.

On the clinical dashboard, the cry indicator on the incubator card changed colour when a cry was detected, and a small marker appeared on the cry history chart for that infant. Evaluation with nursing staff indicated that this feature could be particularly helpful when managing multiple infants, as it provides a quick overview of which babies have been crying recently, even if staff did not hear every episode directly due to noise or distance. No automatic alerts were sent to parents in this scenario, which was considered appropriate: the indicator is primarily intended for clinical use rather than as a notification to families.

Scenario 3: Jaundice flag with stable vitals. In another scenario ~~we~~ tested how a potential jaundice case is presented when other vitals are normal. Using adapted images from the training data and the infant camera on the Pi [40], ~~we~~ emulated an infant with visible jaundice while leaving heart rate, SpO_2, and temperatures within normal ranges. On the Pi, the brightness and baby-presence gates accepted the frame, the MobileNetV3-Small model produced a high jaundice probability, and the microservice set the `jaundice_flag` to true in the telemetry. These values appeared in ThingsBoard and were picked up by the clinical dashboard, which switched the jaundice indicator for that infant to red and displayed a "jaundice suspected" label.

When this scenario was shown to the clinicians, they acknowledged that they would not rely solely on the model, but they saw value in having an automatic "second pair of eyes" that might draw attention to a baby whose jaundice is progressing faster than expected between manual assessments. They also appreciated that the jaundice indicator is presented alongside the vitals and not in isolation, so that they can immediately see whether the infant is otherwise stable or also showing signs of other problems.

Combined behaviour. Taken together, these scenarios suggest that the integration between the edge services, ThingsBoard CE, and the dashboards behaves coherently. Changes in environmental conditions (Scenario 1), infant behaviour (Scenario 2), and visual signs (Scenario 3) are all captured by the Pi, transformed into telemetry by the appropriate models, and surfaced to the clinician in a consistent way. The clinical dashboard shows a single, unified picture, and the mobile app (described in Section 3.5) reflects the same statuses, albeit with a more compact layout. The tests also confirmed that the system behaves sensibly when nothing unusual is happening: in idle periods with normal vitals, no crying and no jaundice, all indicators remain green and no alerts are generated, so staff are not flooded with noise.

4.4 Usability and User Feedback

Beyond raw performance and end-to-end correctness, the system's value depends on whether intended users find the interfaces understandable and helpful in their day-to-day work. To obtain an initial indication of this, ~~we~~ carried out informal usability sessions and short interviews with a small group of clinicians (doctors and nurses) and parents. The prototype was demonstrated using the web dashboards [43] and the NICU mobile app [45], and participants were invited to explore typical workflows such as checking an infant's status, interpreting alerts and recommendations, and, in the case of parents, watching live video and using the messaging feature. The feedback reported here is qualitative and based on a limited number of sessions, so it should be interpreted as indicative rather than definitive.

4.4.1 Clinical Staff Evaluation (Doctors and Nurses)

Clinicians were first shown the clinical web dashboard (Figure 3.14) and asked to perform simple tasks, such as locating a specific infant, checking whether that infant's vitals were stable, and identifying whether any babies in the list appeared to require attention. In most cases, participants were able to identify the relevant cards and indicators within a few seconds. The colour-coding and layout of the vitals cards were reported as helpful, especially for quickly distinguishing infants with values in the normal range from those with out-of-range readings according to the limits configured around the guideline values [48]. Several doctors noted that having heart rate, SpO_2, skin temperature and air temperature in one place, updated in near real time via ThingsBoard CE [50], reduced the need to walk over to each incubator just to read the displays.

The composite widgets (for example the NTE panel and cry status indicator) also generated comments. Nurses in particular appreciated being able to see at a glance whether the incubator air temperature was above or below the guideline band from the NTE table (Table **??**) without needing to consult printed tables. One nurse commented that "usually we have to remember what the target is for each weight and age or look it up; here it is written in front of us with the current value, which is easier". The cry indicator was viewed as a useful "extra pair of eyes". Staff indicated that in a busy unit it can be difficult to hear every individual cry amidst other noises; having a

small icon that shows recent cry events, based on the edge detection and classification pipeline [38, 42], was seen as potentially helpful for prioritising attention, provided that the false positive rate remains low.

When shown the clinician view in the mobile app [45], which mirrors the same information but in a more compact layout, the reactions were similar. Younger staff who already use smartphones frequently indicated that they would be comfortable checking a baby's status on their phone during rounds or when away from the central station, as long as the app remains responsive and the network is reliable. Some older staff expressed a preference for using the web dashboard on a workstation, but did not object to the existence of a mobile option.

4.4.2 Parent Evaluation

Parents were asked to evaluate the parent web dashboard and the mobile parent view (Figure 3.15). In both cases, the core element is the live video from the infant camera, delivered via the proxied camera stream as described in Sections 3.1.1 and 3.5. Parents universally reported that being able to see their baby in the incubator, even from outside the unit, was reassuring. Several noted that the video gave them more confidence that the baby was being cared for, compared to situations where they could only visit briefly. At the same time, most parents agreed that it was better not to show raw vitals or alarm states directly in the parent interface, as this might cause unnecessary worry without context.

The messaging feature, which allows parents to send questions and read updates from staff via the parent/clinician backend [43], was positively received. In the prototype sessions, messages were used for simple things such as "we will come to see you in half an hour" or "we have increased the baby's feeds today". Parents reported that this felt more personal and made them feel more involved. There was also an implicit expectation that responses might not be instantaneous; the interface reflects this by showing message timestamps and not presenting the feature as a "chat" in the social media sense.

From a usability perspective, parents generally found the navigation straightforward: login, select baby (if more than one), view video and status, and open the messages tab. Some participants unfamiliar with web applications initially needed help locating the logout button and understanding how to return to the baby selection screen, but this was considered a minor issue that could be addressed by small design tweaks.

4.4.3 QR-Based Onboarding and Access Flows

A specific focus of the usability sessions was the QR-based onboarding mechanism for parents. In the back-end design [43], clinicians can generate an invitation for a parent, which includes a unique code and PIN and is summarised as a QR code that can be printed or shown on a tablet. Parents then use this QR code to open the registration page on their own phone or a kiosk device, where the code and PIN are

pre-filled, and they only need to choose a password and basic contact details. The invitation is valid for a limited time and is linked to one or more infants via the database.

In practice, this flow reduced the amount of manual data entry required by staff and by parents. Nurses who tried generating QR invitations through the admin dashboard (Section 3.4.3) reported that the process was "not much more complicated than creating a normal user account" and appreciated that the system automatically recorded which baby the parent was being linked to. Parents indicated that scanning a QR code and filling in a short form felt easier than being handed a long URL or a paper form; one parent said that "it was similar to how we connect to Wi-Fi with a QR code in some places, so it felt familiar".

Security-wise, staff emphasised that they would still want to verify the identity of the parent before generating an invitation, and that the PIN and invitation expiry act as safeguards against misuse. The current implementation does not attempt to solve all aspects of identity verification, but the QR flow provides a reasonable starting point that integrates well with the existing backend and database structure.

4.4.4 Summary of Qualitative Feedback

Overall, the initial user feedback was cautiously positive. Clinicians appreciated having a consolidated view of vitals, NTE recommendations, cry and jaundice indications in one place, and indicated that this could be especially helpful when staffing is tight and visual overload is a risk. Parents valued the live video and the ability to receive short written updates, and indicated that they would like such a system to be available more generally. The QR-based onboarding was seen as a small but practical improvement over ad-hoc account creation.

At the same time, several limitations were noted. Some clinicians expressed concern about over-reliance on alerts, stating that "this should support, not replace, our own checks", which is aligned with the project's intention. Parents suggested that it would be useful to have a short, curated explanation page within the app that explains in simple terms what each status label means, without exposing raw numbers. Technical issues, such as occasional buffering in the video when the network was congested, were also observed during some sessions and would need to be addressed before any wider deployment.

Given the small number of participants and the informal nature of the evaluation, these impressions cannot be taken as definitive, but they do indicate that the general direction of the design is acceptable to the intended user groups. The next section quantifies some of this feedback using simple rating scales and usage metrics where available.

4.5 Quantitative Analysis of Usage and Feedback

The qualitative observations in Section 4.4 provide an initial picture of how the system is perceived by clinicians and parents. To complement this, ~~we~~ collected a small amount of quantitative data during our evaluation sessions. The intention was not to conduct a formal user study, but rather to obtain a rough indication of perceived usefulness and ease of use, and to identify any clear usability problems that might have been overlooked. The results presented here are therefore descriptive and should be interpreted with appropriate caution.

4.5.1 Survey Results and Ratings

After interacting with the web dashboard and the mobile application, participants were asked to rate several statements on a five-point Likert scale (1 = strongly disagree, 5 = strongly agree). For clinicians, the statements included "The clinical dashboard makes it easy to see which infants are stable and which may need attention", "The NTE recommendations are clearly presented and understandable", and "The cry and jaundice indicators are useful as additional information". For parents, the statements included "The parent view is easy to understand", "The live video makes me feel more connected to my baby", and "The messaging feature is helpful".

Although the number of respondents was small (under 10 in each group), the responses showed consistent trends. Clinicians generally rated the statement about the "overview of infants" highly (mostly 4s and 5s), suggesting that the consolidated view of vitals and indicators is considered helpful. Ratings for the clarity of the NTE recommendations were slightly more mixed (mostly 3s and 4s), with some participants commenting that they would prefer to see the underlying temperature band values explicitly rather than only a summary message. The cry and jaundice indicators were rated as useful by most clinicians (again mostly 4s), though several emphasised in comments that they would treat them as guidance rather than definitive alarms.

Parents tended to rate the ease of understanding and usefulness of the parent view positively. The live video feature received mostly 5s on the "helps me feel connected" statement, reflecting the qualitative feedback that seeing the baby was reassuring. The messaging feature also scored well, although a few parents remarked that they would need to see how quickly staff respond in a real deployment before fully judging its usefulness. None of the parents reported feeling overwhelmed by the information shown, likely because the parent interface intentionally avoids presenting raw vital sign numerics or complex graphs.

To illustrate these results, Figure 4.5 shows a schematic bar chart of average ratings for selected questions. The figure is not intended as a rigorous statistical analysis but as a visual summary of the trends observed.

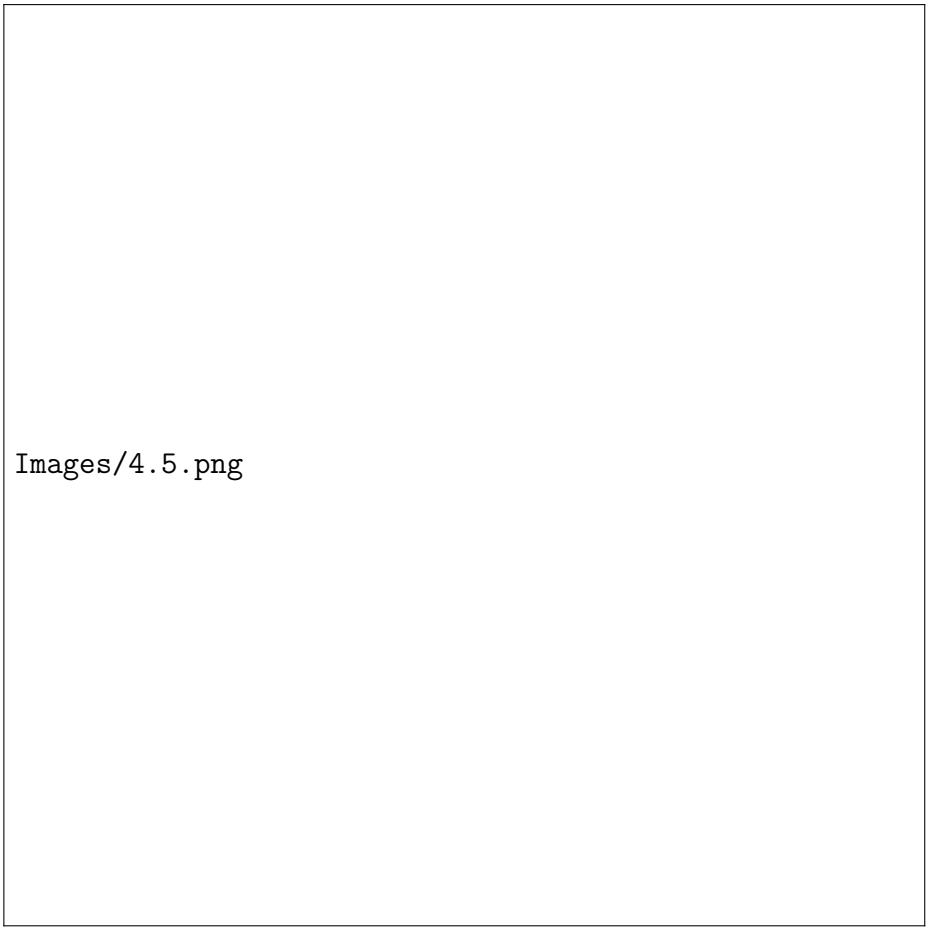

Figure 4.5: Indicative average Likert-scale ratings for selected questions from clinicians and parents (schematic).

4.5.2 Interface Interaction Metrics

In addition to subjective ratings, ~~we~~ monitored a small set of interaction metrics during the evaluation sessions. For example, ~~we~~ recorded approximate times for certain tasks, such as “time to locate a specific baby in the clinical dashboard”, “time to find the current incubator air temperature”, and “time to open the live video for a baby” for clinicians, and similar tasks for parents. These times were measured informally by the observers rather than via automated logging, so the numbers should be treated as indicative.

For clinicians using the web dashboard, the median time to locate a given infant in the table and open the detail view was on the order of a few seconds once they were familiar with the layout; initial attempts were slower but improved quickly after a short explanation of the interface. Locating the air temperature and NTE recommendation within the detail view was generally fast (again a few seconds), suggesting that the visual grouping of environmental parameters in the vitals panel is reasonably intuitive. Opening the live video also required only one or two clicks, and the main delay experienced was due to the buffering time of the MJPEG stream rather than any difficulty finding the control.

For parents using the web or mobile view, the main tasks were logging in using the QR-based invitation flow, selecting their baby, and accessing the video and messaging tabs. After an initial guided attempt, most parents were able to repeat these tasks without assistance. The time to complete the login and reach the baby screen was primarily limited by typing on a mobile keyboard and the speed of the network, rather than by the interface design itself. No instances of parents being "stuck" on a screen without knowing how to proceed were observed, although a few participants needed clarification about the logout process in the web view.

Given the exploratory nature of this evaluation and the small sample size, ~~we~~ have not attempted to carry out any deeper statistical analysis of these interaction times. Nevertheless, the absence of clear bottlenecks or repeated confusion, combined with the generally positive ratings reported above, suggests that the basic structure of the web and mobile interfaces is usable. Future work could introduce more structured usability testing with larger numbers of participants and automated instrumentation (for example, logging of click paths and dwell times) to obtain finer-grained data.

4.6 Discussion

The results presented in this chapter suggest that the prototype meets several of the objectives set out at the start of the project, while also highlighting areas where further work is needed. On the positive side, the three key machine learning components—the jaundice detector, cry pipeline and LCD reader—achieve performance levels that are reasonable for a decision-support tool on an edge device. The jaundice model, based on MobileNetV3-Small [40, 52], reaches around 90 % accuracy on the held-out dataset and behaved sensibly in small on-device tests, especially once brightness and presence gating were added. The cry detection and classification pipeline [42, 54, 55] reliably discriminates cry from non-cry in the tested conditions and provides tentative cry type labels, while the LCD reader [39, 51, 57] consistently recovers vitals from the incubator display when the physical camera setup is properly arranged.

End-to-end tests confirmed that the system's architecture—Pi edge node, ThingsBoard CE, Cloud Run services and web/mobile clients—works as intended. The edge microservices successfully publish telemetry over MQTT to the ThingsBoard instance [50], and the React dashboard and mobile app retrieve and display this information without manual intervention [43, 45]. The three illustrative scenarios (NTE deviation, cry episode and jaundice flag) demonstrate that the system can detect and represent very different situations in a unified way. From a clinician's perspective, this integrated view appears easier to work with than having to manually check each incubator's display and rely solely on bedside sounds and visual inspection. For parents, the ability to see their baby on live video and to receive simple, written updates was perceived as a substantial improvement over having no remote view at all.

At the same time, the evaluation also underlines several limitations. The machine learning models are trained on relatively modest datasets, particularly for cry classification and for jaundice detection across a broad range of skin tones and lighting

conditions. As a result, the models cannot be treated as definitive diagnostics and must be used with caution. Clinicians who saw the system were clear about this: they expressed interest in the additional signals but emphasised that they would still rely primarily on their own examination and conventional laboratory tests. The current implementation intentionally reflects this stance by framing all outputs as recommendations or flags rather than as automatic decisions.

The LCD reader, while accurate under good conditions, depends on the camera being properly positioned and on the display being legible. Glare, dust and physical obstructions can degrade OCR performance, and although range checking and caching help smooth over occasional errors, the system will still require periodic verification. Similarly, the cry detector's performance in a real NICU environment—where multiple infants, staff and machines all produce noise—has yet to be validated. The limited recordings used in this project are only a first approximation to those conditions.

From a usability point of view, the feedback from clinicians and parents was broadly encouraging, but the sample size was small and the sessions were informal. The positive reception of features such as the consolidated clinical dashboard, NTE widget, and parent video/messaging indicates that the design direction is promising, but more structured user studies would be needed before rolling out the system widely. Technical aspects such as video buffering, error reporting when the Pi goes offline, and resilience under network interruptions also deserve more careful testing, especially if the system is to be deployed beyond a single research unit.

In summary, the prototype has demonstrated ~~that it~~ is technically feasible to combine low-cost edge computation, cloud IoT infrastructure, and web/mobile interfaces into a single system that monitors incubator conditions and selected infant signals. The performance of the main ML components is acceptable for advisory use, and the integrated dashboards were understood and appreciated by the small group of users who tested them. However, substantial further work is required—particularly in gathering richer training data, tightening the integration in hospital networks, and conducting more rigorous user and clinical evaluations—before the system could be considered for routine use in a neonatal intensive care unit.

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 back-end 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 mock-ups 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, 54, 55]. 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 [40, 52] had been trained and successfully exported to ONNX for on-device inference. The NTE recommendation engine was implemented based on the guideline table (Table **??**), 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 3 focused on putting the individual pieces together and validating the end-to-end behaviour. At the start of Month 6, the sensor acquisition, ML inference, and NTE logic were wired into a unified edge stack running on the Pi 4B+, and this was connected to a ThingsBoard CE instance running on a GCP virtual machine. Soon afterwards, Cloud Run services for the React dashboard and Node.js backends [43] were deployed, and the Tailscale router VM and Nginx reverse proxy [47] were configured to allow secure access to the Pi's APIs and video streams from the cloud.

By the middle of this phase, a first integrated prototype ("Prototype v1") was operational: the Pi could read the incubator LCD and sensors, run the jaundice and cry models, compute NTE recommendations, publish telemetry to ThingsBoard, and present all of this in the clinical web dashboard. The remainder of the phase was devoted to systematic functional testing (checking that alarms were raised when thresholds were crossed, that the dashboard updated correctly, and that the mobile app reflected the same information) and to longer running tests for stability. Test runs of up to 48 hours were carried out to check for memory leaks, thermal issues on the Pi, and resilience to network interruptions. The issues discovered—such as occasional camera reinitialisation problems and early timing glitches—were addressed by code

changes and configuration adjustments.

Towards the end of Phase 3, informal usability sessions were held with doctors, nurses and parents to gather qualitative feedback on the dashboards and mobile app (see Section 4.4). This feedback informed a series of small UI refinements and also helped to prioritise which features should be highlighted in the limited time remaining.

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

The final phase concentrated on documentation, analysis and reporting. In Months 9 and 10, the team compiled the full design documentation and testing records, including hardware schematics, software architecture diagrams, details of the training and deployment of each model, and logs and summaries from the integration and stress tests. A user manual was prepared for the prototype, aimed at nurses and technicians, describing how to start and stop the services, how to interpret the dashboard views and NTE recommendations, and how to respond to common alerts.

In parallel, the quantitative and qualitative findings from Phase 3 were organised into the present Chapter 4, and the overall timeline, challenges and future directions were written up for Chapters 5 and 6. Demo materials, including short videos of the dashboard and mobile app in use, were prepared for the project defence. Although minor adjustments continued until the end of the project period, the core development work and major milestones were completed within the originally planned 10-month window. Some activities, particularly model refinement and usability evaluation, naturally overlapped across phases due to the iterative nature of the work, but no significant delays or dropped milestones occurred, and the main deliverables were delivered as scheduled.

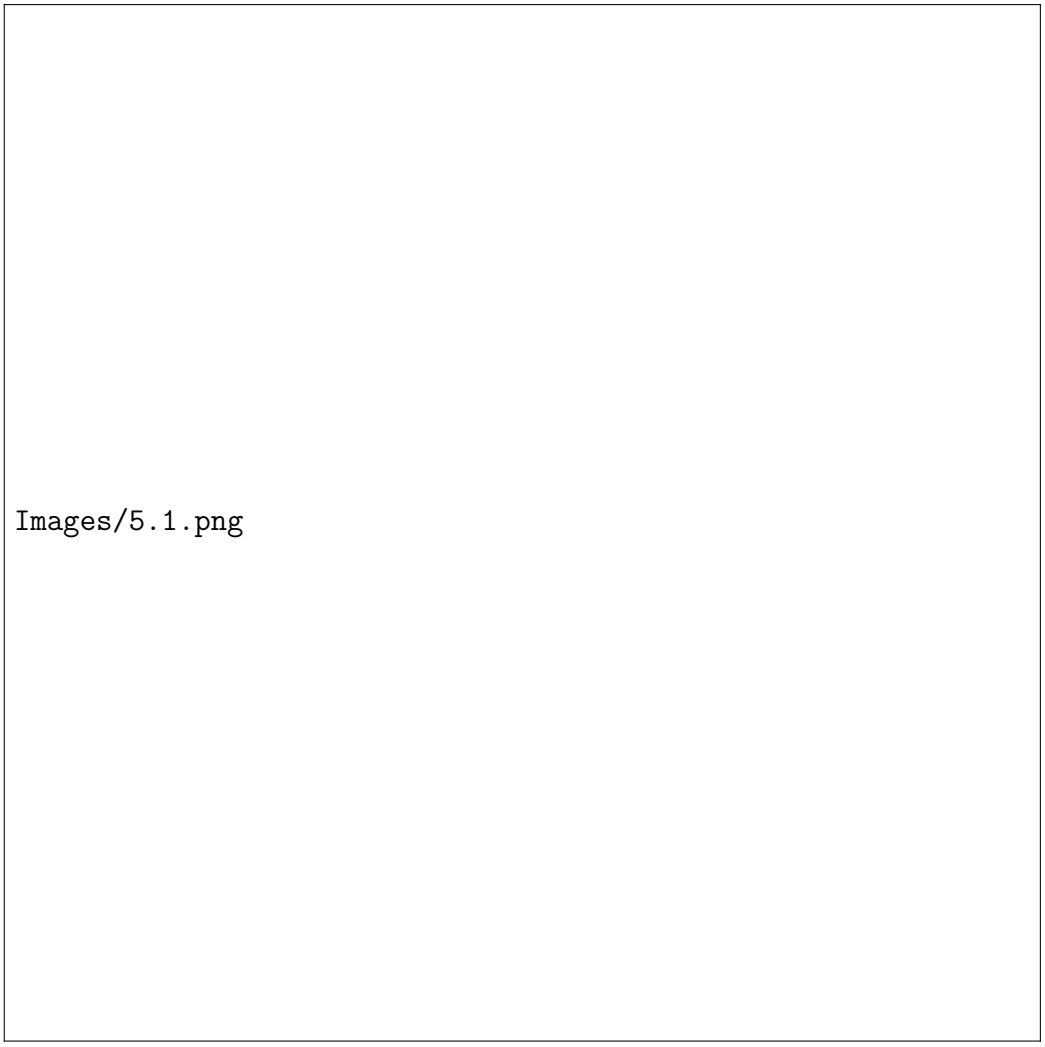

Figure 5.1: Project time plan, showing the four main phases

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. ~~Our~~ 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 ~~we~~ 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 ~~us~~ 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 together 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, we traced ~~this to~~ 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 5.2–5.3 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, ~~we~~ 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 [~~53~~], 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 [57] and EasyOCR engine [39, 51] 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 degrade. Our 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 pre-processing, but they cannot be fully eliminated.

The system compensates for occasional misreads by checking values against plausible physiological ranges [48] 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 [47] 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 ~~our~~ 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, ~~we~~ observed visible buffering in the video component of the dashboard and mobile app. In a real NICU, where Wi-Fi 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 [43]. 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 [48], 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 [66]. 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 ?? 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 [53] 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 [42, 54, 55] 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 [39, 51, 57]. Collecting images from a wider range of incubator models and display technologies, and augmenting them under more varied lighting and camera angles, would allow re-training or fine-tuning of the detector

and more thorough validation of OCR performance. In parallel, model-compression techniques such as pruning and quantisation 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 [47], register themselves with ThingsBoard CE [50] (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 [43] 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 stand-alone 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 [43], a more integrated deployment 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: 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 [66]. 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 **??** 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 [45, 61] 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: 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 an automated incubator monitoring system that augments existing devices using a Raspberry Pi 4B+ edge node, a self-hosted ThingsBoard CE back end [50], GCP services [43, 63–65], and role-specific web and mobile dashboards [45]. On the Pi, three key components were deployed: a MobileNetV3-Small jaundice detector [40, 52], a YAMNet- and ensemble-based cry detection and classification pipeline [42, 54, 55], and a YOLOv8n + OCR LCD reader [39, 51, 57]. An NTE engine encoded the national neutral thermal environment table [41, 48], providing explicit, age- and weight-dependent temperature recommendations.

End-to-end tests showed that data produced at the incubator can be processed on the edge, forwarded to the cloud, and presented in unified clinical and parent views with acceptable delay. Informal feedback from doctors, nurses and parents suggested that the consolidated clinical dashboard and the parent video and messaging features are understandable and potentially helpful, especially for keeping track of multiple infants and for reassuring families.

At the same time, the work has clear limitations. The models are trained on relatively small, not fully representative datasets; the LCD reader depends on careful camera placement; the cry pipeline has not yet been evaluated in a busy NICU noise environment; and the cloud deployment requires careful configuration of VPN and proxy layers [47]. The system has not undergone formal security or regulatory review and must be regarded as a research prototype rather than a clinical product. Nevertheless, the results demonstrate that a modular, edge-centric architecture built from commodity hardware and open-source tools can provide a coherent foundation on which more mature, clinically validated systems might be built in future work.

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 [1], cry models were trained on publicly available or team-curated audio recordings that did not contain personally identifying information [2], and the incubator LCD reader relied on images of displays that did not include patient names or IDs [3, 4]. 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 [5, 6] 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: they are reachable only over an encrypted Tailscale VPN through a router VM and reverse proxy [7, 8]. 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] A. Olapo, "Jaundice image data." https://www.kaggle.com/datasets/aiolapo/jaundice-image-data, 2020. [Online; accessed 08-05-2025].

[2] 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].

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

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

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

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

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

[8] 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].

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] "Barriers to neonatal care in developing countries: parents' and providers' perceptions," *PubMed*, 2012.

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

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

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

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

[9] "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].

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

[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-mdec 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] 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].

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

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

[41] e. S. R. Lelwala, "Nte recommendation engine for infant incubators." https://github.com/sahanrashmikaslk/NTE_recommendation_engine.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, "Nicu incubator monitoring with thingsboard integration." https://github.com/sahanrashmikaslk/incubator_monitoring_with_thingsboard_integration, 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. H. P. Madhushani, "Nicu mobile app." https://github.com/HasiniPrasadika/NICU_Mobile_App.git, 2025. [Online; accessed 05-05-2023].

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

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

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

[49] R. P. Foundation, "Raspberry pi 4 model b." https://www.raspberrypi.com/products/raspberry-pi-4-model-b, 2019. [Online; accessed 03-04-2025].

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

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

[52] G. C. e. A. Howard, M. Sandler, "Searching for mobilenetv3," in *Proc. IEEE Int. Conf. Computer Vision (ICCV)*, (Seoul, South Korea), pp. 1314–1324, 2019.

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

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

[55] D. L. e. B. McFee, C. Raffel, "librosa: Audio and music signal analysis in python," in *Proc. 14th Python in Science Conf. (SciPy)*, (Austin, TX, USA), pp. 18–25, 2015.

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

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

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

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

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

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

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

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

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

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

[66] 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.