

SmartPaw Assistant:
Intelligent Pet Care Through Image Recognition and Machine Learning

By
Lim Xin Yi

A REPORT
SUBMITTED TO
Universiti Tunku Abdul Rahman
in partial fulfillment of the requirements
for the degree of
BACHELOR OF COMPUTER SCIENCE (HONOURS)
Faculty of Information and Communication Technology
(Kampar Campus)

FEBRUARY 2026

COPYRIGHT STATEMENT

© 2026 Lim Xin Yi. All rights reserved.

This Final Year Project proposal is submitted in partial fulfillment of the requirements for the degree of Bachelor of Computer Science (Honours) at Universiti Tunku Abdul Rahman (UTAR). This Final Year Project proposal represents the work of the author, except where due acknowledgment has been made in the text. No part of this Final Year Project proposal may be reproduced, stored, or transmitted in any form or by any means, whether electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the author or UTAR, in accordance with UTAR's Intellectual Property Policy.

ACKNOWLEDGEMENTS

I would like to express my gratitude to my supervisor, Dr. Tan Joi San, for her guidance, valuable comments and constant support throughout the period of this project. Her suggestions and motivation have been instrumental in guiding my ideas and in the successful completion of this assignment. The experience and learning I gained under her supervision are invaluable and will remain a significant part of my educational and personal growth. In the absence of her motivation and encouragement, the effective completion of this assignment would never have been possible.

Besides that, I would also like to offer my sincere appreciation to my friends and family for their unconditional support, patience and motivation. Their kind heartedness, understanding and belief in me have bestowed the power of heart and will to continue, especially through problematic and difficult times. Their constant words of encouragement and their availability to be with me in times of tension and uncertainty, have been a huge comfort. I am truly grateful for their unconditional love and support, which have been crucial in helping me to complete the project successfully.

ABSTRACT

The surge in pet ownership has revealed some constraints in pet care applications, such as overdependence on manual data entry, inadequate personalisation and lack of tools for understanding pet sentiment. The contribution of this work is the SmartPaw Assistant, an AI-based mobile application using Flutter, for the integrated solution. The app offers several functionalities, such as analysing pet food nutrition with image recognition, evaluating health status through image scanning, generating automated timeline-based health reports, calendar reminders and an expense tracker, as well as a community interaction and a price comparison tool. At the heart of the framework lies a robust dual-model pet emotion interpretation functionality, which is a CNN-BiLSTM model for audio classification that reaches an accuracy of 89% and a MobileNetV2 model for visual body language recognition that reaches an accuracy of 87%. To improve user-specific adaptability, a personalisation strategy based on transfer learning is exploited for offline model updates by utilizing user feedback every two weeks. This process enables an increasing and near-perfect accuracy for each pet. The backend is built with FastAPI and Supabase Database with Ngrok connecting during development. In general, SmartPaw Assistant showcases the promise of an AI-powered and comprehensive pet care service that minimizes manual control, personalises and improves the effectiveness of pet care management.

Area of Study: Image Recognition, Machine Learning, Mobile Application Development

Keyword: Smart Pet Management, Image Recognition, Supervised Learning, Deep Learning, Transfer Learning, Nutrition and Health Analysis, Pet Emotion Interpretation, MobileNetV2, CNN-BiLSTM

TABLE OF CONTENT

TITLE PAGE	i
COPYRIGHT STATEMENT	ii
ACKNOWLEDGEMENTS	iii
ABSTRACT	iv
LIST OF FIGURES	ix
LIST OF TABLES	xv
LIST OF ABBREVIATIONS	xvi
CHAPTER 1: Project Background	1
1.1 Introduction	1
1.2 Problem Statement	3
1.3 Motivation	6
1.4 Project Objectives	7
1.5 Project Scope	9
1.6 Impact, Significance and Contribution	10
CHAPTER 2: Literature Review	11
2.1 Review of Existing Application	11
2.1.1 Pawdi – Pet Food Scanner	11
2.1.2 TTcare – Pet Health Detector	14
2.1.3 MeowTalk – Pet Sound Interpretation Tool	17
2.1.4 11Pets – Pet Care Management Application	19
2.2 Review of Previous Works on Interpreting Pet Emotion	23
2.2.1 Domestic Cat Sound Classification Using Learned Features from Deep Neural Nets	23
2.2.2 DeepCat: A Deep Learning Approach to Understand Cat’s Body Language	24
2.3 Limitation and Proposed Solution	27
2.3.1 Limitation of Existing App	27
2.3.2 Limitation of Previous Studies	30
2.3.3 Proposed Solution	30
CHAPTER 3: Proposed Methodology	35
3.1 System Methodology	35
3.2 Tool to Use	36

3.2.1 Hardware Architecture	36
3.2.2 Software Architecture	37
3.3 System Design	40
3.3.1 Architecture Diagram	40
3.3.2 Use Case Diagram	42
3.3.3 Activity Diagram	44
3.3.3.1 Sign In Activity Diagram	44
3.3.3.2 Sign Up Activity Diagram	45
3.3.3.3 Pet Food Nutrition Analysis Activity Diagram	46
3.3.3.4 Pet Health Analysis Activity Diagram	47
3.3.3.5 Pet Emotion Interpretation Activity Diagram	48
3.3.3.6 Pet Calendar Reminder Activity Diagram	49
3.3.3.7 Pet Expense Tracker Activity Diagram	50
3.3.3.8 Pet Community Interaction Activity Diagram	51
3.3.3.9 Price Comparison and Price Drop Alert Activity Diagram	52
3.3.3.10 Edit Profile Activity Diagram	53
3.3.3.11 Logout Activity Diagram	53
3.4 Project Timeline	54
CHAPTER 4: System Design	55
4.1 System Overview Flow Chart	55
4.2 Database Design	57
4.3 Module Design	58
4.3.1 Sign In Sequence Flow	58
4.3.2 Sign Up Sequence Flow	59
4.3.3 Pet Food Nutrition Analysis Sequence Flow	59
4.3.4 Pet Health Analysis Sequence Flow	60
4.3.5 Pet Emotion Interpretation Sequence Flow	61
4.3.6 Pet Calendar Reminder Sequence Flow	62
4.3.7 Pet Expense Tracker Sequence Flow	62
4.3.8 Pet Community Interaction Sequence Flow	63
4.3.9 Price Tracker Sequence Flow	64

4.4 Model Architecture	65
4.4.1 Sound Interpretation - Hybrid CNN + BiLSTM Model	65
4.4.2 Visual Interpretation - MobileNetV2 Pretrained Model	67
4.5 Implementation Issues and Challenges	69
4.6 Summary	70
CHAPTER 5: System Implementation and Testing	71
5.1 Setting and Configuration	71
5.1.1 Frontend Setup	71
5.1.2 Backend Setup	72
5.1.3 Database Setup	73
5.1.4 Communication Bridge – Ngrok	74
5.2 System Operation	75
5.2.1 Sign In, Sign Up and Question Screen Development	75
5.2.2 Home Screen and Logout Feature Development	78
5.2.3 Pet Profile Edit Feature Development	79
5.2.4 Pet Food Nutrition Analysis Feature Development	81
5.2.5 Pet Health Analysis Feature Development	83
5.2.6 Pet Sound Interpretation Feature Development	88
5.2.7 Pet Calendar Reminder Feature Development	95
5.2.8 Pet Expense Tracker Development	98
5.2.9 Pet Community Interaction Development	101
5.2.10 Price Tracker Development	104
5.3 Pet Emotion Interpretation Model Training	107
5.3.1 Sound Interpretation Model	107
5.3.1.1 Dataset	107
5.3.1.2 Data Preprocessing	107
5.3.1.3 Dataset Splitting	109
5.3.1.4 Model Training	110
5.3.1.5 Performance Evaluation	113
5.3.1.6 Model Comparison	117
5.3.2 Visual Interpretation Model	119

5.3.2.1 Dataset	119
5.3.2.2 Data Preprocessing	119
5.3.2.3 Dataset Splitting	121
5.3.2.4 Model Training	121
5.3.2.5 Performance Evaluation	123
5.3.2.6 Model Comparison	128
5.3.3 Personalised Model via Transfer Learning	131
5.3.3.1 Personalisation Strategy Overview	131
5.3.3.2 Transfer Learning Implementation	131
5.3.3.3 Proof of Learning	134
5.4 Summary	137
CHAPTER 6: Conclusion and Recommendation	138
6.1 Project Review, Discussion and Conclusion	138
6.2 Novelties and Contribution	138
6.3 Future Work	139
REFERENCES	140
APPENDIX	A-1
POSTER DESIGN	A-1

LIST OF FIGURES

Figure Number	Title	Page
Figure 1.1.1	Relationship Between Pets and Owners	1
Figure 1.1.2	Nutrition Analysis	2
Figure 1.2.1	Problem of Manual Input	4
Figure 1.2.2	Misunderstanding Pet Emotion	5
Figure 1.2.3	Personalised Care	5
Figure 2.1.1	Scanning Barcode Functionality in Pawdi	12
Figure 2.1.2	Overall Nutrition Analysis in Pawdi	12
Figure 2.1.3	Nutrition Score Evaluation in Pawdi	13
Figure 2.1.4	Pros and Cons Description in Pawdi	13
Figure 2.1.5	History Review Functionality in Pawdi	14
Figure 2.1.6	User Interface in TTcare	16
Figure 2.1.7	Scanning Functionality in TTcare	16
Figure 2.1.8	Health Analysis in TTcare	17
Figure 2.1.9	User Interface in MeowTalk	18
Figure 2.1.10	Manually Recording Unfamiliar Voice in MeowTalk	19
Figure 2.1.11	Manually Entering Translation for Unfamiliar Voice in MeowTalk	19
Figure 2.1.12	Home Page in 11Pets	20
Figure 2.1.13	Management System in 11Pets	21
Figure 2.1.14	Auto-generate Schedule in 11Pets	21
Figure 2.1.15	Shareable Report in 11Pets	22
Figure 2.1.16	Invitation Link in 11Pets	22
Figure 2.2.1	Deep Neural Net Model Architecture Diagram	23
Figure 2.2.2	Preprocessing Procedures Before Implementing DeepCat Algorithms	25
Figure 2.2.3	Overview of the DeepCat Algorithms with Three Selected Features	26
Figure 2.3.1	Moderate Nutritional Score of Chocolate Although It Is Unsuitable for Pet	28
Figure 2.3.2	Unsuitable Description That Easily Be Overlooked by Users	28

Figure 2.3.3	Receive Alert Message When Harmful Food is Detected	31
Figure 3.1.1	Prototyping Methodology	35
Figure 3.3.1	Architecture Diagram of Pet Care Application	40
Figure 3.3.2	Use Case Diagram of Pet Care Application	42
Figure 3.3.3	Sign In Activity Diagram	44
Figure 3.3.4	Sign Up Activity Diagram	45
Figure 3.3.5	Pet Food Nutrition Analysis Activity Diagram	46
Figure 3.3.6	Pet Health Analysis Activity Diagram	47
Figure 3.3.7	Pet Emotion Interpretation Activity Diagram	48
Figure 3.3.8	Pet Calendar Reminder Activity Diagram	49
Figure 3.3.9	Pet Expense Tracker Activity Diagram	50
Figure 3.3.10	Pet Community Interaction Activity Diagram	51
Figure 3.3.11	Price Comparison and Price Drop Alert Activity Diagram	52
Figure 3.3.12	Edit Profile Activity Diagram	53
Figure 3.3.13	Logout Activity Diagram	53
Figure 3.4.1	Project's Timeline	54
Figure 4.1.1	Flow Chart of Pet Care Application	55
Figure 4.2.1	ERD Diagram of Pet Care Application	57
Figure 4.3.1	Sign In Sequence Flow Diagram	58
Figure 4.3.2	Sign Up Sequence Flow Diagram	59
Figure 4.3.3	Pet Food Nutrition Analysis Sequence Flow Diagram	59
Figure 4.3.4	Pet Health Analysis Sequence Flow Diagram	60
Figure 4.3.5	Pet Emotion Interpretation Sequence Flow Diagram	61
Figure 4.3.6	Pet Calendar Reminder Sequence Flow Diagram	62
Figure 4.3.7	Pet Expense Tracker Sequence Flow Diagram	62
Figure 4.3.8	Pet Community Interaction Sequence Flow Diagram	63
Figure 4.3.9	Price Tracker Sequence Flow Diagram	64
Figure 4.4.1	Sound Model Architecture Diagram	65
Figure 4.4.1	Visual Model Architecture Diagram	67
Figure 5.1.1	Screenshot of IDE's File Explorer	71
Figure 5.1.2	Screenshot of pubspec.yaml File	71
Figure 5.1.3	FastAPI Swagger UI	72
Figure 5.1.4	Screenshot of Showing Unicorn Terminal Run Successfully	72

Figure 5.1.5	Screenshot of Pet Table	73
Figure 5.1.6	Screenshot of Supabase Storage Dashboard	73
Figure 5.1.7	Screenshot of Active Ngrok Session	74
Figure 5.1.8	Example of Base URL Setup	74
Figure 5.2.1	Sign In Screen	75
Figure 5.2.2	Error Message Appears When Email or Password Incorrect	75
Figure 5.2.3	Sign In Bloc	75
Figure 5.2.4	Sign Up Screen When Password Format Correct	76
Figure 5.2.5	Format Reminder Remain White Color If Password Format Wrong	76
Figure 5.2.6	Sign Up Bloc	76
Figure 5.2.7	First Question Screen (Pet Name)	77
Figure 5.2.8	Second Question Screen (Pet Breed)	77
Figure 5.2.9	Third Question Screen (Pet Age)	78
Figure 5.2.10	Fourth Question Screen (Pet Weight)	78
Figure 5.2.11	Fifth Question Screen (Pet Allergy)	78
Figure 5.2.12	Account Successfully Created in Supabase Authentication	78
Figure 5.2.13	Home Screen	79
Figure 5.2.14	Switch to Another Pet	79
Figure 5.2.15	Edit Profile Screen	80
Figure 5.2.16	Successfully Edit Message	80
Figure 5.2.17	Pet Name Before Change in Database	80
Figure 5.2.18	Pet Name After Change in Database	80
Figure 5.2.19	Food Nutrition Analysis Screen	81
Figure 5.2.20	Food Analysis Result	81
Figure 5.2.21	Image Uploading State Transition	82
Figure 5.2.22	Parallel Fetching of Pet Data and Image Identification	82
Figure 5.2.23	Commercial Food Database Fuzzy Matching Function	82
Figure 5.2.24	Allergy Alert Message	83
Figure 5.2.25	Allergic Food Detected	83
Figure 5.2.26	Health Analysis Screen	84
Figure 5.2.27	Navigation Menu	84
Figure 5.2.28	Skin Analysis Results	85

Figure 5.2.29	Stool Analysis Results	85
Figure 5.2.30	Bodu Size Analysis Results	86
Figure 5.2.31	Download Report	87
Figure 5.2.32	Report Period Selection	87
Figure 5.2.33	Health Report Send via Email	87
Figure 5.2.34	Health Report Sample	87
Figure 5.2.35	Generate Report Based on Time Interval	88
Figure 5.2.36	Pet Emotion Interpretation Screen	89
Figure 5.2.37	Video Source Upload Method	89
Figure 5.2.38	Sound Model Processing	90
Figure 5.2.39	Visual Model Processing	90
Figure 5.2.40	Rulebook for Final Prediction	91
Figure 5.2.41	Pet Emotion Interpretation Flow	91
Figure 5.2.42	Emotion Interpretation Result	92
Figure 5.2.43	Feedback Section	92
Figure 5.2.44	Correct Feedback	93
Figure 5.2.45	Correction Label When Predict Wrongly	93
Figure 5.2.46	Screenshot of Supabase Multimodal Table	93
Figure 5.2.47	Feedback Logic	94
Figure 5.2.48	Supervised Fine-tuning Training	94
Figure 5.2.49	Personalised Model Created	94
Figure 5.2.50	Pet Calendar Reminder Screen	95
Figure 5.2.51	Activity Addition Form	95
Figure 5.2.52	Activity Successfully Saved Message	96
Figure 5.2.53	Activity Successfully Deleted Message	96
Figure 5.2.54	Reminder Notification	96
Figure 5.2.55	Navigation to Vet Clinic	97
Figure 5.2.56	Calendar Event Backend Logic	97
Figure 5.2.57	Notification Setup and Autogenerated Next Scheduled Event	98
Figure 5.2.58	Expense Tracker Screen	99
Figure 5.2.59	Category Filtering	99
Figure 5.2.60	New Expense Creation	100

Figure 5.2.61	Expense Updation	100
Figure 5.2.62	Expense Splitting	100
Figure 5.2.63	Detail Autogenerated Logic	100
Figure 5.2.64	Community Interaction Screen	101
Figure 5.2.65	Breed-Specific Tips Database	102
Figure 5.2.66	Specific Tips Logic	102
Figure 5.2.67	Post Creation	103
Figure 5.2.68	Edition or Deletion	103
Figure 5.2.69	Provide Comment on Post	103
Figure 5.2.70	Saved and Like Posts	103
Figure 5.2.71	Search Bar for Price Comparison	104
Figure 5.2.72	Top 3 Lowest-Price Product Display	104
Figure 5.2.73	Redirect User to Online Store	105
Figure 5.2.74	Target Price of Price Drop Alert	105
Figure 5.2.75	Notification Shown When Price Drop	105
Figure 5.2.76	Query Search Using SerpAPI	106
Figure 5.2.77	Price Drop Alert Logic	106
Figure 5.3.1	Dataset Summary in Sound Model	107
Figure 5.3.2	MFCC Function in Sound Model	108
Figure 5.3.3	Z-Score Normalisation in Sound Model	108
Figure 5.3.4	Data Augmentation Function in Sound Model	109
Figure 5.3.5	Final Dataset Size in Sound Model	109
Figure 5.3.6	Daatset Splitting in Sound Model	110
Figure 5.3.7	Dataset Distribution in Sound Model	110
Figure 5.3.8	CNN Block	110
Figure 5.3.9	BiLSTM Block	111
Figure 5.3.10	Attention Mechanism	111
Figure 5.3.11	Dense Later and Optimiser	111
Figure 5.3.12	Sound Model Training Loss and Accuracy Curves	112
Figure 5.3.13	Model Training in Sound Model	112
Figure 5.3.14	Sound Model Architecture	113
Figure 5.3.15	Classification Report in Sound Model	114
Figure 5.3.16	Confusion Matrix in Sound Model	114

Figure 5.3.17	ROC Curve in Sound Model	115
Figure 5.3.18	Precision-Recall Curve in Sound Model	116
Figure 5.3.19	Sound Model Classification Report for FYP1	118
Figure 5.3.20	Visual Dataset	119
Figure 5.3.21	Synthetic Data Generation in Visual Model	120
Figure 5.3.22	Target Dataset in Visual Model	120
Figure 5.3.23	Train Data Generator in Visual Model	120
Figure 5.3.24	Dataset Splitting in Visual Model	121
Figure 5.3.25	Splitting Result in Visual Model	121
Figure 5.3.26	Sequential with MobileNetV2 Base and Custom Head	122
Figure 5.3.27	Visual Model Architecture	122
Figure 5.3.28	Phase 1 and Phase 2 Training Blocks	123
Figure 5.3.29	Visual Model Training Loss Curve	123
Figure 5.3.30	Classification Report in Visual Model	124
Figure 5.3.31	Confusion Matrix in Visual Model	125
Figure 5.3.32	ROC Curve in Visual Model	126
Figure 5.3.33	Precision-Recall Curve in Visual Model	127
Figure 5.3.34	Parameter Size Comparison	129
Figure 5.3.35	Validation Loss Comparison	129
Figure 5.3.36	Classification Report for InceptionV3 Model	130
Figure 5.3.37	Threshold for Retrieving Records of Fine-Tuning	132
Figure 5.3.38	Freezing Strategy	133
Figure 5.3.39	Optimiser Initialisation	133
Figure 5.3.40	Training Personalised Model	134
Figure 5.3.41	Accuracy and Confidence Improvement Comparison	135
Figure 5.3.42	Learning Curve in Transfer Learning	136
Figure 5.3.43	Training Loss in Transfer Learning	136

LIST OF TABLES

Table Number	Title	Page
Table 1.5.1	Comparison of Problem Statement and Project Objective	8
Table 2.3.1	Functionality Comparison Between Preview Apps and Proposed App	33
Table 2.3.2	Proposed Solution to Address Weakness in MeowTalk	34
Table 3.2.1	Specifications of Laptop	36
Table 3.2.2	Specifications of Samsung Smartphone (Android)	36
Table 3.2.3	Software Architecture Description Table	37
Table 5.3.1	Model Comparison in Sound Model	117
Table 5.3.2	Model Comparison in Visual Model	128

LIST OF ABBREVIATIONS

<i>AI</i>	<i>Artificial Intelligence</i>
<i>CAGR</i>	<i>Compound Annual Growth Rate</i>
<i>MFCC</i>	<i>Mel-Frequency Cepstral Coefficient</i>
<i>CNN</i>	<i>Convolutional Neural Network</i>
<i>BiLSTM</i>	<i>Bidirectional Long Short-Term Memory Network</i>

CHAPTER 1: Project Background

1.1 Introduction

Nowadays, pet ownership has increased tremendously all around the globe. Pets are now being considered as members of the family since they can provide companionship, emotional support and measurable health benefits to their owners [1]. According to research, having a pet can reduce stress or symptoms of loneliness and enhance physical health by encouraging more exercise, which has strengthened the closeness between pet owners and their pets in recent years [2]. However, this growing trend brings the responsibility to ensure proper pet care, which many pet owners find challenging due to time constraints, lack of expertise or a failure to understand what their pets need.



Figure 1.1.1 Relationship Between Pets and Owners

To address these challenges, the pet care industry has witnessed the swift evolution of niche apps to offer cutting-edge features such as automated diet tracking, health monitoring and behavioural analysis. These applications provide solutions that enhance the progress of pet care. Take for example, an application that automates the handling of health maintenance by incorporating vaccination reminders, tracking medications and maintaining electronic health records have reduced missed treatments by 40% compared to manual methods [3]. Besides that, to reduce the waiting time for veterinary appointments, some pet care applications also facilitate community support through expert-moderated forums where owners have reported that it helps them increase their confidence in taking care of pets [4].

While these current pet care applications provide some valuable assistance to pet owners, they are nevertheless afflicted with some limitations, such as a great reliance on manual data input by pet owners and a lack of advanced functionalities that could indeed revolutionise pet care

[5]. Numerous applications only offer generic advice rather than tailored recommendations based on a pet's specific needs and limited effective communication tools for decoding pet behaviours. This incoherent strategy overlooks custom requirements and tailored solutions for different pets. The absence of such sophisticated analytical capabilities makes these apps unable to interpret or predict the real requirements of a pet. Therefore, this project has integrated advanced technologies in computer vision and machine learning for better solutions.

Through the utilisation of image recognition technology, automated food identification can replace the traditional manual entry process. Instead of prompting owners to manually enter specific food brands, ingredients or serving sizes, the AI-based image recognition can simply identify the pet food from an image. Pet food can be photographed by pet owners and AI will automatically examine its nutritional content as well as suggest a healthier alternative or the best cooking method suitable for pets. This eliminates the need for owners to read labels, measure portions or remember to log meals. In addition, for medication tracking, image recognition technology can detect the health issues of a pet's external body symptoms or waste to provide early detection and reduce the need for guesswork. This symptom recognition allows owners to recognise concerning conditions such as rashes and injuries. AI-powered techniques automatically create the documentation and timestamp these observations. It provides a comprehensive health timeline without requiring detailed written descriptions.



Figure 1.1.2 Nutrition Analysis

Furthermore, by implementing machine learning pet care applications, a tailored pet translator can be created to drastically improve the precision of decoding animal communication. As supervised learning models are trained using labelled data, this increases the accuracy at the starting point [18]. Transfer learning models can analyse audio recordings of pet vocalisations alongside corresponding owner feedback to establish correlations between specific emotions. As each pet has idiosyncratic ways of expressing hunger, discomfort, playfulness or affection

[6], this translator can adapt accordingly through continuous learning. Meanwhile, deep learning allows the system to learn from unlabelled audio data, reducing the need for extensive manual labelling [19]. After a duration of utilisation time, this translator will become increasingly personalised over time by emphasising the unique communication patterns of individual pets as opposed to generic species templates. These breakthroughs not only raise the quality of pet care but also deepen the bond between pets and their owners by making interaction more informed.

Additionally, this project provides several intelligent functions that include a wide variety of different functional components that improve the overall experience of pet owners providing care for their pets. One of the features is a calendar-based reminder system that will assist pet owners in tracking important activities such as vaccinations, grooming schedules and medications through automated notifications. Moreover, this project has also implemented an expense tracking module that allows a pet owner to take a picture of purchase receipt with their mobile device and have the expense categorised and coded based on category. Finally, a community platform is incorporated to allow pet owners to make price comparisons, share posts, exchange experiences and seek advice from other pet owners, creating a caring and supportive community.

The next generation of pet care applications will be about deploying the powers of AI to provide proactive, personalised and helpful assistance. Moving beyond basic tracking abilities and utilising intelligent technology such as image and machine learning, this project endeavours to develop an entirely new kind of app that not only facilitates easy pet care but also further develops the emotional relationship between owners and pets. The integration of AI into pet care is not just a technological leap but a meaningful step forward for more compassionate pet ownership.

1.2 Problem Statement

Recently, more and more pet owners are utilising pet care applications to assist them in taking care of their pets. However, despite having various kinds of existing pet care applications, many fail to deliver a seamless user experience due to some critical limitations. Therefore, four major key problem statements are identified below.

i. Highly Dependent on Manual User Input

Numerous existing pet care applications that over-rely on manual data entry have placed excessive demands on pet owners, increasing both error rates and owner burden. For example, to obtain an accurate analysis in nutrition tracking, pet owners will need to spend considerable time weighing exactly each portion of food to the gram and manually classify food types and log every ingredient in homemade meals. As a result, this may be time-consuming for owners. Even a single mistaken entry can lead to inaccurate diet recommendations. Seriously, it will mask nascent health issues like obesity or deficiencies. Besides that, for pets' health detection, many applications rely on text-based descriptions rather than AI-assisted image recognition. However, some symptoms are hard to recognise and describe in words. For instance, subjective text descriptions like “red skin” fail to capture symptom severity or progression. Thus, most of the pet owners may struggle to explain symptoms clearly, leading to misdiagnosis or delays in treatment for conditions like rashes, skin infections or abnormal growths.



Figure 1.2.1 Problem of Manual Input

ii. Inadequate Pet Emotion Interpretation Tool

In the modern era, many pet owners are struggling to understand their pets' emotion. Although some applications have already developed interpretation tools to decode these vocalisations, their effectiveness remains limited due to low accuracy rates and lack of adaptability. Most of the interpretation tools rely on generic sound databases that fail to account for the significant variation in vocalisations across individual pets. As an example, a cat's “stress meow” may be considerably different based on its size, environment or personality. Research shows that these one-size-fits-all models misinterpret pet emotion up to 50% of the time [7]. Additionally, current pet emotion interpretation tools also interpret pet emotion without considering their environment. However, the same sounds can have

different meanings depending on the situation. This contextual blindness leads to frequent misinterpretations and always confuses owners more than it aids them.



Figure 1.2.2 Misunderstanding Pet Emotion

iii. Lack of Personalisation and Absence of Automated Documentation

In this day and age, current pet care applications fall short in delivering personalised care and automated health tracking. Most existing applications rely heavily on broad recommendations that fail to account for individual pet needs. This limitation stems from an over-reliance on manual owner input which frequently leads to incomplete and inconsistent data. Without AI-driven automation, current applications cannot offer tailored insight for nutrition analysis, behaviour analysis and health detection. Moreover, lack of automated documentation gives rise to vast inefficiencies in pet medical care. Pet owners are forced to maintain unordered and manual health histories which often lack critical details needed for accurate veterinary diagnosis. For example, without automated image logging, owners struggle to accurately describe skin symptoms and the veterinarian has a hard time tracking the pet's symptom progression. Consequently, this incomplete data causes delayed diagnosis, prolonging identification and treatment of skin infections.

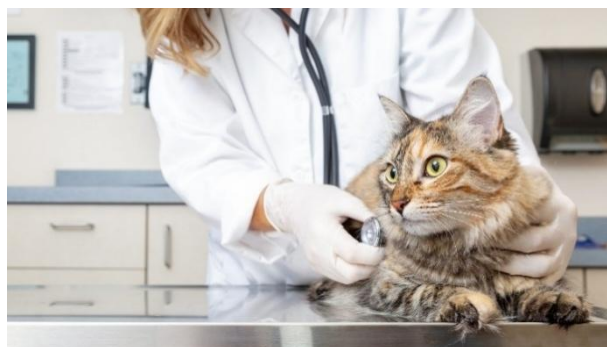


Figure 1.2.3 Personalised Care

iv. **Lack of Integrated Pet Activity Management and Reminder System**

Many pet owners do not have a concrete plan for scheduling important events regarding their pets. They often rely on memory or external tools to keep track of pet-related activities such as vaccinations and medical appointments. Unfortunately, this approach is often unreliable and can result in missing or being late on getting treatments. Studies have shown that ineffective reminder systems can cause incomplete vaccination schedules and missed appointments, increasing health risks for pets [26]. A gap between the intended and actual behaviours of pet owners with respect to preventing illness is present among veterinarians that do not provide an adequate reminder system [27]. In addition, pet owners may face challenges in managing expenses and identifying cost-effective products due to the absence of price comparison tools. The lack of a pet care community platform limits opportunities for users to seek advice, making it more difficult to make informed decisions regarding pet care.

1.3 Motivation

In today's fast-paced era, many pet owners are occupied with their daily routines such as jobs and education and it is difficult for them to closely monitor their pets' diets, health and habits continuously. Thus, they often rely on traditional methods such as manual symptom inspection or estimation, which are time-taking, error-prone and less tailored. This may lead to unseen health issues and nutritional disorder in pets' needs which will affect pets' well-being. Also, most of the existing pet care apps have common drawbacks such as minimal support for pet emotion interpretation with very limited reliable channels left for owners to gain a clearer understanding of their pets. Thus, according to the issues, there seems to be a clear need for an intelligent app that can offer pet owners data-driven care. Therefore, this project proposes the development of an AI-powered pet care application that uses image recognition and machine learning to provide pre-detection of health, nutritional analysis and translation of pet emotion into human-understandable information. This can reduce the burden on pet owners, strengthens the human-pet bond and improves pets' general quality of life.

1.4 Project Objectives

In order to solve the problem statement above, the project aims to simplify the process of caring for pets by applying image recognition and a machine learning approach (Supervised Learning, Deep Learning, Transfer Learning), while also strengthening the connection between pets and their owners. This application is designed to provide users with a complete package of features which includes nutrition analysis for both barcode-labelled and non-barcode foods, health analysis based on external body symptoms and waste detection, health report generation and pet emotion interpretation. This sophisticated application of image recognition and machine learning algorithms will enhance efficiency, reduce the error and maximise the personalisation factor.

There are 4 sub-objectives that derive from the main objectives:

i. Simplify the analysis process for nutrition and health

The project implements advanced image recognition techniques to automate pet health and nutrition analysis. This includes scanning both barcode and non-barcode food for nutrition information, detecting external body symptoms and analysing waste conditions for comprehensive health analysis. It also automatically creates the health report according to the timeline, thereby improving their understanding and connection with their pet. By minimising the amount of manual data entry, the app can save time for users and remove the possibility of human error. Hence, it simplifies the pet care process to be less burdensome to execute.

ii. Improve accuracy and personalisation of pet emotion interpretation for individual pets

By using an approach of machine learning through supervised learning, deep learning and transfer learning, the app is able to enhance its performance on its own. This not only boosts the accurate interpretation of pet emotion into human-understandable language, but also expands the scope of vocalisation recognition. This is because it can learn from the unknown sound structures and patterns. At the same time, the app adapts to each pet's unique individual behaviours, enabling increasingly tailored emotion interpretation. Consequently, the app can reduce interpretation error and provide customised pet care support to users.

iii. **Strengthen safety monitoring**

The application integrates a smart alert feature that is useful in recognising dangerous or allergy food through image recognition. When it detects dangerous food, the app instantly sends a warning message to the user in order to allow them to take immediate action, protecting their pet's health. This proactive functionality introduces a safety aspect into pet care by avoiding accidental feeding. As a result, it helps to strengthen user confidence in the application.

iv. **Developed an all-in-one integrated smart pet care application**

This app aims to develop a unified pet care platform that integrates nutrition analysis and health analysis, pet emotion interpretation and activity management into a single application. The app includes some essential features such as reminder functions, expense tracker and community platform to enhance user experience by reducing the need to switch between several apps. By combining all these features into an intelligent system, the app reduces the need for multiple tools, improves user convenience and provides a comprehensive digital solution for managing all aspects of pet care effectively.

Overall, these features allow users to take better care of their pet's health and foster a stronger bond with pets.

Table 1.5.1 Comparison of Problem Statement and Project Objectives

Problem Statement	Project Objective
Highly Dependent on Manual Input	Simplify Analysis Process for Nutrition and Health
Inadequate Pet Emotion Interpretation Tools	Improve Accuracy and Personalisation of Emotion Interpretation
Lack of Personalisation and Absence of Automated Documentation	Strengthen Food Safety Monitoring for Each Individual Pet (Generate Personalised Analysis & Auto-generate Health Reports)
Lack of Integrated Pet Activity Management and Reminder System	Develop an all-in-one integrated smart pet care application with reminders, tracking, and activity management features

1.5 Project Scope

The final deliverable of the project is a mobile application-based pet care assistance application that uses image recognition and machine learning to improve pet health monitoring and owner–pet engagement. The system combines five main functions. First, the food nutrition analysis function allows for the identification of pet foods that contain either barcodes or non-barcodes and provides information about the food such as food ingredients, suitability percentage, nutritional facts and safe cooking methods or alternate food brands. If harmful ingredients are present, safety warnings will show up. Second, the health analysis function facilitates image-based detection of external symptoms, injuries, abnormal waste or body weight. This function provides the following insights that include signs of potential health concerns, the percent severity level, potential health conditions and personalised recommendations. In addition, the app provides an automated health report download that can be helpful for veterinarians, as there will be a reduction in handwriting and manually entering symptoms. Third, the emotion interpretation function will utilise supervised, deep and transfer learning to analyse the sound and body language made by pets and eventually convert the emotion into human-understandable insight such as angry, happy and hungry. The function includes continuous learning, whereby user feedback can be submitted to improve the overall accuracy of results and responses to each pet. Fourth, a pet activity management system which includes a calendar-based reminder system that allows users to record vaccination date, grooming appointments and other pet-related activities. It will generate automated reminders and push notifications to ensure timely pet care management. Besides, an expense tracking feature was developed to allow users to make wise money management. Last, a community platform that enables pet owners to make price comparisons, gain knowledge or share questions among their pets.

The app is primarily designed for pet owners, particularly those who are new to pet ownership and busy owners who may not know or have the time to manage their pets' daily health needs. This app also can be a complementary resource for veterinarians by providing structured health data for consultation purposes. However, because of time and resource constraints, it is only feasible to consider cats. This narrowed scope also relates to the fact that the dataset required to train and fine-tune the model to improve performance is difficult to obtain. Besides that, the app will be developed only in Android with the hopes to maximize accessibility while further narrowing the technical development scope. Overall, the end result of the project is a fully

functional mobile application that provides pet owners with AI-based insights, improved communication with veterinarians and increased responsible and proactive pet care.

1.6 Impact, Significance and Contribution

Since available solutions within pet care also often possess well-known shortcomings like high reliance on user interaction, lack of personalisation and inadequate emotion interpretation, this project aims to provide users with a smart and user-friendly application that reduces pet care effort. The impact of this application is to change how pet owners monitor and care for their pets through providing precise and convenient information about nutrition, health conditions and emotional needs. By detecting illness earlier and better supporting nutritional balance and personalised care strategies, the app fosters healthier pets and creates positive relationships between pets and their owners. It can also help eliminate unnecessary veterinarian costs for insignificant problems and decrease the total costs per veteran while allowing veterans more time to spend on significant or needed issues.

Hence, this application addresses the limitation with multiple features. For instance, through food nutrition analysis via image recognition, owners can scan homemade or commercial food to assess nutritional value, identify toxic ingredients and find recommendations for healthier alternatives or portion adjustment. To help monitor health, the app will enable users to take and upload their cat's stool or skin which will be analysed for possible health issues such as rashes, gastrointestinal disease or parasites and generate a health report to help with consults at the veterinarian. Additionally, the pet emotion interpretation feature allows owners to record the videos of their pets and the app determines the emotions or needs such as happiness or anger by applying machine learning. With the use of transfer learning, the app improves with accuracy over time, becoming increasingly personalised to the individual pet. Further, the extended functionalities including a pet activity calendar, expense tracking feature and community platform, enhance the overall usability and practicality of the app. This helps users in organising their pet's routine, tracking financial and providing opportunities for knowledge sharing within the pet owner community. In brief, this project provides a complete pet assistant that not only provides healthier pets but also eases the workload of the owners, simplifies communication with vets and enhances the pet-owner relationship.

CHAPTER 2: Literature Review

2.1 Review of Existing Application

Pet care applications nowadays are designed to assist owners in tracking and maintaining the health, nutrition and well-being of their pet. With the rapid growth of artificial intelligence and mobile technologies, most pet owners now depend on using intelligent digital tools to help them with everyday pet care. AI has indeed shown incredible results in human healthcare by performing tasks like diagnosis, patient monitoring and even complex surgeries which at times were endowed with similar or even better expertise than human professionals [8]. On the other hand, the global pet care market valued at USD 150.67 billion in 2021, is growing at a 5.1% CAGR driven by rising pet humanisation and consumption [9]. With individuals keeping more small pets and spending on high-quality care, the need for intelligent AI-based pet care apps is growing at a very fast pace. Therefore, by bringing these advanced technologies into pet care detection, applications can now offer features such as nutritional analysis, health condition detection and emotional or behavioural interpretation. All these help pet owners to gain a better understanding of their pets' needs, detect potential health problems early and take improved care decisions. In addition, many applications also include extended features such as activity calendars, expense tracking systems and community platforms to further support pet care management.

2.1.1 Pawdi – Pet Food Scanner

Pawdi is an innovative AI-powered mobile application developed for assisting pet owners in evaluating and selecting healthier food options for their pets. By using advanced image recognition technologies, Pawdi instantly analyses pet food products and provides clear information regarding their nutritional value in real-time. This app is intended to create convenience and precision by providing a new solution to owners who desire to make the healthiest diet for their pets. With its fast analysis and user-friendly interface, Pawdi stands out among the growing number of pet care applications available today.

First and foremost, one of the key strengths of Pawdi is its ability to scan barcodes of pet food and immediately deliver a comprehensive nutritional analysis [10]. After scanning, users will receive a clear nutritional score, a description of the food, detailed nutritional information including vitamins and minerals and enhanced ingredient analysis. Other than this, the app even outlines the pros and cons of the pet **food** that users scan which helps users make informed

decisions within seconds. Compared to other existing apps such as Pet Food Expert that offer basic or limited data [11], Pawdi's detailed breakdown ensures pet owners truly understand what they are feeding their pet.

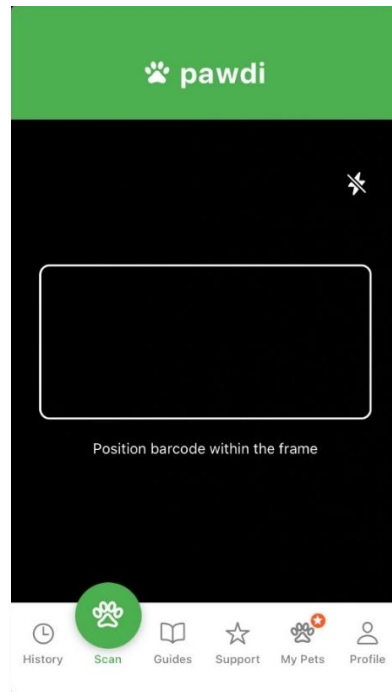


Figure 2.1.1 Scanning Barcode Functionality in Pawdi

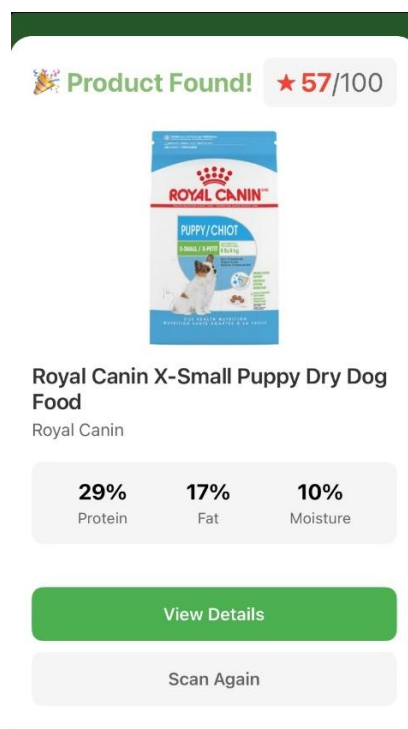


Figure 2.1.2 Overall Nutrition Analysis in Pawdi

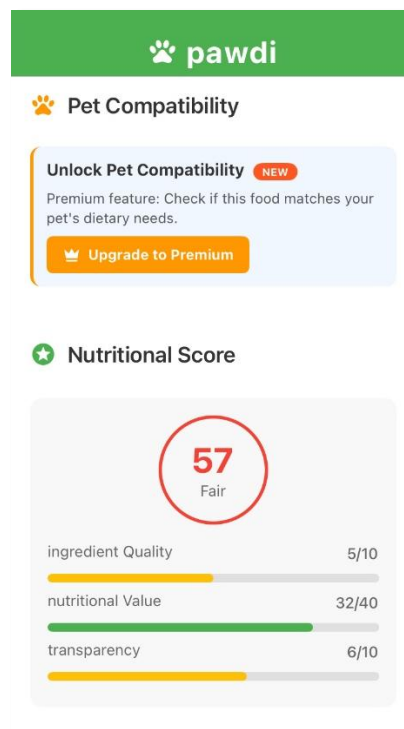


Figure 2.1.3 Nutrition Score Evaluation in Pawdi

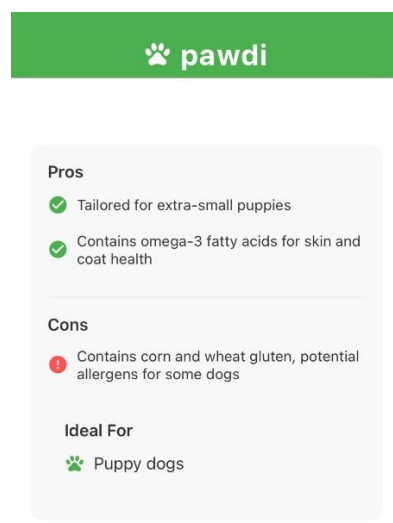


Figure 2.1.4 Pros and Cons Description in Pawdi

In addition to on-the-spot analysis, Pawdi allows users to view their entire history of scanned pet food products, making it easy for users to revisit and review their past choices. This feature is important to enable pet owners to track the nutritional quality of food they have considered over time and make better-informed comparisons when selecting products [13]. Besides that, the app also provides personalised food recommendations based on the pet's breed, body weight and overall health [12]. Its easy-to-use design clearly marks important details, ensuring

users can quickly interpret the result without confusion. This seamless and intuitive experience differentiates Pawdi from many traditional pet care apps. As an example, users that use traditional pet care apps require time-consuming manual input of a lot of information and finally just get generic advice from apps.

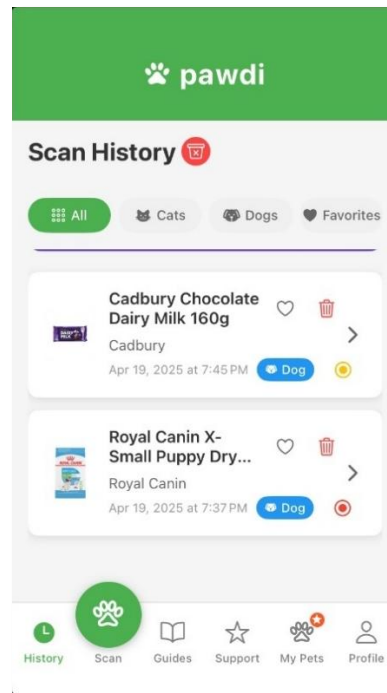


Figure 2.1.5 History Review Functionality in Pawdi

In short, Pawdi combines the power of AI with a strong focus on user experience by providing pet owners a fast and reliable way to manage their pets' nutrition. Its clear nutritional breakdowns, scanning history feature, personalised recommendations and simple interface make it a highly preferred choice for those who want an easier and smarter way to take care of their pets' dietary needs.

2.1.2 TTcare – Pet Health Detector

TTcare is an innovative pet healthcare app that empowers pet owners to scan their pet using photos to detect early signs of diseases without the need for any expensive or special equipment [14]. The users have been advantaged through the app because it has enabled them to detect pets' health issues in the early stages before they develop into serious conditions. In addition to advancing pet care technology, it is assisting in creating pet healthcare regulations and has received a 2-year special exemption from the South Korean government under a regulatory 'sandbox', with which it can offer online veterinary consultations despite current legal

prohibition. The innovative work of this app has made it the recipient of CES 2022 and 2023 Innovation Awards and has been featured in well-known publications like Forbes Asia. It has established its reputation as a pet healthcare future leader [15].

Therefore, compared to other existing applications, one of the strong features of TTcare is that it is able to provide early detection at a higher accuracy rate which is 90%, as it is backed by patents and veterinary research [16]. By simplifying the uploading of photos of pets' skin, eyes or other body parts, pet owners can get health insights about their pets. This app provides pets with a better chance for quicker recovery and avoiding costly treatments. As shown in research, early diagnosis can significantly improve the effectiveness of medical interventions [17], which TTcare enables early detection by analysing the visual symptoms of pets in a precise manner.

Furthermore, another key function of TTcare is to provide automated health summaries [16]. This feature has simplified pet health management for owners. It can automatically compile data about the pet's medical history, symptoms and health metrics to create a comprehensive summary that can be shared with veterinarians. Hence, this app reduces the need for manual tracking and data entry. Moreover, TTcare is also designed with simplicity in mind and is easy to navigate. For instance, pet owners who are not familiar with high-tech applications are able to have quick access to its features without spending most of their time struggling with complex tools.

In conclusion, TTcare is a complete pet care application that allows pet owners to stay proactive about their pets' health. Due to early disease detection, automated health summaries and simple design, TTcare is a valuable feature of contemporary pet care. It assists in improving a healthier life for pets and gives owners peace of mind.

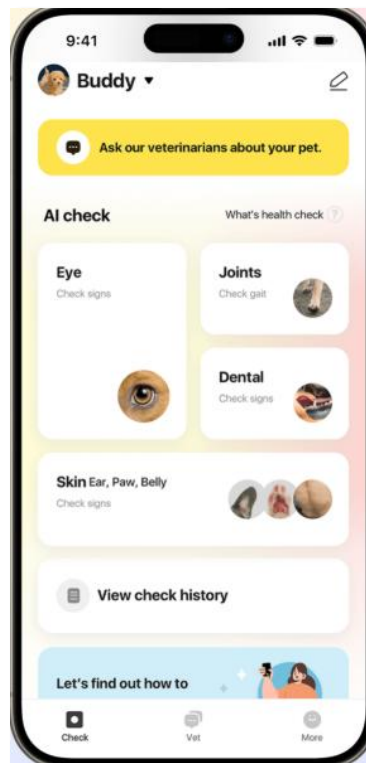


Figure 2.1.6 User Interface in TTcare



Figure 2.1.7 Scanning Functionality in TTcare

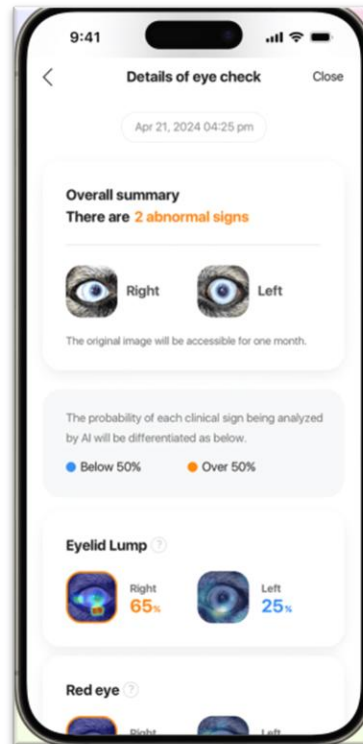


Figure 2.1.8 Health Analysis in TTcare

2.1.3 MeowTalk – Pet Sound Interpretation Tool

Research shows that pets use vocalisations to communicate information, thus their sounds can carry a wide range of meanings [21]. The creation of MeowTalk became a useful tool for pet owners to understand their pets' needs and enhance the longest and happiest pets' lives [20]. MeowTalk is a groundbreaking cat translator mobile application that uses smart AI to bridge the communication gap between domestic cats and their owners. It aims to provide a greater understanding of pets' emotional lives and needs such as hunger, discomfort or affection.

One of the key functions of MeowTalk is that it is able to provide accurate translation to pet owners. Pet owners can record their cat's sound and this app will offer instant translations of cat meows into emotions without needing to wait for a long period. As this app is applying supervised learning training, MeowTalk has collected and trained a set of labelled data which enhances the initial user experience with a higher starting accuracy rate of around 90% compared to other existing apps [22].

Besides that, through reinforcement learning, MeowTalk provides personalisation to each unique pet. It can continuously improve its recognition capacity by learning from user feedback. As an example, if the app cannot automatically recognise the meow sound, users can manually

record a voice sample and select the translation they believe is correct. By submitting the sample, users help the app adjust the interpretation within 24 hours [23]. This feedback loop allows MeowTalk to get increasingly personalised over time.

To sum up, MeowTalk offers an innovative solution to the challenge of understanding feline communication. By employing supervised learning and reinforcement learning, the app not only translates sound into information that can be understood but also becomes more accurate over time. Thus, this app helps pet owners connect with their pets on a deeper level.

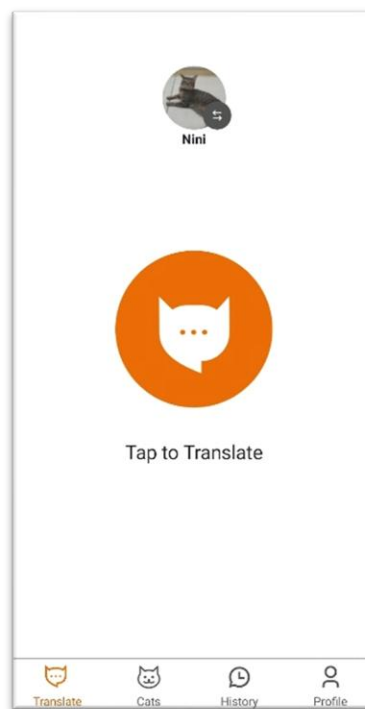


Figure 2.1.9 User Interface in MeowTalk

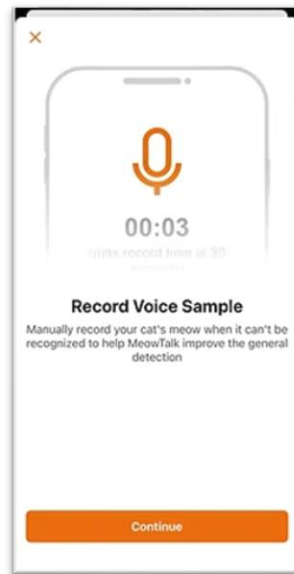


Figure 2.1.10 Manually Recording Unfamiliar Voice in MeowTalk

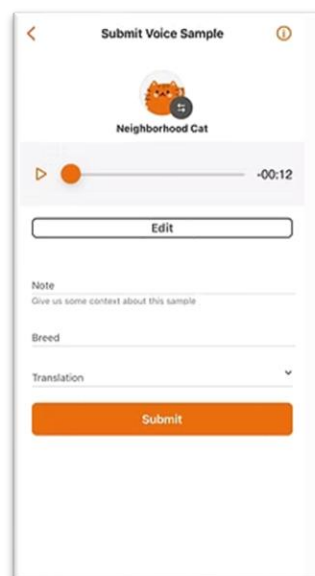


Figure 2.1.11 Manually Entering Translation for Unfamiliar Voice in MeowTalk

2.1.4 11Pets – Pet Care Management Application

11Pets is a pet care management application that helps pet owners organise and maintain essential pet-related information in a structured digital system. One of its key features is the vaccination, deworming and medication management system which allows users to store detailed records of vaccination and deworming schedules. The application automatically generates upcoming schedules and sends timely reminders when the next action is due, helping owners ensure that important health routines are not missed. Users can also enter medication details such as dosage, frequency and administration timing, allowing the app to provide

consistent notifications for each required dose. In addition, the app supports calendar synchronisation, which allows all pet care schedules to be seamlessly integrated with the user's device calendar [28]. This ensures pet care activities always remain well-organised and up to date.

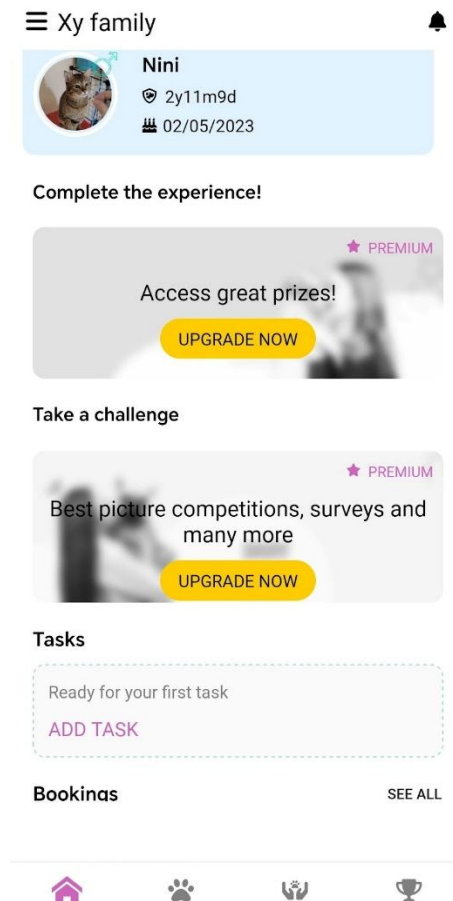


Figure 2.1.12 Home Page in 11Pets

< Nini

Tasks

Category
Medical

Sub-Category
Internal Deworming

Date
11/04/2026

Time
8:20 pm

Repeat Every
1

Every
Month

Repeat On
Monthly on the Second Saturday

Ends
On Occurrences Never

Date
11/05/2026

Notification
Enable Notification

before
10 Minutes

Add note

Figure 2.1.13 Management System in 11Pets

< Nini +

Tasks

All Hygiene Cleaning Activities

Custom range Filter

11 Apr 2026

Internal Deworming 20:20 pm

09 May 2026

Internal Deworming 20:20 pm

Figure 2.1.14 Auto-generate Schedule in 11Pets

Besides that, another important feature is the medical record management system, which enables sixers to store comprehensive pet health information into a single platform [28]. This includes medical documents such as X-rays, laboratory test results, blood reports, diagnosed medical conditions, allergies, medications and surgical history [28]. Users can also attach notes and photos to better document their pet's health condition over time. Further, this app enables users to securely share complete medical history with veterinarians or specialists when necessary, supporting more accurate diagnosis, better treatment planning and improved continuity care.

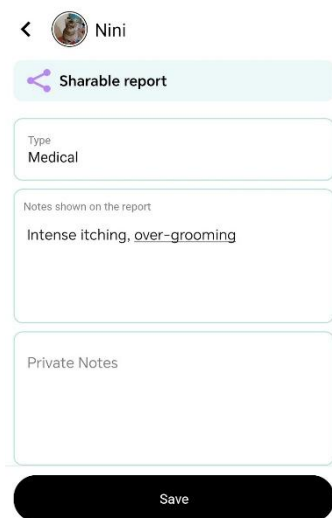


Figure 2.1.15 Shareable Report in 11Pets

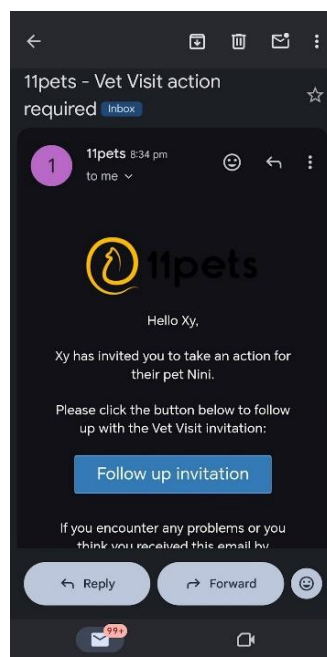


Figure 2.1.16 Invitation Link in 11Pets

Lastly, 11Pets is also beginner-friendly, as it provides guidance videos to assist users in understanding pet care practices. Videos such as “Common Mistakes Pet Owners Make”, are designed to educate beginner pet owners by highlighting typical errors and providing basic care knowledge. Such content helps users improve their understanding of proper pet care practices and supports them in managing their pet more effectively.

2.2 Review of Previous Works on Interpreting Pet Emotion

2.2.1 Domestic Cat Sound Classification Using Learned Features from Deep Neural Nets

Pandeya et al. [39] investigated domestic cat sound classification employing deep learning methods. The study is inspired by the fact that cats are popular pets and have various vocalizations that depend on mood and situation. Interpretation of these sounds allows better communication between cats and owners, who in turn can better meet their needs. However, conventional methods are less practical for ordinary pet owners because it required manual analysis of animal behaviour experts.

To address this, the study constructed a dataset called CatSound, containing 10 categories of cat vocalisations such as happy, angry, resting, warning, fighting, hunting, mating, mother call and pain. This dataset was created and labeled by domain experts to ensure quality. Due to the small size of the dataset, the authors adopted a number of audio data augmentation techniques, such as time-shifting and noise insertion, for artificially enlarging the training set and enhancing the ability of generalisation of the model.

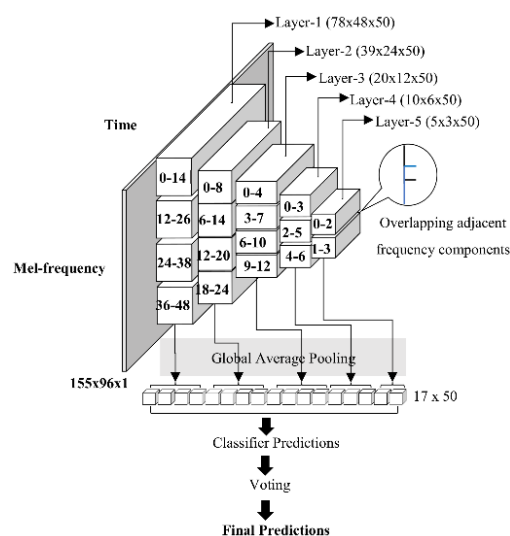


Figure 2.2.1 Deep Neural Net Model Architecture Diagram

For feature extraction and classification, two kinds of deep learning methods are investigated in this study. The first was a pre-trained CNN model trained on a large music dataset, which was used as a feature extractor by means of transfer learning to the representation of cat sounds. The other was a Convolutional Deep Belief Network (CDBN) in an unsupervised manner which was learned from the recorded cat sounds themselves. On top of these feature extractors, the authors introduced a novel pooling scheme named Frequency Domain Average Pooling (FDAP), claiming that more meaningful frequency-level features can be obtained compared with the traditional Global Average Pooling (GAP) method.

The classification at the output layer demonstrates that the features extracted from the pre-trained CNN by transfer learning achieve comparable accuracy on the CatSound dataset, which is better than several baseline methods. The CDBN features yielded also quite good results, especially when combined with the proposed FDAP pooling. The experiments showed that deep learning can be used to classify cat sounds with success, even if the labelled data at hand is very limited.

2.2.2 DeepCat: A Deep Learning Approach to Understand Cat's Body Language

Fawzy et al. [40] propose a deep learning framework that enables cat owners to gain insights into their feline's body language and emotional state. Motivated by the fact that the pet owner has difficulty interpreting the non-verbal cues of the cat such as their posture, tail position and facial expressions. Compared to dogs, cats are more secretive in showing how they feel and it is even more difficult for their owners to tell when they feel anxious, uncomfortable or joyful. Because animals are unable to take the advantage of human language to express their conditions, an intelligent instrument is in urgent demand to fill the gap in animal-human communications.

To construct the model, the authors collected a dataset of 10,000 cat images and ran an automatic labelling for a number of their body features such as the tail, eyes and mouth positions. The labelling was done by using LabelImg, a graphical image annotation tool which can be used to draw bounding boxes around the object to be detected. Subsequently, the YOLOv8 object detection model was trained on this annotated data to recognize the three body parts and identify their location in real-time. The choice of YOLOv8 was motivated by its speed, being able to run 1 image in 0.007 seconds and 140fps in video mode, making it ideal for real time applications on mobile devices.

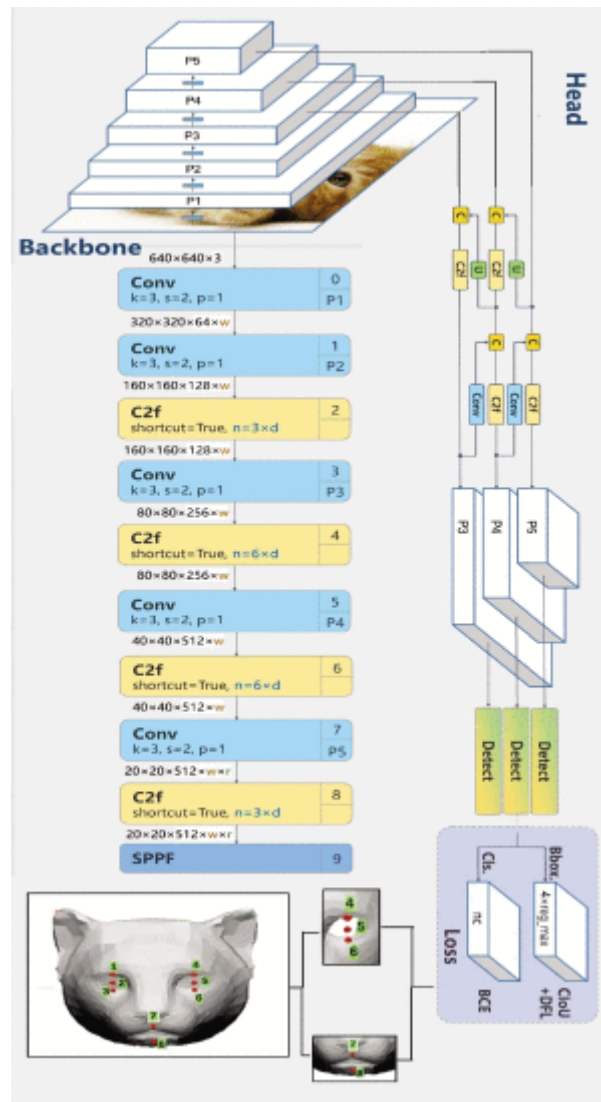


**Figure 2.2.2 Preprocessing Procedures Before Implementing
DeepCat Algorithms**

The model training and inference processes consist of three stages, which are all responsible for generating the resulting emotion prediction. In the first step, the fine-tuned YOLOv8 model predicts the bounding boxes of the eyes, mouth and tail within each frame of the input image or video. In the second step, data from each detected particle of the body is analysed separately by using rule-based geometric formulations that built upon the geometric shape of the body part in question. To determine if the eyes are open wide or closed, the system calculates the ratio of white to dark regions of pixels in the eyes. For the mouth, a state information such as open or close, is associated with each frame and the state that appears most frequently in the whole video sequence is output as the final result for the mouth. For the tail, the detected area is surrounded with a contour and the vertical-length of the contour is

compared with the horizontal-length to determine left or right tail orientation and then calculating angle through trigonometric functions to tag the tail as raised or dropped or curled. Lastly, a voting classifier is used to combine the prediction of the three body parts in the third stage, which outputs the most frequently predicted emotion as the final estimate. This voting scheme is also very beneficial for the video case, since the posture of the cat might change from one frame to the next.

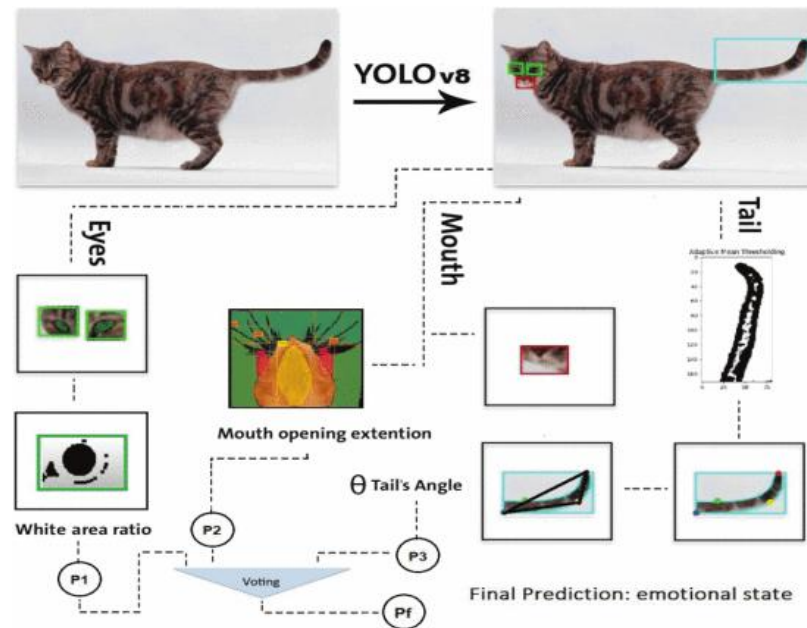


Figure 2.2.3 Overview of the DeepCat Algorithms with Three Selected Features

The entire solution is wrapped in a Flutter mobile app, allowing deployment on both android and ios devices with live camera feed. The report was awarded the third prize at the OpenCV AI Competition January 2023 and was presented at the 2023 International Japan-Africa Conference on Electronics, Communications and Computations. The authors themselves pointed out under their future work plans that they would investigate transformer based or LSTM-enhanced CNN models for better accuracy. They also specifically named cat voice analysis as a planned extension of the work to be able to perform visual analysis with acoustic signals.

2.3 Limitation and Proposed Solution

2.3.1 Limitation of Existing App

In spite of the notable progress achieved by existing pet care applications that have been mentioned above (Pawdi, TTcare, MeowTalk, 11Pets), there are several limitations. One of the biggest limitations that has been faced by these three applications is a narrow focus and lack of integrated features. Current apps in use today are focused primarily on one aspect of pet maintenance. For example, Pawdi focuses primarily on pet food nutrition, TTcare focuses on identifying apparent body symptoms, Meowtalk focuses on cat emotion interpretation, while 11Pets mainly focuses on pet care management such as scheduling and medical record tracking. This is due to the lack of full functionality and inconvenience. Pet owners are forced to download and switch between multiple apps to cover different aspects of pet care.

Besides the weakness of the narrow focus, each app had also met specific limitations that reduced overall user experience. For Pawdi, the limitation of scanning scope became an obvious weakness. Apart from scanning pet food products via barcode, this app does not support non-barcode items such as homemade meals or custom diets, leaving a gap in its coverage. Moreover, Pawdi is also not suitable for beginner pet owners. Although it has provided a clear detailed nutrition analysis to pet owners, the app assumes that pet owners already have some prior knowledge of pet nutrition. A comprehensive alert system not being implemented makes the app weak at providing awareness to pet owners. Critical alerts for harmful ingredients are subtle and easy to miss. For example, if a dog owner scans a chocolate barcode, Pawdi fails to provide a clear warning that chocolate is harmful to dogs. Instead, it places a brief description at the bottom of the screen which can easily be overlooked. This limitation potentially puts pets at risk if users don't notice these important alerts.

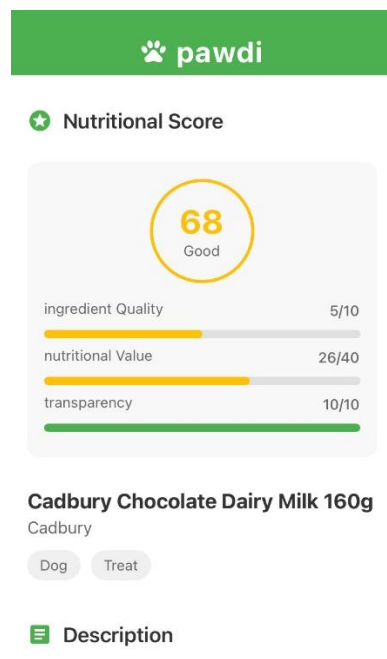


Figure 2.3.1 Moderate Nutritional Score of Chocolate Although It Is Unsuitable for Pet

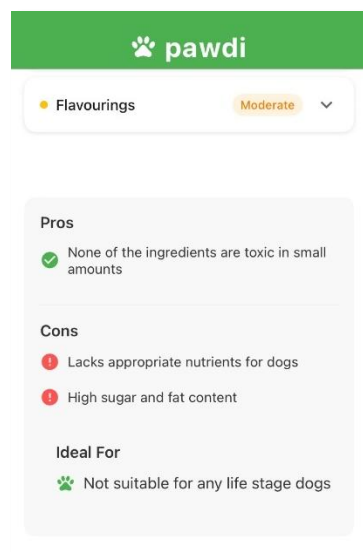


Figure 2.3.2 Unsuitable Description That Easily Be Overlooked by Users

While TTcare provides a useful scanner for detecting visible external symptoms like skin infections and eye diseases, it has limitations in scope when it comes to broader health detection. This app primarily focuses on certain body-related symptoms like skin issues, eye infections or ear problems, but may not recognise more complex or internal conditions that lack clear visual signs. Other than this, TTcare's detection accuracy heavily depends on photo quality. Sometimes, the slight symptoms may not be easily captured and analysed. External factors like

poor lighting or blurry images also will affect the accuracy of analysis, reducing the reliability of results.

In addition, for MeowTalk, one of the weaknesses is the limitation of recognition variety. Since it primarily utilises supervised learning, it will struggle in interpreting rare or unique vocalisations that are unlabelled and outside of common emotional expressions. Although MeowTalk attempts to solve this problem by integrating simple reinforcement learning, it is still not a better approach to provide a user-friendly experience. This is because it relies heavily on manual feedback. To improve personalisation, pet owners have to manually record and label the sounds of their pets. This can lead to a slower learning process. As a result, MeowTalk is not able to provide real-time translation for unfamiliar sounds. For example, if a meow is unrecognised, pet owners must manually submit it and the app may take up to 24 hours to process and adjust the interpretation [23].

Furthermore, 11Pets also presents several weaknesses as a traditional pet care management system. Although it provides useful features such as vaccination schedules and medical record storage, it mainly functions as a manual record-keeping application. Thus, one of its key limitations is it relies heavily on users inputting data manually. Pet owners are required to perform all the record-keeping and pet care tasks themselves. This increases user workload and may result in incomplete or inaccurate records over time. In addition, 11Pets does not incorporate artificial intelligence capabilities. As a result, it lacks intelligent features such as image-based health analysis, automated nutrition assessment and behavioural interpretation. Moreover, although the app provides guidance tutorials, the content is limited in depth and scope, which may not effectively address all user problems. This may make it hard for older users or users that need more detail to understand information to find information they are looking for.

In short, although previous studies have provided useful functions for pet owners, they still struggle with some challenges. To enhance pet care management and user experience, all these issues should be addressed through the solution proposed in 2.3 section.

2.3.2 Limitation of Previous Studies

Although the studies reviewed in Section 2.2 significantly advance the study of cat emotion via deep learning, they are hindered by limitations that prevent a full-scale applicability in real-world and long-term care for a pet.

In Pandeya et al [39]., the audio classification model is trained one time on the overall CatSound dataset and is entirely fixed after the training phase. By default, the model classifies all cats and makes no identification for an individual cat. However, every cat has a unique voice such as pitch, duration, intensity and their voices are influenced by their breed, age and character. Such a model will always hit an accuracy ceiling when used for any particular pet, as it has never seen that animal's particular vocal patterns.

Furthermore, in the case of DeepCat[40], the system is centered on visual body language analysis exclusively and does not use any audio signal. This is quite a big hole because the best way to assess a cat's emotional state is to simultaneously consider the pet's vocalisation. A cat can be vocalising in distress with a relaxed stance or it can bring its tail up and emit a happy noise without sound. In addition, the rule-based geometric method employed to examine the eyes, mouth and tail is quite sensitive to changes in camera angles, illumination, image resolution and the shape and size of the subject's body when applied on varied cat breeds, which may lead to diverse results. Besides, similar as Pandeya et al. [39], DeepCat learns a fixed general model and uses the same weights for all cats without any capability of personalisation.

In conclude, the two main limitations that both studies suffer from are no multimodal analysis that integrates audio and visual signals for joint prediction, and no personalisation approach to fine-tune the model to a particular individual pet using real-world feedback from the owner. These two gaps are the basis upon which the offered application in this project is developed.

2.3.3 Proposed Solution

The present pet care mobile app is designed to bring an accessible environment by providing an integrated system of functions to fill in the gaps of existing applications. Rather than concentrating on one aspect of pet care, this app will bring together a number of essential pet-care functions in one place.

In addition to obtaining nutrition analysis for food products by scanning commercial food, the proposed app will utilise scanning for homemade foods. This allows pet parents who feed

homemade diets to their pets to continue to receive accurate nutrition information and to maintain balanced feeding routines. Furthermore, the alert function for harmful ingredients will be added in the proposed app. This brings up the consciousness of the pet owners with immediate notification when any undesired or harmful pet food is identified. Moreover, to further the result of a health check, the app is not only equipped with image recognition for external body symptom images, but also with image-based detection to analyse pet excrements. The integration of waste visual observation analysis is vital for preventive veterinary care because it allows the detection of internal health disturbances at a very early stage. As a demonstration that appears in research, analysis of waste products can reveal major intestinal parasites that can be present without showing signs of the body until a fast decline in health occurs [24]. Therefore, by leveraging image processing technology to analyse the colour, texture and mucus contamination of the faeces, the app can deliver useful information to help users to track their pet's health [25]. Besides that, a method for automated generation of health reports will also be incorporated in such apps to offer ease of use for the pet owners, as well as the vets. This particular functionality provides a rapid and non-invasive means to collect baseline information pertaining to the health of the animal companions.

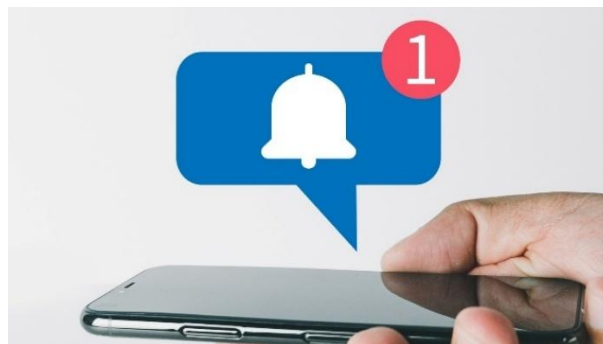


Figure 2.3.3 Receive Alert Message When Harmful Food Is Detected

Meanwhile, the proposed app brings in intelligent automation and enhanced capabilities to avoid the exhaustive manual data entry. The app does not depend on users to enter all data manually, it uses AI-powered capabilities such as image recognition for automated health and nutrition assessment, thereby minimizing user effort and enhancing accuracy. Apart from the calendar function, an automated expense tracking system by receipt scanning is developed, along with a community platform for users to share their stories and learn from fellow pet parents. The app also combines community engagement with a smart price tracking feature, enabling users to buy smartly. With these enhancements implemented, the app will

revolutionize the way pet parents have historically kept records and provide a more intelligent, engaging and overall easier to use pet care app.

Lastly, to improve the accuracy and customisation of the pet emotion translator, the solution combines audio and visual analysis on the same multimodal input processing pipeline and includes a learning mechanism for continuous training. The sound model uses a CNN-BiLSTM with an attention mechanism to obtain MFCC features of cries, while the visual model uses MobileNetV2 to recognize body posture from the video sequences. By fusing these modalities, the system can handle competing signals by choosing the prediction with the greater confidence, yielding a more robust output. Furthermore, the method utilizes supervised learning, deep learning and transfer learning as part of a continuous personalisation pipeline. Supervision learning brings strong initial accuracy from the use of labelled datasets, deep learning is used to model complex patterns over the audio and visual signals, transfer learning allows for continuous adaptation to specific individuals. User's feedbacks are used to simplify the "Correct" or "Wrong" feedback to relabel incorrect prediction and store them for final tuning of a personalised model.

In summary, the proposed application brings solutions to the aforementioned problems of both the state-of-the-art applications and the literature related to classification of cat sounds and body language. The proposed app improves the accuracy and ease of use of pet care by multilevel multimodal audio-visual analysis, a feedback-driven transfer learning-based personalization pipeline, additional food scanning capabilities, warning messages, overall health evaluation including waste analysis and a pet management feature.

Table 2.3.1 Functionality Comparison Between Preview Apps and Proposed App

Mobile App Functionality	Pawdi	TTcare	MeowTalk	11Pets	Proposed App
Commercial Pet Food Nutrition Analysis					
Homemade Pet Nutrition Analysis					
Alert Message					
External Body Health Analysis					
Waste Analysis					
Automated Generation of Health Report					
Emotion Interpretation by Combining 3 Type of Machine Learning Approach					
Calendar Reminder System that Notified User Automatically					
Expense Tracker by Scanning Receipt					
Community Platform					

Table 2.3.2 Proposed Solution to Address Weakness in MeowTalk

ASPECT		PROPOSED SOLUTION
Learning Approach	Primarily supervised learning with some reinforcement learning	Combination of supervised learning, deep learning and transfer learning
Recognition Variety	Limited to labelled and common vocalisations	Able to generalise and recognise both common and unfamiliar sound and body language using deep learning
Feedback System	Complex, requiring manual typing of the emotionals' label	Simple, providing "Correct" or "Wrong" feedback. Simply selecting correct label if wrong prediction.
Personalisation	Slow and user-dependent.	Continuous and automated via fine-tuning based on user feedback
Adaptability to Individual Pets	Limited, applying general model with minimal adaption	High, using transfer learning to adapt to each pet's unique behaviour

CHAPTER 3: Proposed Methodology

3.1 System Methodology

In the development of pet care mobile applications, the Prototyping Methodology is selected. This is because this method allows developers to provide an early working version of an app feature. It enables the implementation process to receive immediate feedback and perform direct improvements to the app. Through continuous improvement cycles, the final app will be more fulfilling with project goals and real-world needs. Therefore, to make the process clear, the development process is divided into several key steps like gathering requirements, designing, building a prototype, testing it, improving it and finally deploying it.

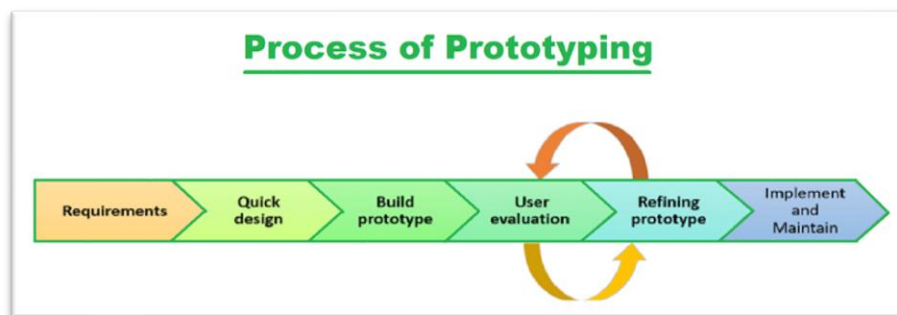


Figure 3.1.1 Prototyping Methodology

First, during the requirement-gathering and analysis phase, the essential features such as food-image recognition, pets' external body symptoms or waste detection, emotion interpretation and alert messages are identified. From comparing with other existing pet care apps, detailed requirements are collected to ensure the app's practicality. Then, in the prototype design phase, an initial user interface and system architecture plans will be involved to design a mobile application interface. In the prototype development phase, an early working version which focuses on implementing the core functionalities with basic interaction is built. For example, the development tools such as TensorFlow are used to implement a emotion interpretation tool, Firebase is used for storing pet details. Following this, the prototype is evaluated during the prototype testing phase. During this phase, developers can interact with the app to identify the problem and weakness of each functionality. Based on the feedback, prototype refinement will be started and focus on improving the app's features, user interface and performance. Refining and testing steps are repeated in loop until a satisfy prototype is achieved. Finally, at deployment phase, a real application is installed for actual use. This systematic development process ensures that the final app satisfactorily meets user requirements and project objectives.

3.2 Tool to Use

3.2.1 Hardware Architecture

The development of a pet care mobile application involves the utilisation of hardware components which include a laptop and a smartphone. As shown in table 3.1.1, a HP Laptop 15-fd1xxx served as the main workstation for design and development tasks. Besides that, a Samsung Galaxy A50 (Android), shown in Table 3.1.2, is used as the primary devices for evaluation and testing purposes in real-world environments.

Table 3.2.1 Specifications of Laptop

Description	Specifications
Model	HP Laptop 15-fd1xxx
Processor	Intel(R) Core (TM) 5 120U 1.40 GHz
Operating System	Windows 11
Graphic	Intel(R) Arc™ Graphics
Memory	8GB RAM
Storage	512GB

Table 3.2.2 Specifications of Samsung Smartphone (Android)

Description	Specifications
Model	Samsung Galaxy A50
Processor	Samsung Galaxy A8 (2018) Samsung Galaxy A7
Operating System	Android 11 with One UI 3.1
Graphic	Exynos 9610
Memory	6GB
Storage	128GB

3.2.2 Software Architecture

In order to successfully develop each feature in the pet care mobile application, software components play a crucial role in cooperating with hardware devices to develop a user-friendly mobile application. The following software tools will be utilised in this project:

Table 3.2.3 Software Architecture Description Table

Software	Description
	(a) Mobile Application Framework Development Flutter is chosen for this mobile app for pet care because it can enable cross-platform development, wherein the app is run on Android from a single source, resulting in time saving for development while offering consistent performance and interface. Built on the Dart language, Flutter enables fast compilation, reactive programming model and hot reload, accelerating prototyping, testing and tuning while maintaining high performance and silky-smooth user experience.
	(b) Integrated Development Environment (IDE) Visual Studio Code and Android Studio are selected to ensure a successful development process. Visual Studio Code is utilized for application source code writing and maintenance, offering debugging tools, extensions and version control integration.
	(c) API Integration for Nutrition and Health Analysis The pet care app sends important data from uploaded images to the Gemini API in the cloud, which performs food identification for barcode and non-barcode items and detects visible markings on pets. The API provides extensive nutritional information, health facts and tips for better food options or care. It makes development

	simpler with the convenience of having image analysis handled neatly and accurately without extra processing libraries.
	(d) API Integration for Price Comparison SerpAPI enables the extraction of up-to-date information on searches of products from Google. This functionality enables the application to conduct price comparisons for commercial pet foods by searching through different websites on the internet. The backend of the application analyzes the searches and returns the best and cheapest products.
	(e) User Data Cloud Storage Supabase is used as the cloud-based database for storing and managing application data. It handles various types of data, including user profiles, pet information, analysis details, expense records, reminders and community posts. Supabase operates on the PostgreSQL database management system, making it possible for data to be organized into tables and related to one another through relations. This makes it possible to handle complex data such as the relationship between users and pets and transactions. In addition, Supabase provides secure, scalable and real-time data management, ensuring that user data is consistently updated and easily accessible across the application.
	(f) Backend Service A FastAPI backend is employed to load the initial machine learning model and perform predictions against uploaded pet videos. The service interprets the audio data by performs the emotion interpretation and delivers the predicted outcomes to the mobile app in an efficient real-time manner.

	(g) Model Training Platform and Library The models are trained using Google Colab, and the TensorFlow, Librosa library for video processing, making feature extraction possible from pet videos. Using various methods in learning like supervised learning, transfer learning and deep learning, the app can decode pet vocalizations to human language. By the use of this approach, there can be continuous learning and individualization for a pet's own emotions.
	(h) Development Tuning Service The tool used for exposing the local backend FastAPI through a URL is Ngrok. This makes sure that the frontend mobile application can talk to the backend while the backend is being hosted locally. The role of Ngrok is to facilitate easy communication between the frontend and the backend. It enable seamless data exchange without requiring full deployment.

3.3 System Design

3.3.1 Architecture Diagram

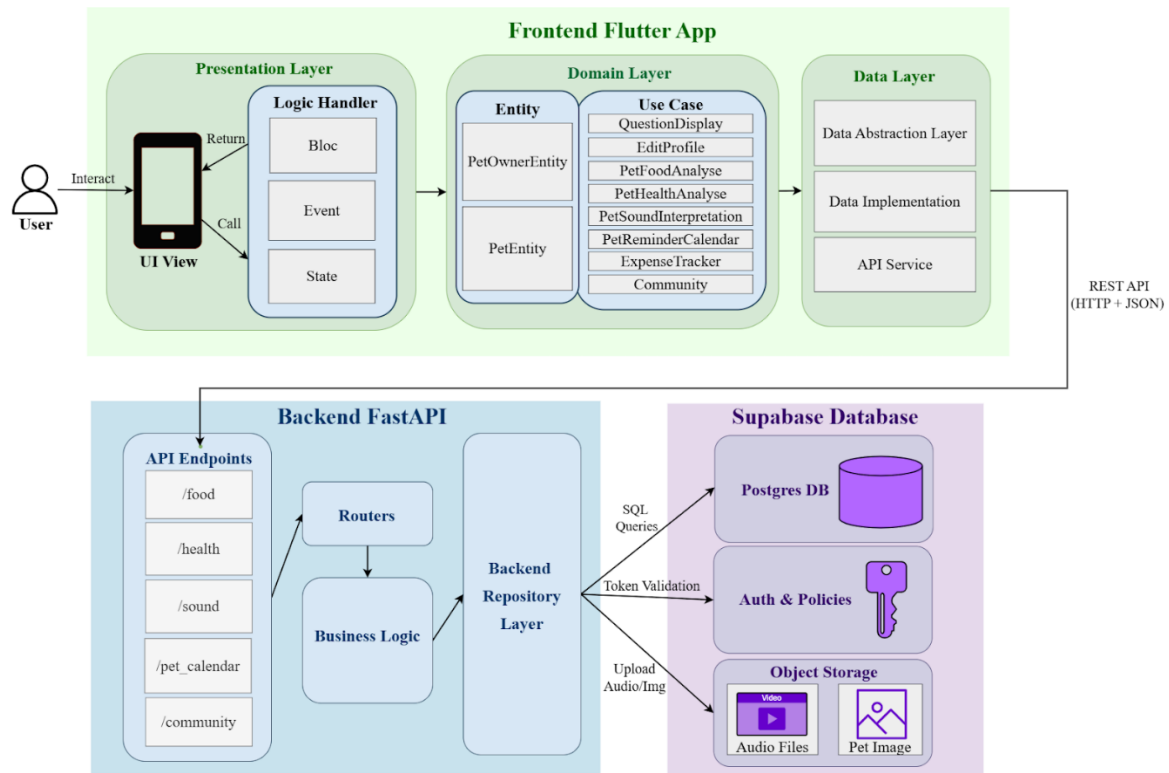


Figure 3.3.1 Architecture Diagram of Pet Care Application

The application is designed following a client-server model and consists of 3 main components: Flutter-based Frontend, FastAPI Backend and Supabase Database. The user interacts directly with the mobile app developed using Flutter, which follows Clean Architecture principles by separating the app into presentation, domain and data layers. Within the frontend part, the presentation layer acts as a user interface to communicate with the logic handler using BLoC (Business Logic Component) pattern, which manages events and states to create a smooth and well-structured user experience. Next, the domain layer contains entities and use cases. These use cases help to define how data is processed while keeping the app logic independent from the external system. Last, the data layer responsible for handling API services and data abstraction, allowing the frontend to communicate with the backend through RESTful APIs without directly accessing the database, thereby improving modularity and maintainability.

On the backend side, FastAPI is used to manage API endpoints such as food analysis, health detection, pet emotion interpretation, calendar management and community interactions. All core methods, data processing and business logic are implemented in the backend to maintain a clear separation between a user interface and the system logic. Incoming requests from the

frontend are routed through routers and pass through business logic layers before interacting with the database via a repository layer. As a result, all computations, including machine learning, model execution and data validation, are completed in the backend. This design makes the application more securely designed, scalable and easily maintained. By separating logic from frontend, the application avoids code duplication, reduces client-side complexity and improves the ease of changing or updating user interface.

Finally, the Supabase database component is implemented to provide a PostgreSQL-based relational database along with authentication and storage services. It is responsible for storing structured data such as user profile, pet information, health analysis details using well-defined tables with primary and foreign key relationships. Beside that, Supabase also manages authentication and access control through built-in policies, ensuring that only authorised users can access or modify data. In addition, object storage is used to store media files such as pet audio videos and pet images required for analysis, The backend interacts with the database by performing SQL queries, validating user tokens and handling file uploads. This ensures efficient and secure data management across the application.

3.3.2 Use Case Diagram

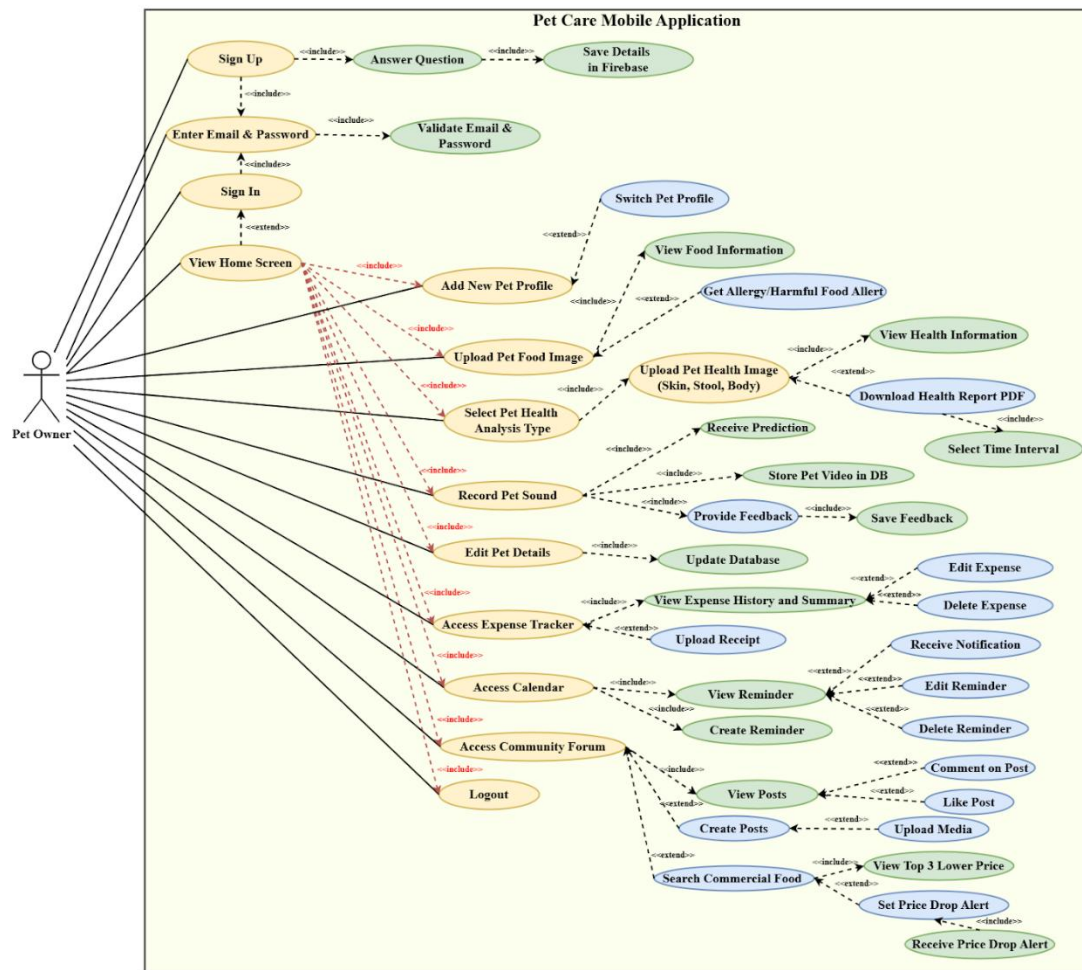


Figure 3.3.2 Use Case Diagram of Pet Care Application

The mobile app for pet care offers users a variety of exciting interactive functionalities for taking care of their pets. To get started, users sign up or sign in using their email and a password. The sign up process allows users to answer some initial information that is then stored in Firebase. After signing up or signing in, users will arrive on the home page that acts as the central place to access important features for pets such as uploading images of pet food or health images, recording pet videos, tracking expenses, creating reminders, accessing community platforms and editing pet profiles.

Firstly, users can upload food images to get analysed nutritional information. If harmful or allergenic ingredients are detected, the app sends an alert message. Besides, for health images, users must first select a specific analysis type before uploading or capturing an image. After selecting and uploading images, users can receive a detailed health report with the option to download. Beyond image analysis, the app also enables emotion interpretation by allowing

users to record pet videos which will be stored in Supabase for processing. After analysis, the app delivers predictions to users. Users can provide corrective feedback and all the feedback will be saved locally to improve personalised accuracy over time. Additionally, users can change and edit their pet profiles and these changes will be saved securely to Database. For users that have multiple pets, they are able to add multiple pet profiles and switch the profile between different pets.

Furthermore, the app also provides several pet care management features such as expense tracker, reminder system and community platform. For the expense tracker, users can upload receipts to record expenses spent on their pets and view a summary of their spending. Users are able to edit or delete expense records when necessary to ensure accuracy. In addition, the calendar reminder system allows users to create, view, edit and delete reminders for important pet-related activities. The app will also send notifications to users one day before the scheduled reminder to help them stay organised. For the community platform, users can browse and create posts, leave comments and like content shared by others. They can also search for commercial food with price comparison. After successful searching, users are able to view the top three lowest prices of that commercial food and set price drop alerts for it. Once the app detects the price dropping, it will send a notification to users. Last, users can log out of the app by using a logout function which will ensure that all data is secured appropriately and provide proper session management.

3.3.3 Activity Diagram

3.3.3.1 Sign In Activity Diagram

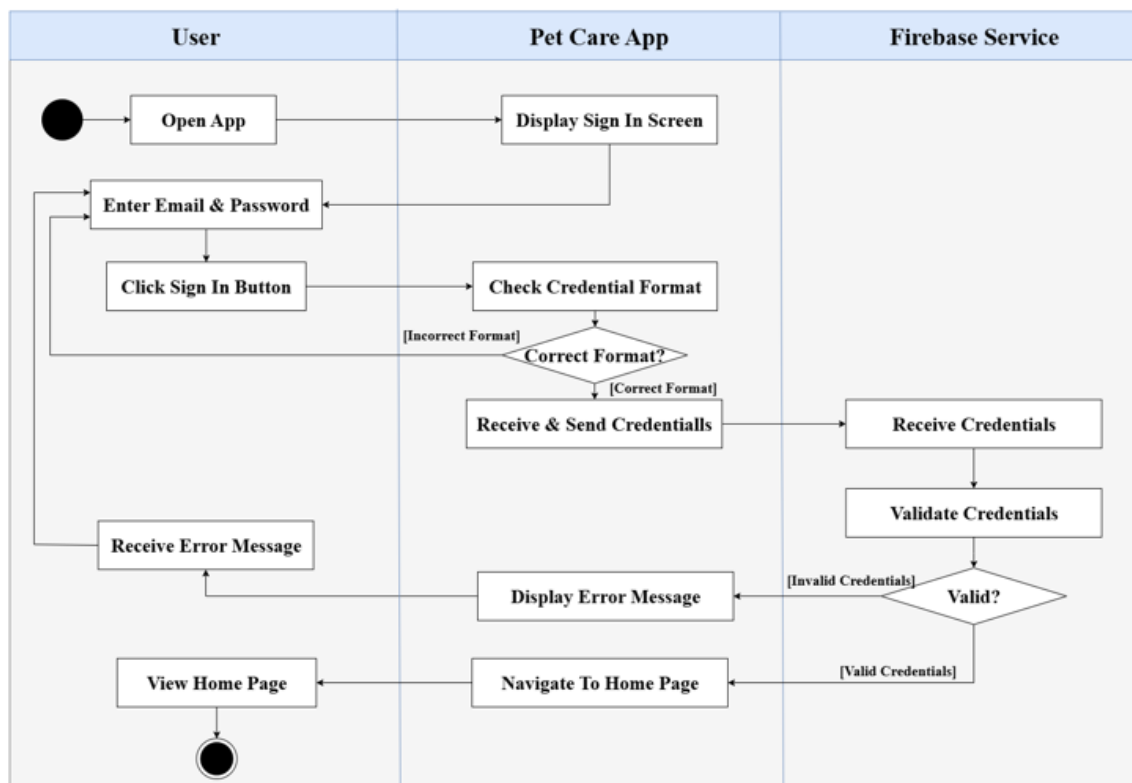
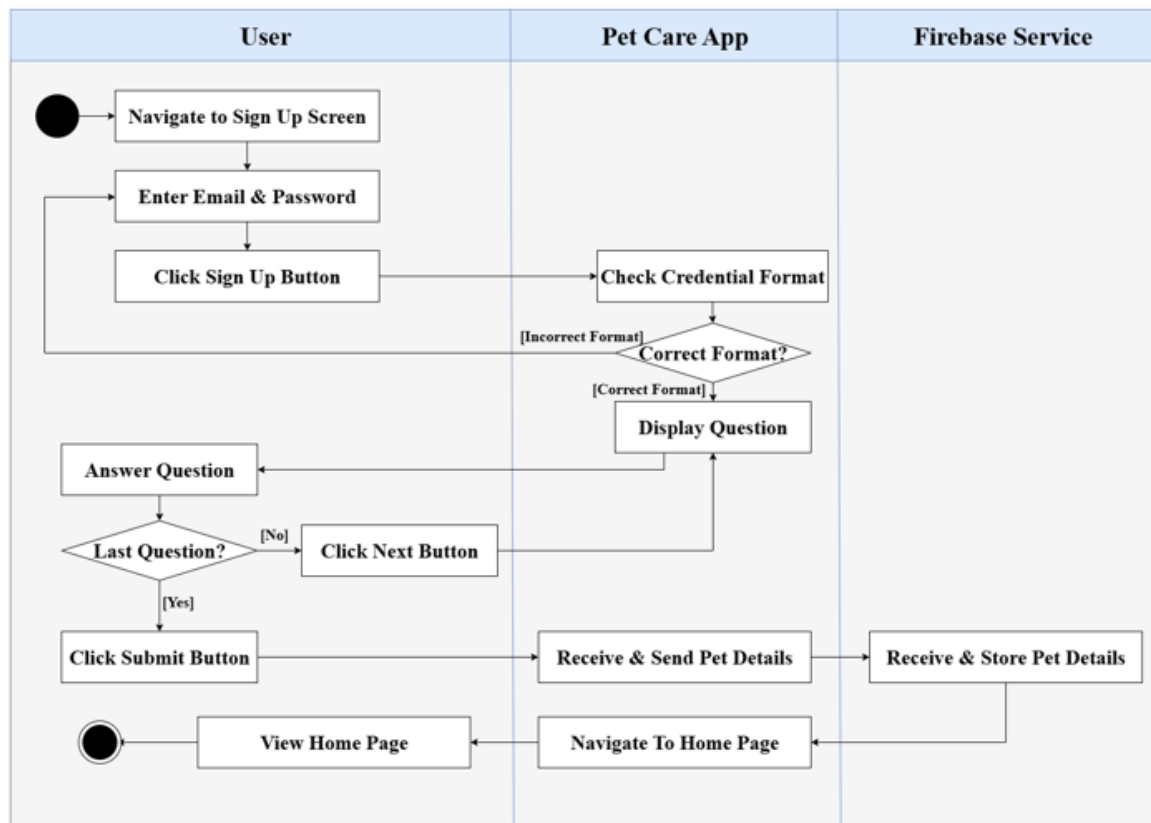


Figure 3.3.3 Sign In Activity Diagram

The app will begin with a Sign In Screen after users click to open the app. To access the Home Page, users must enter their credentials before selecting the “Sign In” button. The app will then check the credentials format. If the entered credentials format is incorrect, users have to re-enter the email or password until it is correct. If the credentials format is correct, the app will receive and send the credentials to Firebase Service. Firebase Service will validate the accuracy of credentials. If there are no matching credentials, users have to re-enter the credentials.

3.3.3.2 Sign Up Activity Diagram**Figure 3.3.4 Sign Up Activity Diagram**

For novice users, they should navigate to the Sign Up Screen to create an account by entering email and password in the correct format. The app will check the format. If not fulfilling the format, users have to re-enter the correct format credentials. If users fulfill the format, the app will display questions to let users answer, Users have to answer all the questions and click on the Submit Button. After submitting, the app will save all the details in Firebase Service and display Home Page.

3.3.3.3 Pet Food Nutrition Analysis Activity Diagram

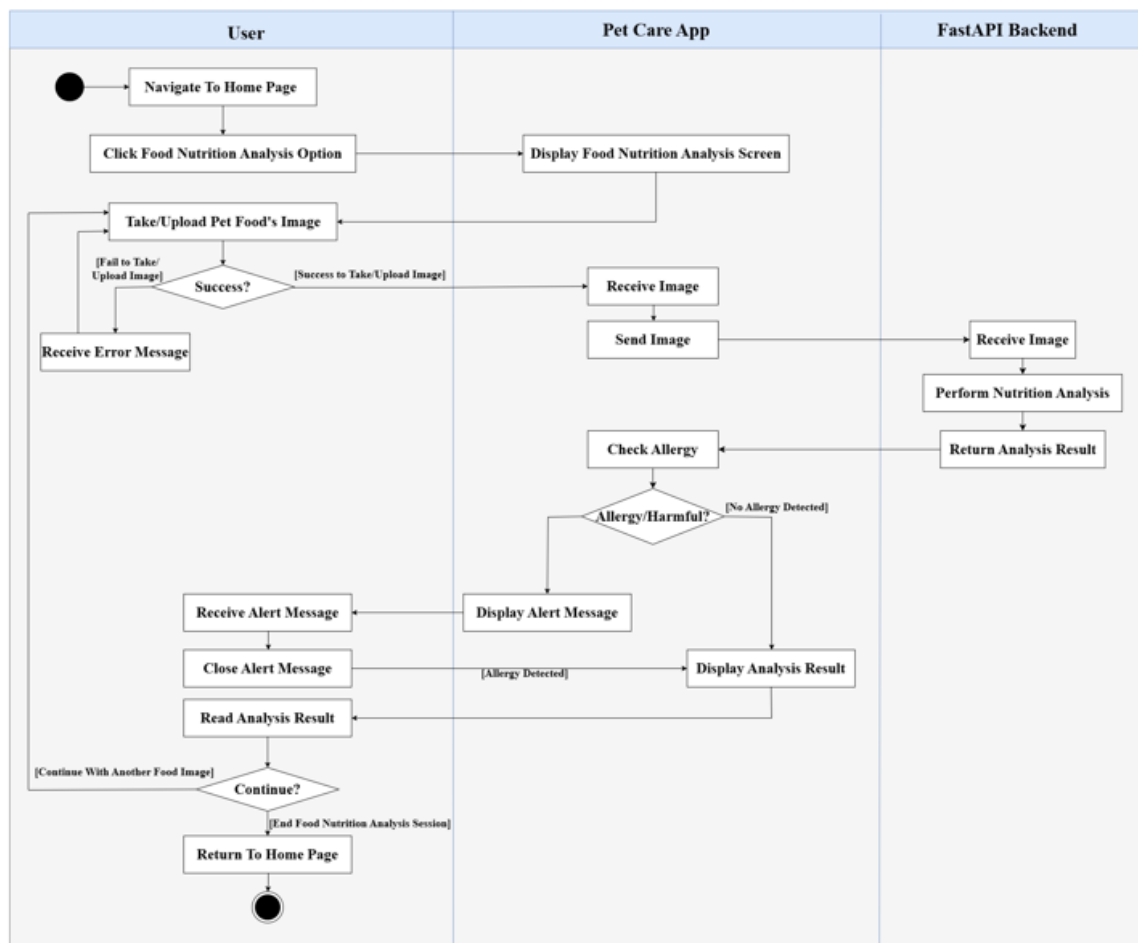


Figure 3.3.5 Pet Food Nutrition Analysis Activity Diagram

In the Home Page, users can click Pet Food Nutrition Analysis Option and the app will display Pet Food Nutrition Analysis Screen. In this screen, users can choose to either take or upload images of pet food to make analysis. If the image fails to upload, users will receive an error message. If the image is successfully uploaded, the app will send the image to the backend in order to perform nutrition analysis. Once successful analysis, the backend will return the analysis result back to the app. If a harmful or allergenic ingredient is found, an alert will be sent directly to the user immediately. If no harmful ingredients are detected, the resulting analysis will appear for the user to read. Once completed reading, users can continue uploading another food image or terminate the session by returning to Home Page.

3.3.3.4 Pet Health Analysis Activity Diagram

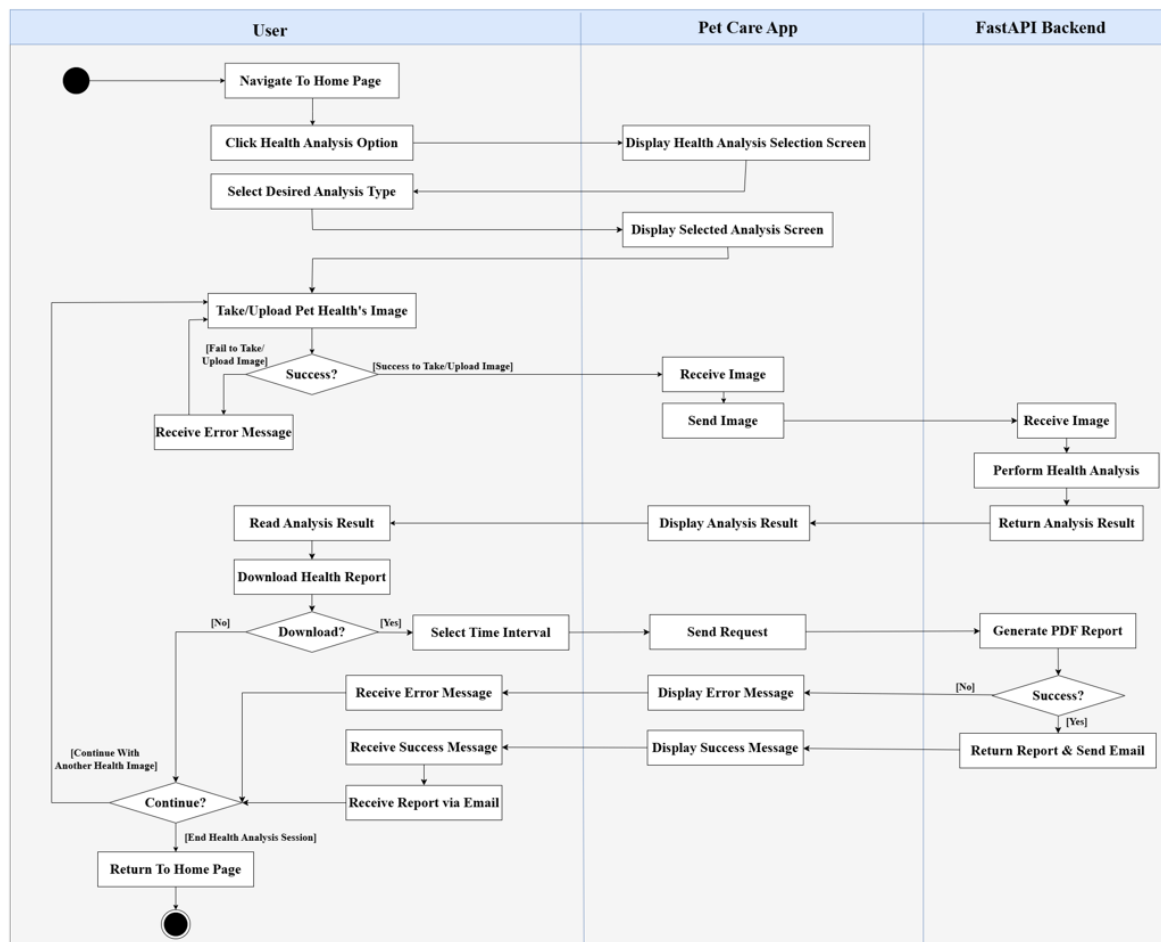


Figure 3.3.6 Pet Health Analysis Activity Diagram

Users can access the Pet Health Analysis Screen by tapping on the Health Analysis option on the Home Page. After the Screen is displayed, users can choose their desired analysis type to allow the app to take the user to the appropriate screen. Upon swiping the screen, users can choose either capturing a picture of a pet health image or uploading a pet health image. If the user fails to upload the image, then an error message is shown and allows the user to retry uploading the health image. If the image upload is successful, the app sends the health image to the backend. The backend will start performing health analysis using integrated API. After the analysis is completed, the results are shown to help users understand their pet's health condition better. Users are able to select a specific time interval to download the health report that is generated by the backend from the history collection. The user can also either start another analysis type or return to the Home Page to end this session.

3.3.3.5 Pet Emotion Interpretation Activity Diagram

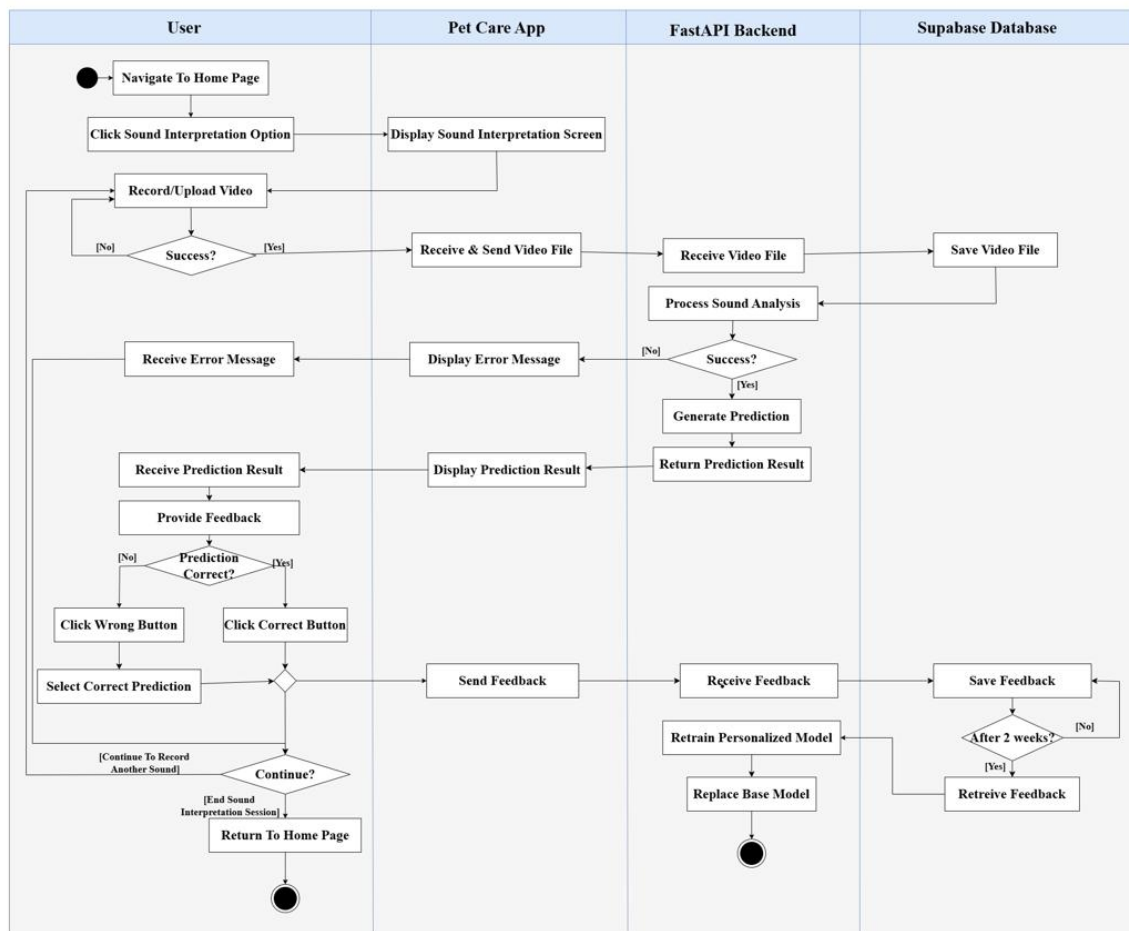


Figure 3.3.7 Pet Emotion Interpretation Activity Diagram

From the Home Page, users can click the Pet Emotion Interpretation option which will take them to the Emotion Interpretation Screen. In this functionality, users can record or upload pet videos by clicking the button. When the video is successfully uploaded, the app sends the video file to the FastAPI backend to process analysis. The backend will also save the video into the database. Once the analysis is completed, the backend will generate a final prediction and return it to the app. The app will be responsible for displaying the prediction result to the user. Users can continue the flow by providing feedback to the app. If the prediction is wrong, users can click on Wrong Button and select the correct label that they think is correct. If the prediction is correct, users can click on Correct Button. Both kinds of the feedback will be saved into database through the backend. Every 2 weeks, the backend will retrieve the collected feedback from the database and retrain the personalised model. This personalised model will replace the base model to improve the accuracy of the model. Last, users choose to go to another emotion interpretation or back to the Home Page.

3.3.3.6 Pet Calendar Reminder Activity Diagram

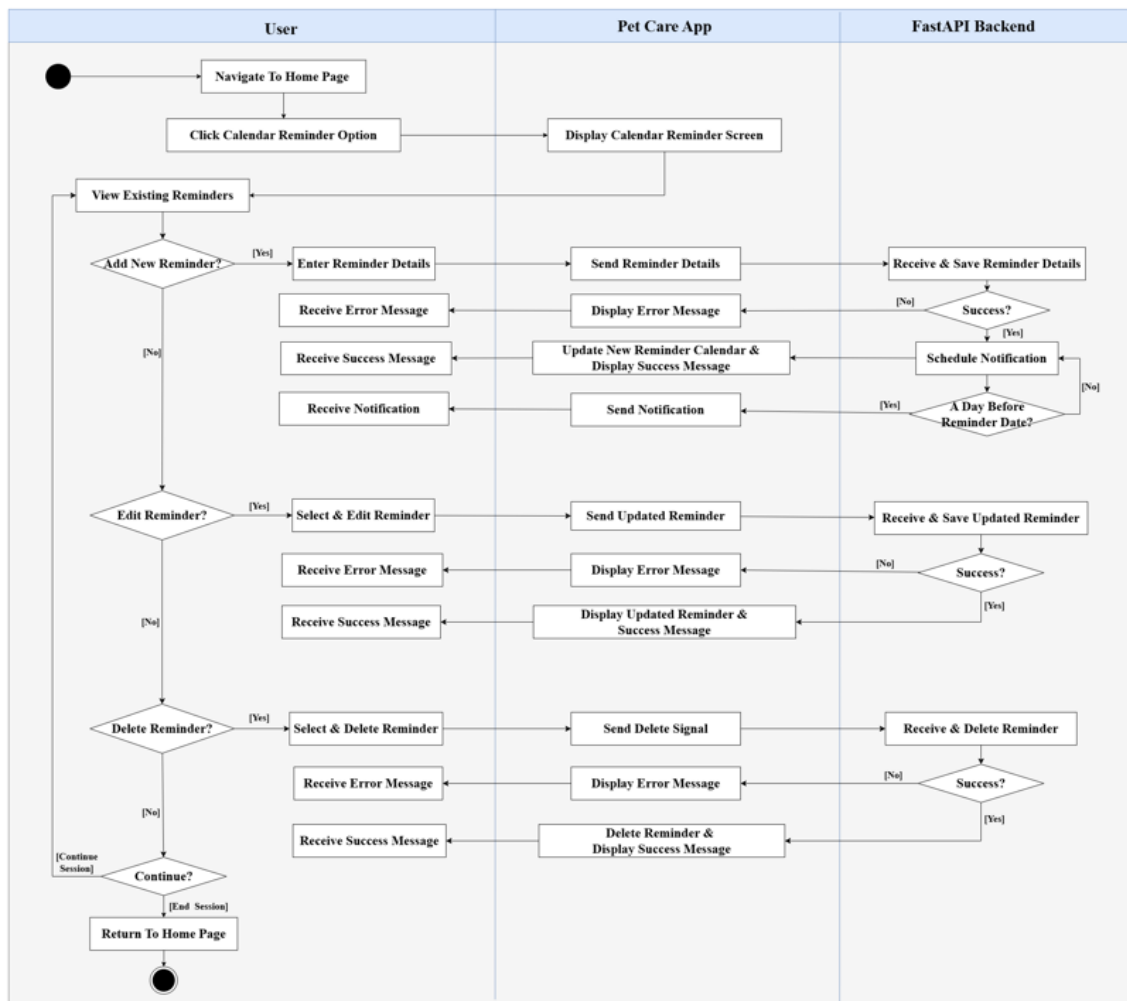


Figure 3.3.8 Pet Calendar Reminder Activity Diagram

Users can view existing reminders by clicking on the Calendar Reminder option in the Home Page. In this screen, they can choose to add new reminders, edit or remove existing reminders. To create a new reminder, users can simply enter the required details in the app. Then, the app will send the details to the backend. Once the reminder is successfully saved, the backend schedules the notification and updates the calendar view. A confirmation message is shown to the user and the updated calendar is displayed. The scheduled notification will be popped out a day before the scheduled reminder to help users keep tracking the pet-related activities. Besides, for editing or removing reminders, users can select the reminder that they wish to modify and update or remove it. Any changes or deletion requests are sent to the backend, which updates or deletes the corresponding reminder in the database. If users want to exit the screen, they can click on the back button to return to the Home Page.

3.3.3.7 Pet Expense Tracker Activity Diagram

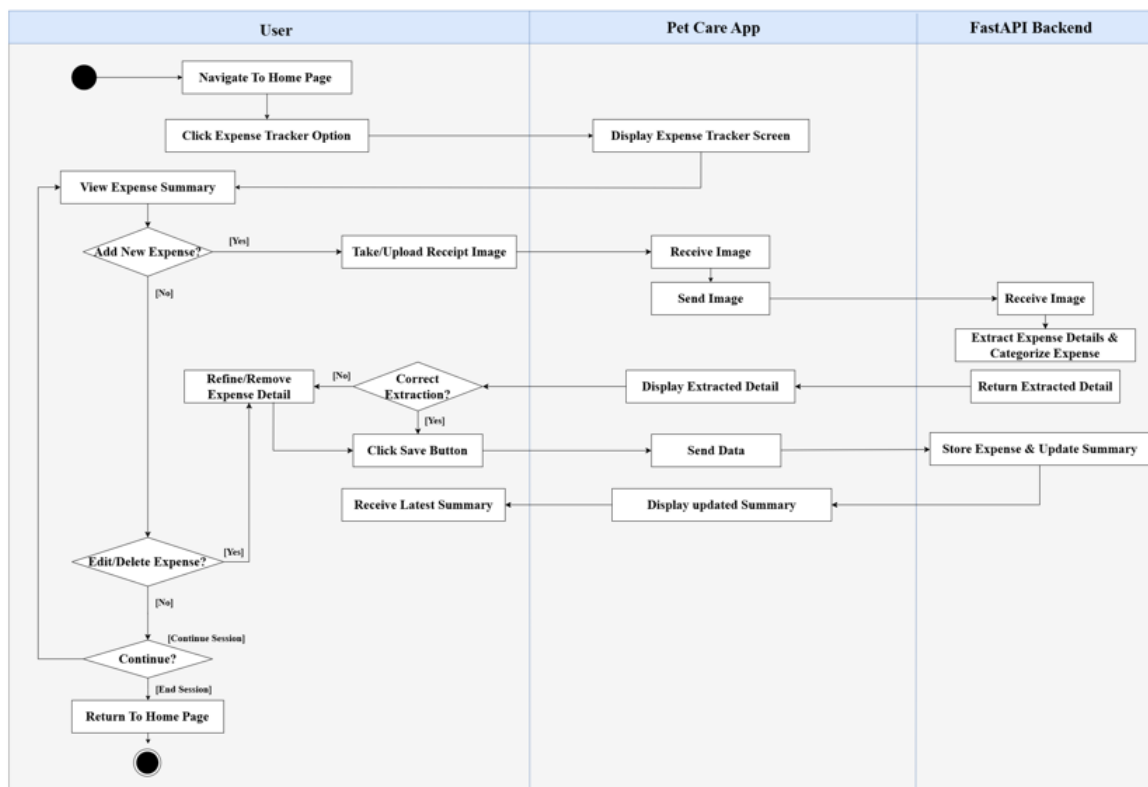


Figure 3.3.9 Pet Expense Tracker Activity Diagram

From the Home Page, users are able to click on the expense tracker selection to monitor the expense spent on their pets. The app will provide a summary of expenses inside this screen. To insert new expenses, users can take or upload receipt images to the app. The app then sends the image to the backend, extracting total expense and categorising it into one of the categories automatically. Once the processing is completed, the app will display the extraction information to the user for confirmation. Based on the extraction, users can determine whether it is correct or not. If the extraction is correct, users can proceed by clicking on the save button and the data will be saved in the database. The latest expense summary will be updated accordingly. Otherwise, if the extraction is incorrect, users can refine or remove the expense detail. To exit the expense tracker screen, a back button is provided to return back to Home Page.

3.3.3.8 Pet Community Interaction Activity Diagram

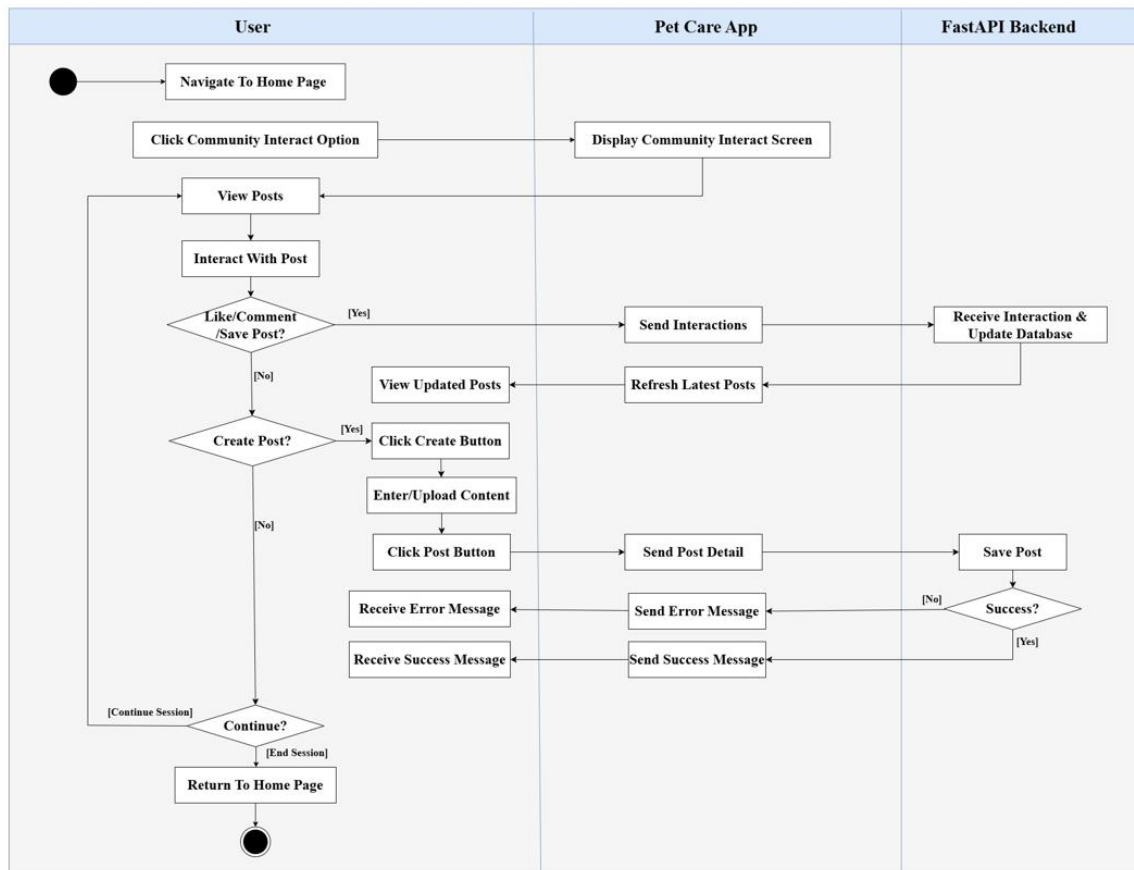


Figure 3.3.10 Pet Community Interaction Activity Diagram

To access community interaction features, users can tap on the Community Interact Option that is located at the bottom of the Home Screen. Once clicked, the app will redirect users to the Community Interact Screen. Users are able to browse the posts shared by others as well as their own. They can also interact with the post by liking, commenting or saving the posts. All the interactions will be sent to the backend and stored in the database. Thus, the posts' content is updated in real-time. To create a new post, users can click on the create button and enter or upload content. After completing the post, users can click on the post button in the app. The content of the post will be automatically saved in the database through the backend. If the process is successful, the app will send a success message to the user. However, if the process is interrupted or failed, users will be notified by an error message.

3.3.3.9 Price Comparison and Price Drop Alert Activity Diagram

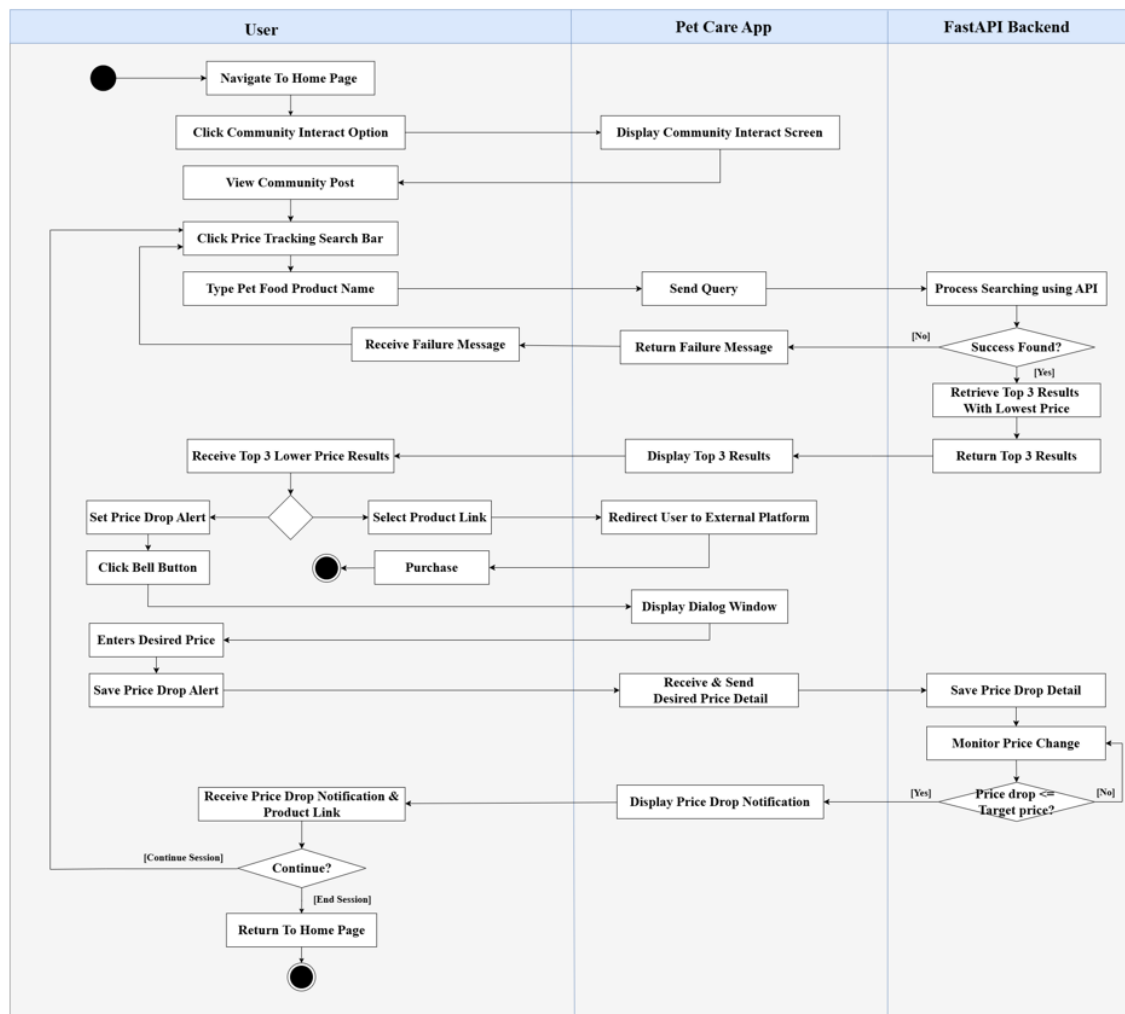


Figure 3.3.11 Price Comparison and Price Drop Alert Activity Diagram

In the Community Interaction Screen, users can also perform price comparisons for the commercial pet food. They can use the price tracking search bar that is located at the top of the screen to type the product name. After clicking the search button, the app will send a query to the backend. The backend is responsible for process searching using Google SerpAPI. Once the backend successfully finds the answer, it will retrieve and return the top three lowest-priced results to the user. Users can select a desired product from the result and click on the product link that is attached. Then, the app will redirect users to the specific external platform, so users can make purchasement. Besides that, users can also set a price drop alert by clicking on the bell icon and entering the desired target price into dialog windows that display on the screen. After confirming, users have to click on the save button. The information will be saved in the database through the backend. The backend will continuously monitor the changes of price

every midnight. If the price drops to or below the users' target price, the app will display a price drop notification to inform them.

3.3.3.10 Edit Profile Activity Diagram

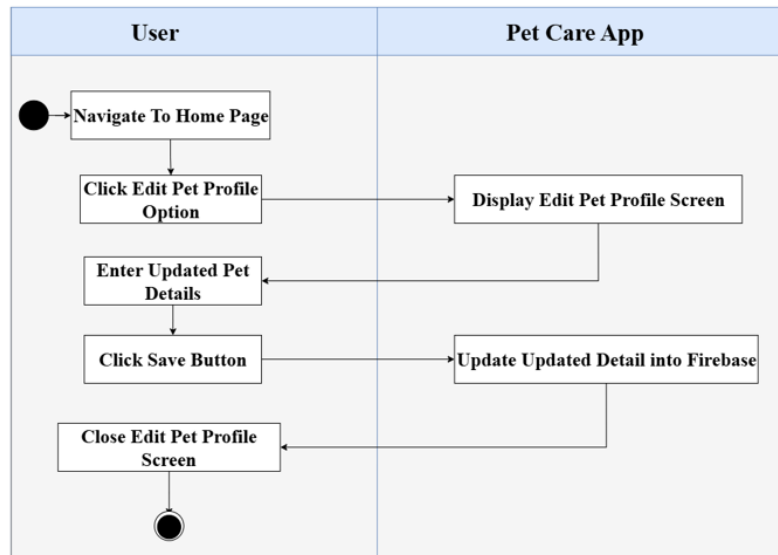


Figure 3.3.12 Edit Profile Activity Diagram

To edit or update pet profiles, users can click Edit Pet Profile Option in the Home Page. After clicking, the app will display the specific screen to let the user enter updated pet details. Once completed editing, users can click on Save Button and the app will update all the details into Firebase Service. Users can choose to return to the Home page to terminate this session.

3.3.3.11 Logout Activity Diagram

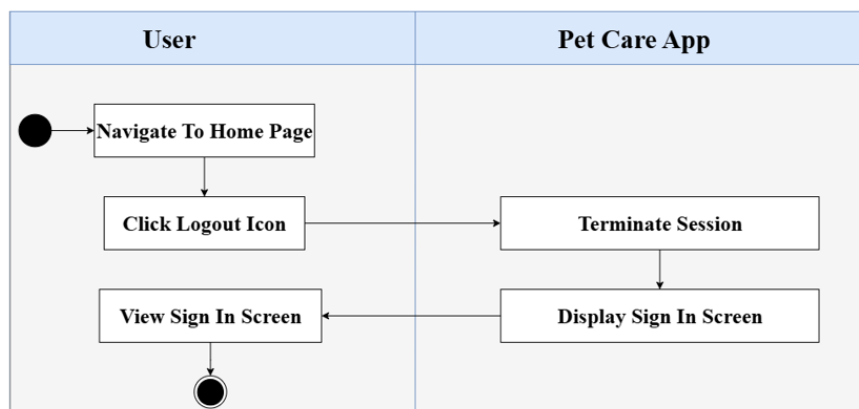


Figure 3.3.13 Logout Activity Diagram

To exit the app, users can click on the Logout Icon that is shown in the Home Page. The app will exit the app and display Sign In Screen to users.

3.4 Project Timeline

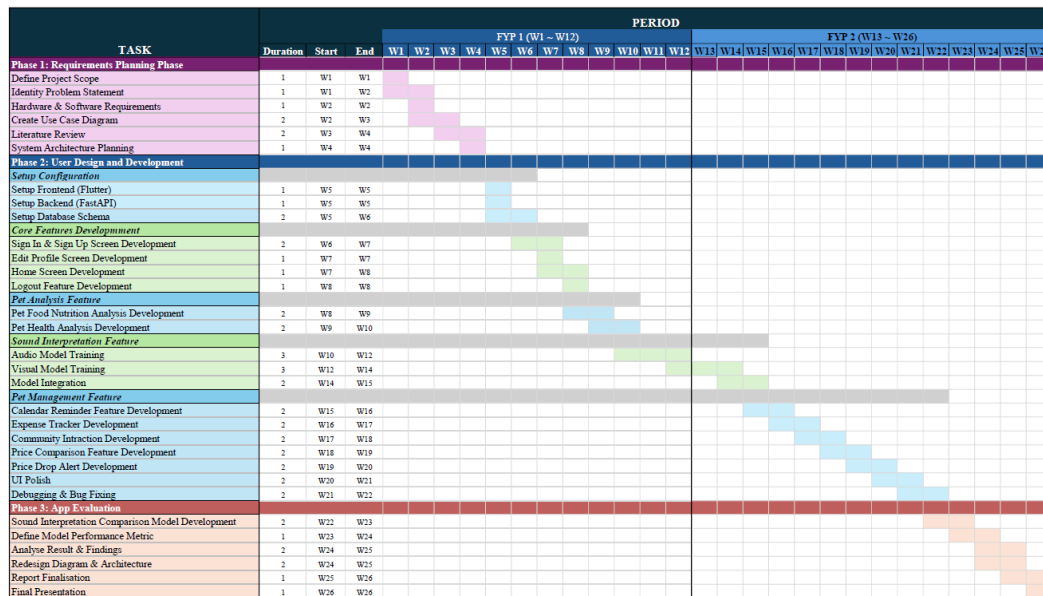


Figure 3.4.1 Project's Timeline

The project Phase 1 Requirements Planning is the first four weeks and at this time the foundation is set by scoping the project, stating the problem and developing hardware and software requirements and other planning artifacts such as use case diagrams and a system architecture. This phase also includes the review of the related literature to learn about the prior art and as basis of the system to be proposed, so that the group will be guided in the course of system development having clear direction and well-grounded document.

From Week 5 to Week 22, the focus of work the bulk of the project work psycho: Phase 2 User Design and Construction. It starts with the frontend, backend and database configuration and continues by implementing the core screens such as sign in, sign up, home screen, profile management. Then the project proceeds to develop pet-oriented functions such as food nutrition analysis, health analysis and a signature emotion interpretation system, which requires training an audio model as well as a visual model prior to combining both. In the last half of this phase, pet management with a calendar reminder, expense tracker and community interaction module, price comparison tool and price drop alert are also be developed.

Last, Phase 3 App Evaluation is concerned with testing. An alternative model is designed to serve as a reference for the emotion interpretation feature, then the performance measures are defined and the outcomes are studied. Concurrently, the system diagram are reviewed and refined as necessary, the final report is written and edited and the project is terminated with the Final Presentation in Week 26.

CHAPTER 4: System Design

4.1 System Overview Flow Chart

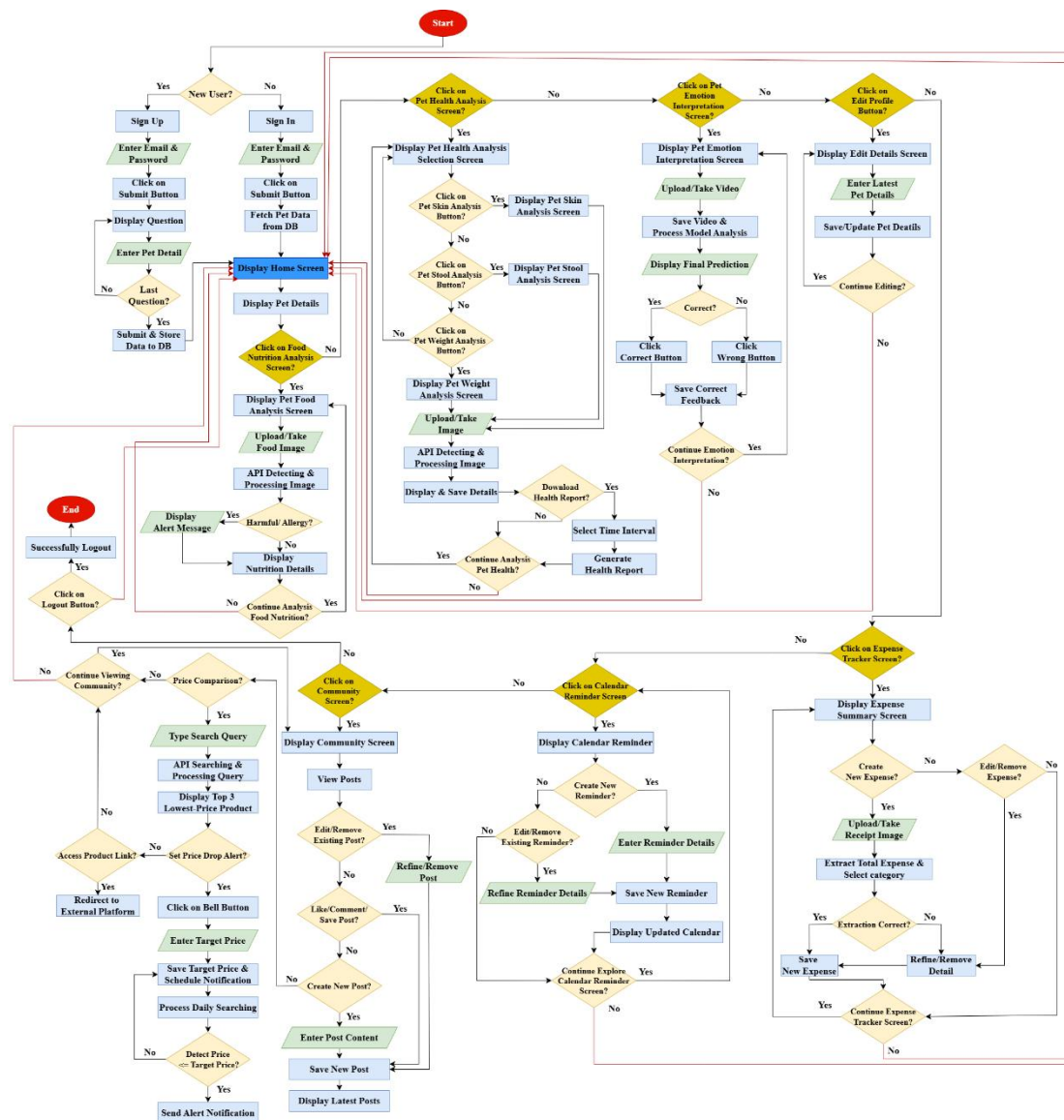


Figure 4.1.1 Flow Chart of Pet Care Application

The app will start with the Sign In process, which guides both new or old users into the app. To set up an account, new users must enter their email and password, followed by responding to five questions about their pet information. These details will be saved in the database and provide the basis for future analysis. Once they complete this process, the user will be redirected to Home Screen. For all returning users, logging with the correct credentials will bring them directly to Home Screen without repeating the questionnaire.

At the Home Screen, the app will automatically display the stored pet details and provides 7 main features: Pet Food Nutrition Analysis, Pet Health Analysis, Pet Emotion Interpretation, Calendar Reminder, Expense Tracker, Community and Edit Profile. Each feature is created to assist pet owners to more effectively care for their pets by providing tailored information and tools needed. When users are finished engaging with the app, they can freely choose any feature, go back to Home Screen or logout.

In Food and Health Analysis, users can upload or capture images for processing through APIs. The Food Analysis can assess the nutritional content and provide warning for hazardous or allergenic items, while Health Analysis has a few different categories: skin, stool and body weight. Each of these health features processes the uploaded image and provides details about each of the analysed objects, and users can optionally download reports in PDF format. In both modules, users can proceed to additional analyses or return to the Home Screen after a session.

Furthermore, the Emotion Interpretation and Profile Management complete the app's functionality. In Pet Emotion Interpretation, users record video which can be stored in a database and run it through the FastAPI backend to predict pet emotion. Users can choose to confirm and correct the prediction with feedback for continuous improvement. Meanwhile, the Calendar Reminder feature allows users to keep track of their pet-related activities by entering reminders' details in the app. All the reminders will be saved in the database and automatically generated a notification a day before the activity. Besides, the Edit Profile features will allow users to update or edit information about their pets that is stored at Firebase if required again in the future. Thus, users can get a seamless user experience which allows them to analyse, update or simply even navigate the application easily. These features had the goal of providing a seamless and interactive pet care application.

Users can manage pet-related expenses through the Expense Tracker by uploading or capturing receipt images, where the system automatically extracts and categorises the expenses. If the extracted details are inaccurate, users can edit or remove them easily. To enhance the user experience, a Community Interaction section was also developed. Users are allowed to read posts, share experiences and ask questions. For price comparison, users can search for pet food products using a built-in search bar which returns the top 3 lowest-priced results. Then, users can choose to purchase the product via the provided link or set a price drop alert. For price drop alert, the app will execute the saved query daily. The app monitors saved queries daily and notifies users when the price falls below the desired value.

4.2 Database Design

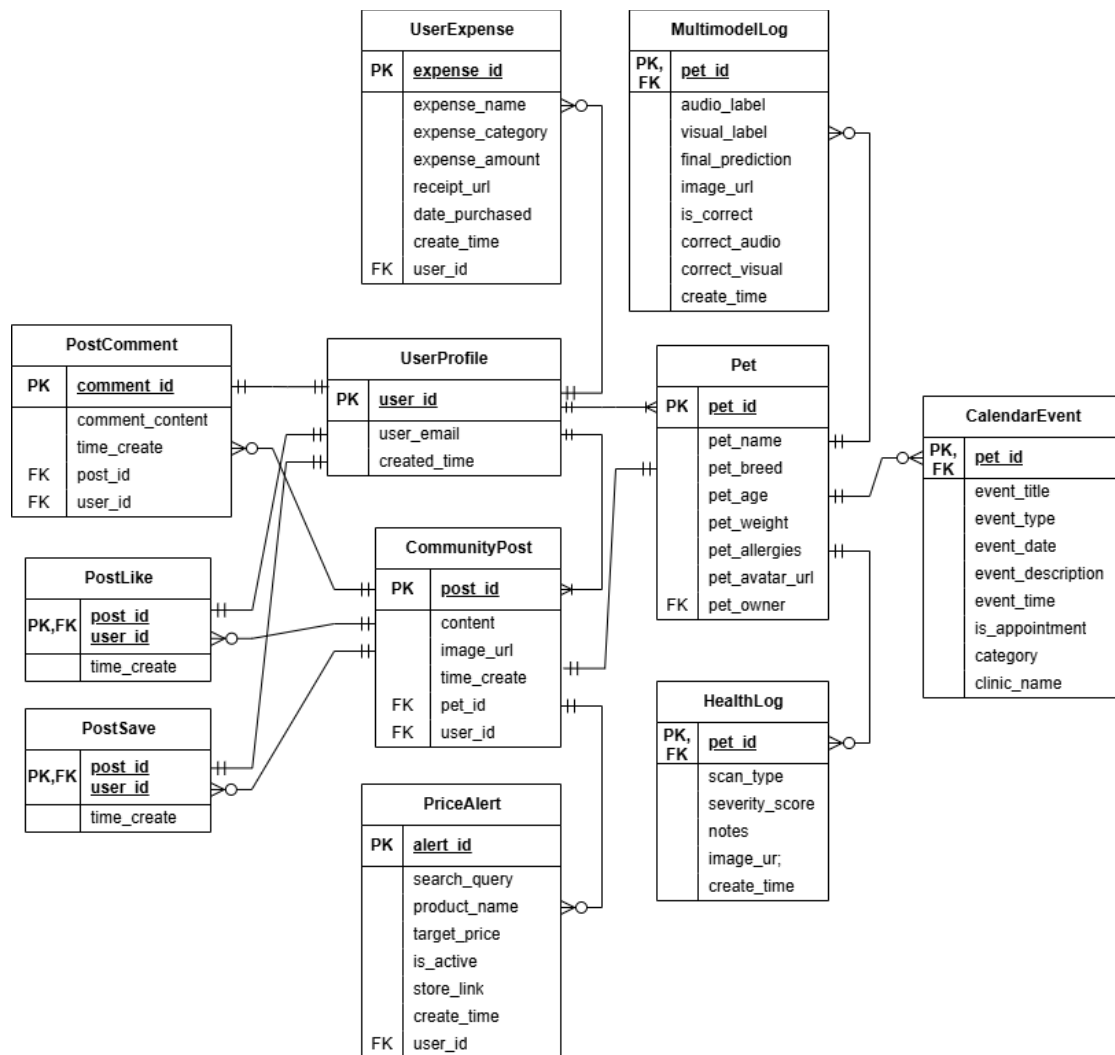


Figure 4.2.1 ERD Diagram of Pet Care Application

The ERD diagram above is built around UserProfile which connects all features through user_id. Each user can manage multiple Pets which act as the core entity that connects to HealthLog and MultimodelLog entity. The HealthLog table is responsible for storing AI scan results and its severity score, while the MultimodelLog table is accountable for recording predictions from the interpretation model. CalendarEvent table manages vet appointments and pet-related reminders. Furthermore, through the CommunityPost table, this app offers a social media aspect where users can post pet updates and communicate through comment, like and save features. On the financial side, the UserExpense table allows users to upload their receipts to track their expenses and the PriceAlert table to notify users when the price of pet food falls below their target amount. In summary, the application will provide features to help track pet health, social interaction, manage expenses related to pets and provide smart alerts, all through the use of a single user account.

4.3 Module Design

4.3.1 Sign In Sequence Flow

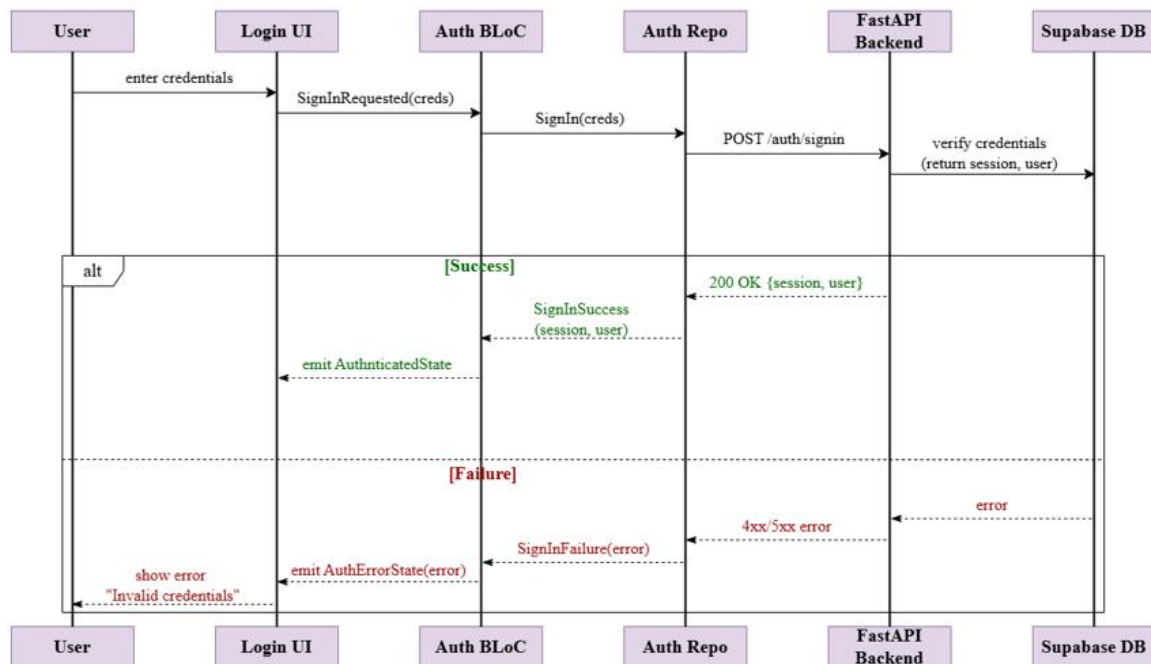


Figure 4.3.1 Sign In Sequence Flow Diagram

When a user logs into the app via the Login User Interface, credentials are submitted to the Authentication BLoC, then sends an event with the user's credentials to the FastAPI backend. The FastAPI backend checks to see if the credentials are valid by querying the Supabase database for a user with those credentials. If there is a valid user, then the backend sends back a session. Upon receiving a session, the BLoC emits an `AuthenticatedState` to the rest of the app along with the tokens received from the FastAPI backend. If the `SignIn` event fails to process, a `SignInFailure` error is returned back to the Login user interface with an invalid credential user message.

4.3.2 Sign Up Sequence Flow

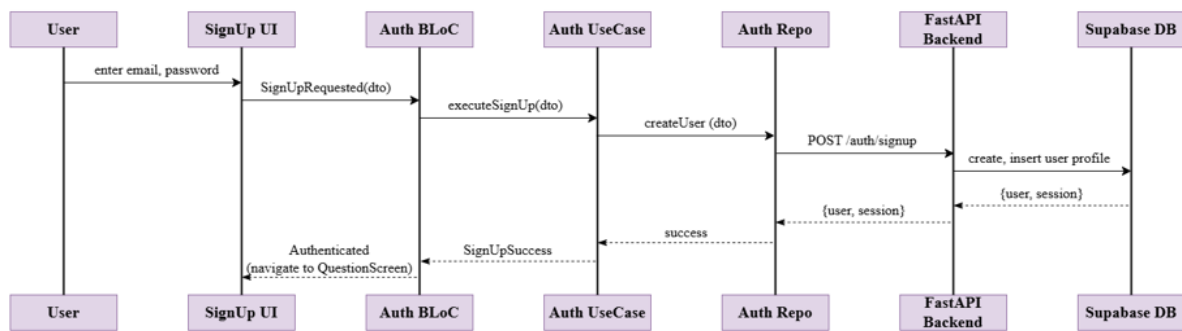


Figure 4.3.2 Sign Up Sequence Flow Diagram

When a user creates their account using the SignUp user interface, they provide their email and password, which sends a SignUpRequested event through the Authentication BLoC and UseCase layers to the FastAPI back-end via the Authentication Repo. The FastAPI backend creates a user in the Supabase database, along with their profile. If the create user account process succeeds, the backend returns a session back through the layers and the user will be authenticated and navigated to the Questions screen.

4.3.3 Pet Food Nutrition Analysis Sequence Flow

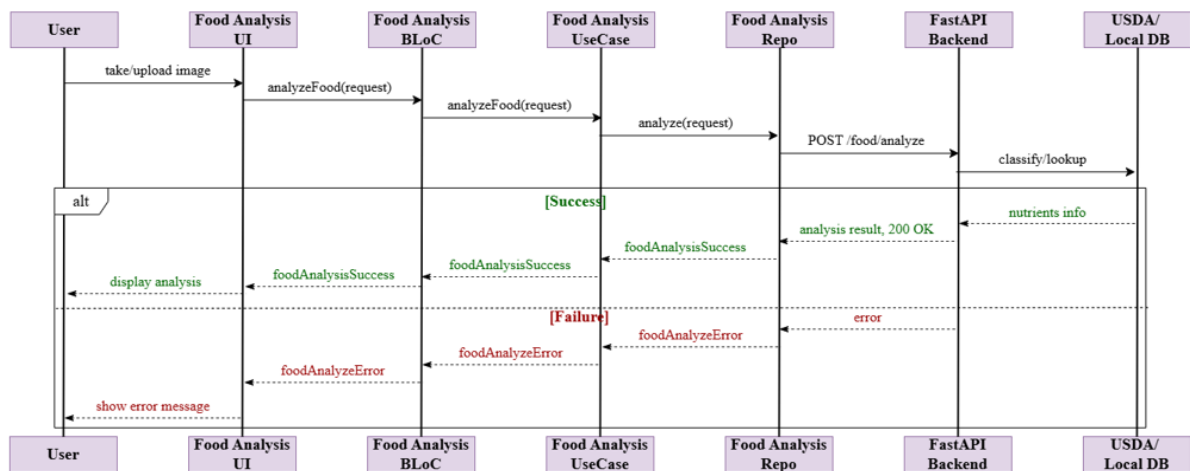


Figure 4.3.3 Pet Food Nutrition Analysis Sequence Flow Diagram

The user takes or uploads a food image in the Food Analysis UI which triggers analyzeFood through the BLoC, UseCase and Repo layers before hitting the FastAPI backend with a POST /food/analyze request. The backend classifies the food using the USDA for noncommercial food or Local DB for commercial food, then returns the nutritional analysis info. Once successful analysis, the result flows back through the layers and is displayed to the user. But, if analysis meets trouble, an error message is shown to the user.

4.3.4 Pet Health Analysis Sequence Flow

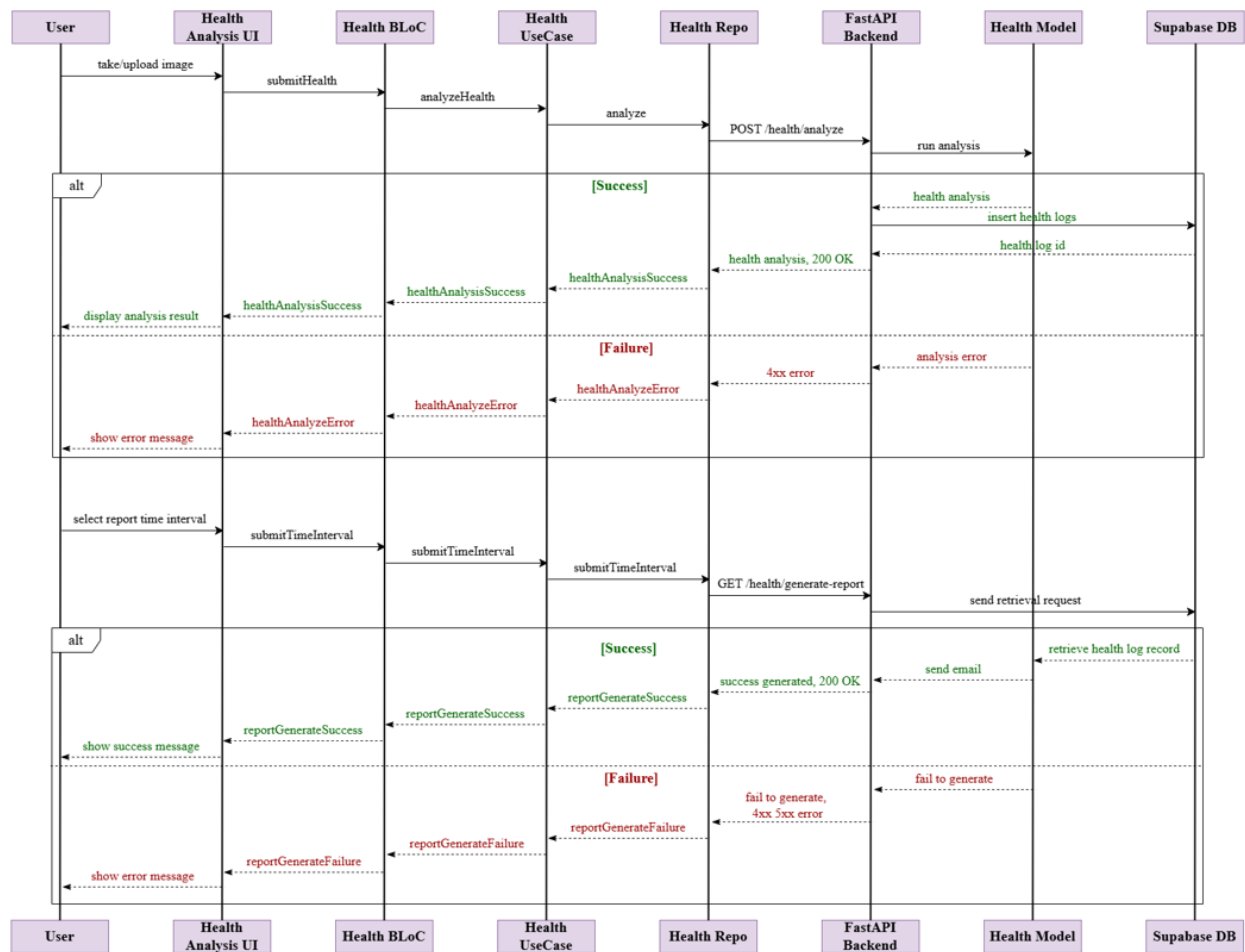


Figure 4.3.4 Pet Health Analysis Sequence Flow Diagram

The user submits a photo via the Health Analysis UI, which sends the request through the Health BLoC, UseCase and Repo to the FastAPI backend via POST /health/analyze. The backend runs the analysis using a Gemini API, inserting the result into Supabase DB and returning the health analysis details. Once successful, the result is displayed to the user and they can optionally select a report time interval to generate a health report. The backend will then retrieve the relevant health log records and send a report, or show an error if generation fails.

4.3.5 Pet Emotion Interpretation Sequence Flow

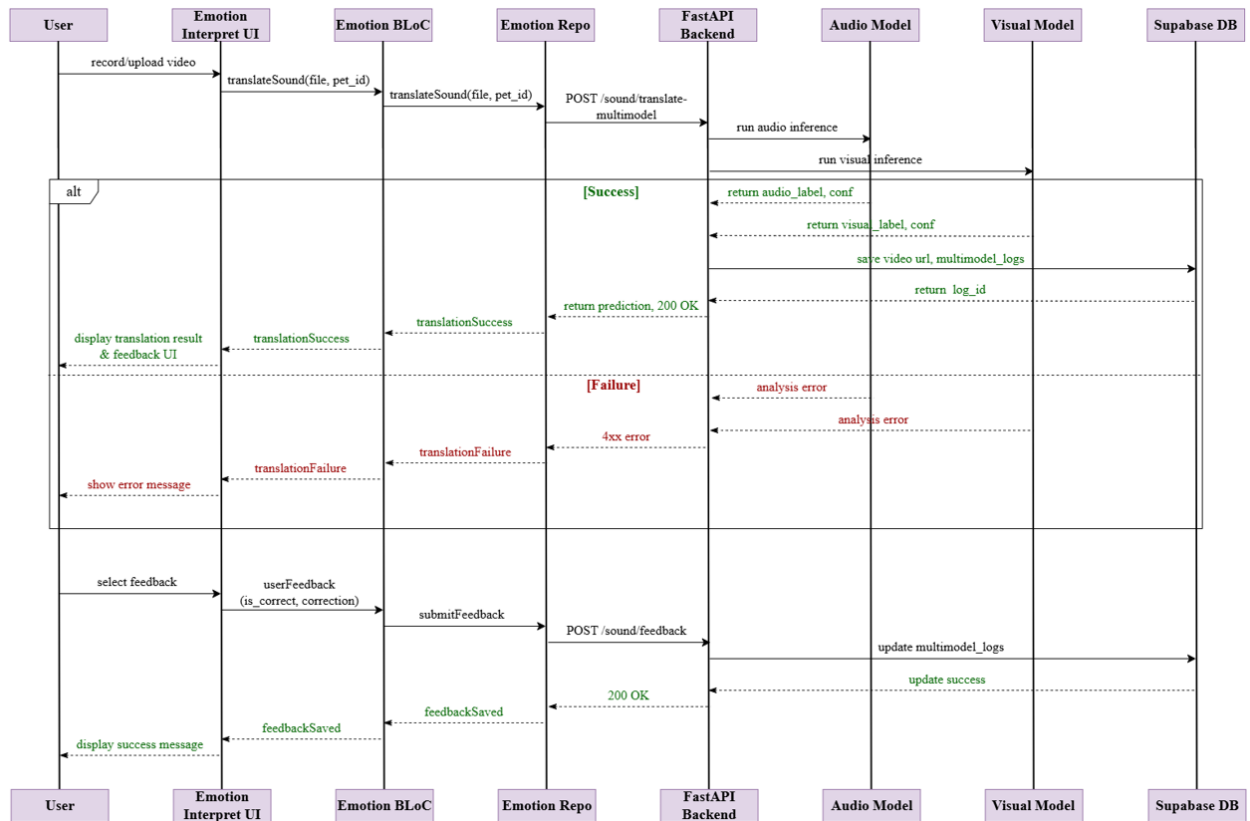


Figure 4.3.5 Pet Emotion Interpretation Sequence Flow Diagram

The user records or uploads a pet video which the Sound BLoC forwards through the Sound Repo to the FastAPI backend via POST /sound/translate-multimodal. The backend runs both audio and visual inference in parallel using separate models, then provides the final results, saves the media URL and records the prediction to Supabase DB. If the analysis succeeds, the translation result and a feedback UI will be shown to the user. Otherwise, an error message is displayed if there is a failure. After viewing the result, the user can submit feedback on whether the translation was correct and the app will trigger a POST /sound/feedback call that updates the log record in Supabase accordingly.

4.3.6 Pet Calendar Reminder Sequence Flow

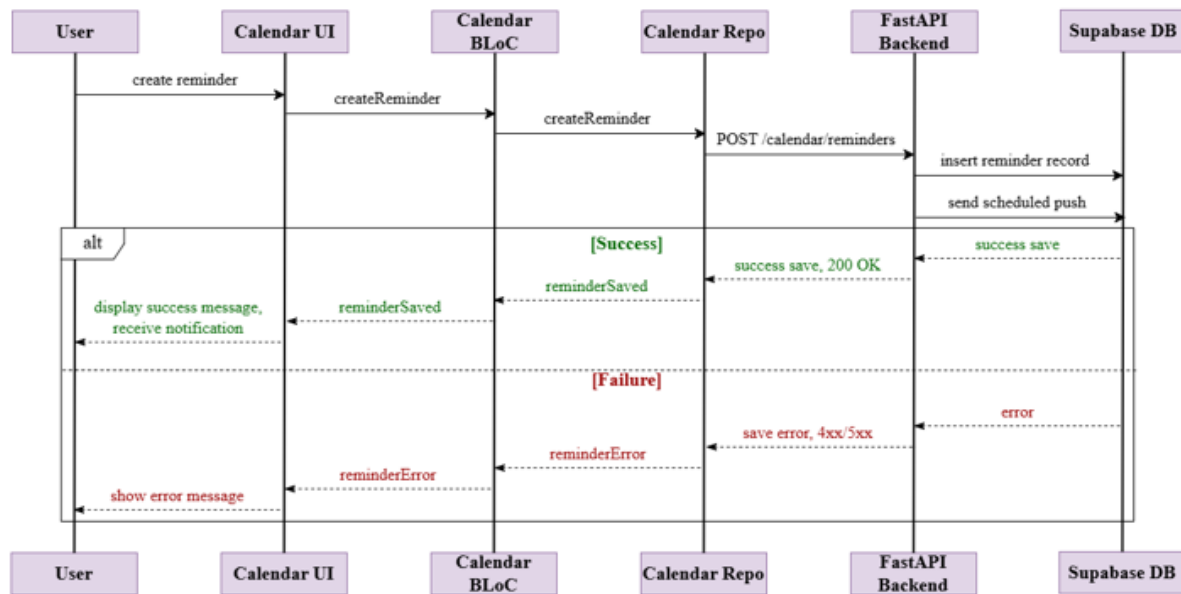


Figure 4.3.6 Pet Calendar Reminder Sequence Flow Diagram

When a user creates a reminder, the Calendar UI passes the request through the Calendar BLoC and Repo, which sends a POST /calendar/reminders to the FastAPI backend. The FastAPI backend will store this new reminder record in Supabase database and also schedule a push notification for it once the saving has completed. Next, confirmation of successful save will return back to the user with a success message shown in the UI. Finally, the push notification will then be delivered at its scheduled time to the user. However, if anything went wrong while the app was trying to save the reminder, an error response will be sent back to the user with an error message shown in the UI.

4.3.7 Pet Expense Tracker Sequence Flow

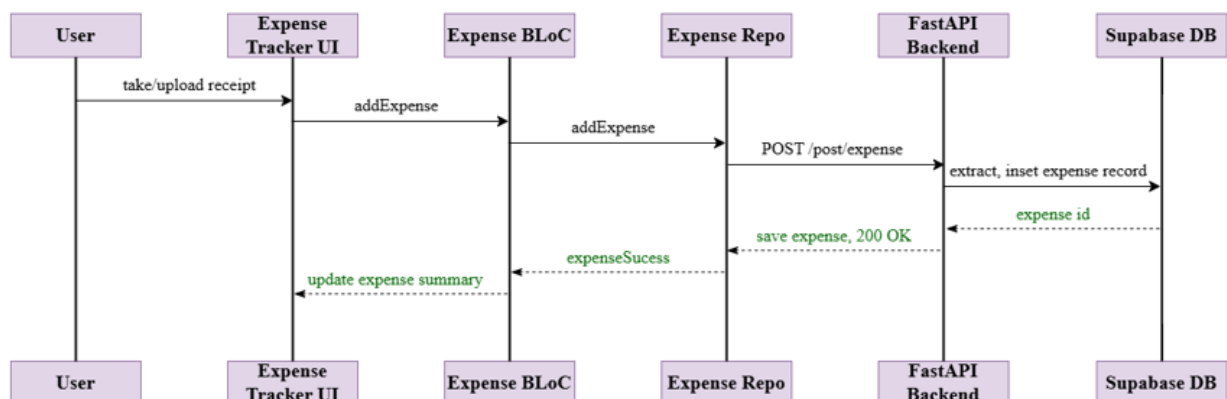


Figure 4.3.7 Pet Expense Tracker Sequence Flow Diagram

When a user takes or uploads a receipt, the Expense Tracker UI fires an addExpense event through the Expense BLoC and Repo, which sends a POST /post/expense request to the FastAPI backend. The backend extracts the relevant receipt's details and inserts an expense record into Supabase DB, returning the generated expense ID on success. The confirmation is sent back to the Expense BLoC for update in their respective UI. The app then recalculates total expenses based on the new expense record created.

4.3.8 Pet Community Interaction Sequence Flow

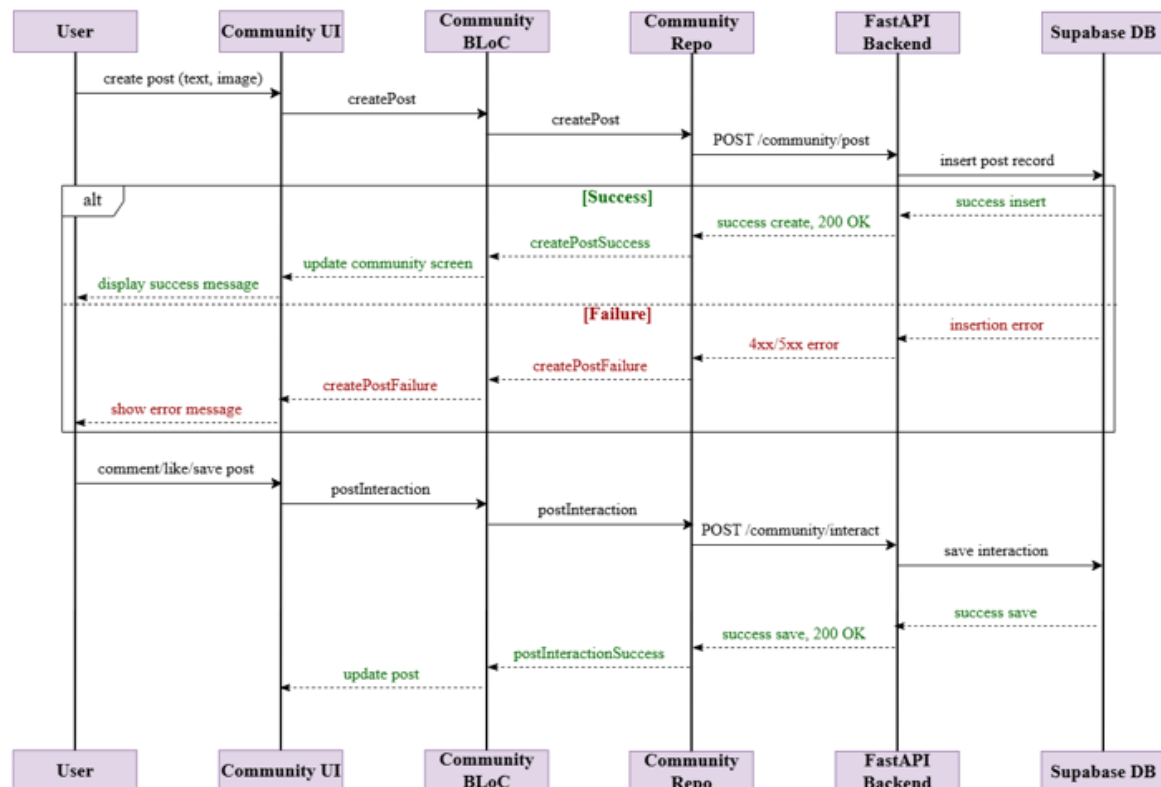


Figure 4.3.8 Pet Community Interaction Sequence Flow Diagram

Users can create posts with content and optionally images by triggering a createPost event, which travels through the Community BLoC and Repo before hitting POST /community/post on the FastAPI backend. The backend gets the post and adds it to the Supabase DB. Once it has successfully been inserted, the user will get a success message and the community will update to the latest content. Alternatively, if an error occurred during the insert, a new post will not be displayed and the user will receive the error message upon the failure of insertion. In addition to posting, users can also interact with existing community posts by liking, commenting or saving them, which sends a postInteraction event through the same process to

POST /community/interact. The database will save the response for interaction and update the state of the post on the frontend, if the interaction was successful.

4.3.9 Price Tracker Sequence Flow

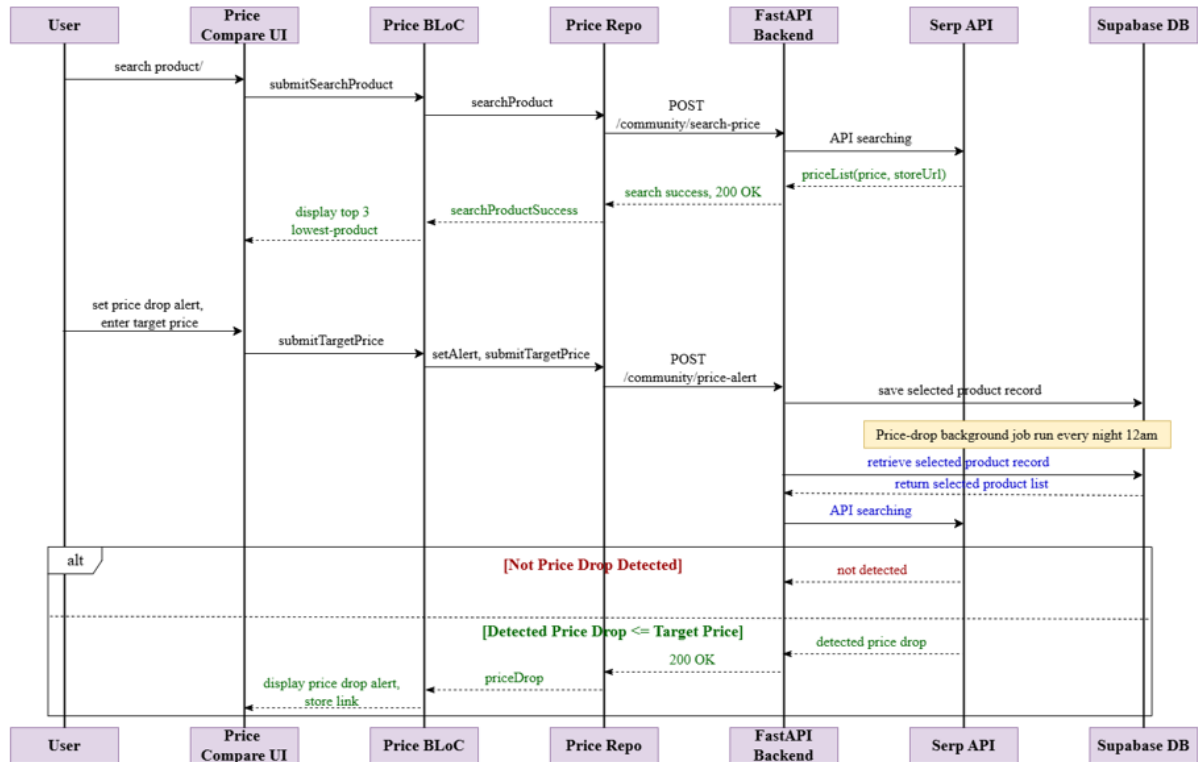


Figure 4.3.9 Price Tracker Sequence Flow Diagram

When a user searches for a product, the Price BLoC routes the request through the Price Repo to the FastAPI backend. The Serp API integrated is responsible for searching by query and finding the current lowest price of a product from amongst the different retailers or stores. The UI then displays a top 3 lowest-price list result for the user reviewing. After getting the result, users can set a price drop alert for the specific product and provide a target price they would like to purchase at. In the background, a periodic job continuously searches for the lowest prices for the selected product daily at night time. Whenever a price drop is detected, the app automatically triggers a notification delivered directly to the user.

4.4 Model Architecture

4.4.1 Sound Interpretation - Hybrid CNN + BiLSTM Model

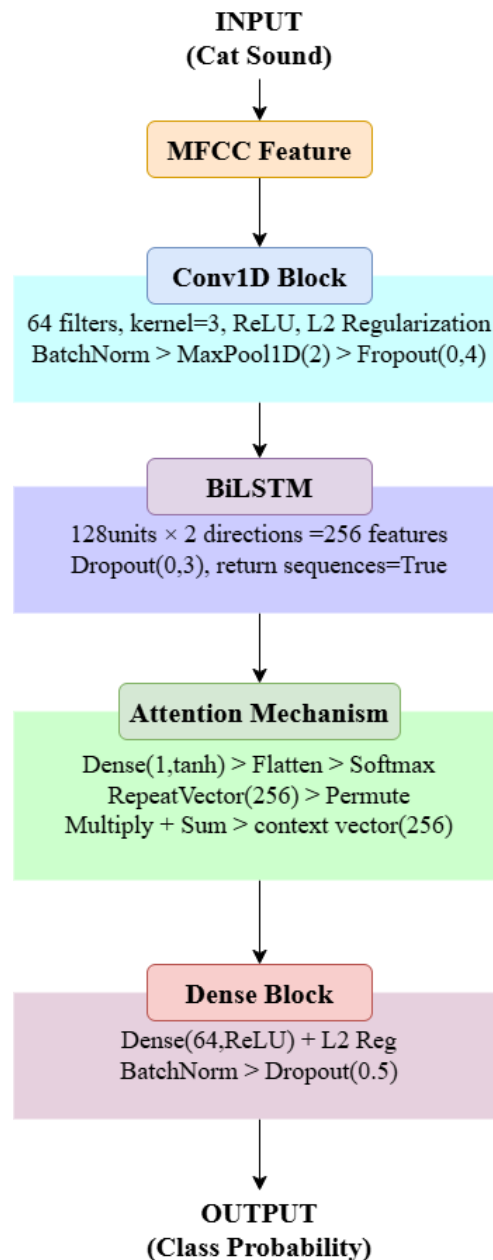


Figure 4.4.1 Sound Model Architecture Diagram

The sound interpretation model is one of the core components within the pet behaviour interpretation app, responsible for automatically classifying cat audio recordings into predefined sound categories such as happy and angry. This model accepts raw audio files and returns a predicted class together with a confidence score, enabling downstream components to act on the recognised sound event in real-time. Based on Figure 4.4.1, the model adopts a hybrid deep learning architecture that combines Convolutional Neural Networks (CNN),

Bidirectional Long Short-Term Memory (BiLSTM) and an Attention Mechanism. This combination is chosen to leverage the strengths of each component [29].

First of all, before entering the network, raw audio signals are first converted into Mel-Frequency Cepstral Coefficients (MFCCs) that mimic the human auditory system by mapping frequencies onto a mel scale [29]. Each audio is represented as a matrix shape (130,40) where 130 is the number of time frames and 40 is the number of MFCC coefficients per frame. This shape is chosen to balance temporal resolution against computational cost and to accommodate the typical duration of the target sound events. Besides that, Data Augmentation technique is used to increase dataset size and improve model generalisation. Normalisation is also applied to ensure that the model is stable and easily converges.

After completed data preprocessing, the model is built in a Conv1D layer with 64 filters and a kernel size of 3. Convolution helps the network to detect local patterns such as onset transients, harmonic bursts and rhythmic textures which are diagnostic of a particular sound category. To prevent overfitting, L2 weight regularisation is applied to convolutional kernels on small datasets by penalising large weights. In addition, Batch Normalisation is also applied immediately after the Conv1D to stabilise the distribution of activations across mini-batches. MaxPooling1D halves the temporal dimension, reducing the sequence length passed to the recurrent layers and acting as a translational invariance operator over short time offsets.

The sequence of audio is one of the important pieces of information for training the classification model. Therefore, to capture long-range temporal dependencies in both directions, the sequence is fed to a Bidirectional LSTM with 128 units in each direction, producing 256 features per time step [30]. Standard LSTMs only process the sequence forward in time, while BiLSTM allows to process each time step in both past and future context [32]. This is valuable for sounds where discriminative content may occur mid-event. It retains or discards information across long sequences, addressing the vanishing gradient problem.

Furthermore, not all time steps in an audio recording are equally informative. For example, silent padding frames or low-energy background noise frames carry little discriminative information. Hence, an Attention Mechanism is built to solve this problem by learning a scalar importance weight for each time step [31]. This mechanism allows the model to focus on the most informative temporal regions automatically, without requiring manual segmentation of audio events. Finally, the attention output is passed into a fully connected Dense layer with 64

units and ReLU activation, followed by Batch Normalisation and Dropout to improve the ability to generalise. The final output layer uses Softmax activation to produce a probability distribution over all sound categories. The predicted class is obtained by taking the argmax of this distribution.

This hybrid architecture provides multiple benefits when interpreting sounds. The CNN layer processes local features in an effective way, BiLSTMs capture time dependencies in a forward and backward manner and the attention mechanism focuses on the most pertinent areas of the audio signal. Combining all of these techniques together results in greater accuracy and robustness than using one method in isolation. Data augmentation and regularisation techniques also increase generalisation and reduce overfitting. Thus, this proposition will enable reliable, customised identification of pet vocalisations through the app. It helps to develop future opportunities in interpreting the sounds of pets in a real-world setting.

4.4.2 Visual Interpretation - MobileNetV2 Pretrained Model

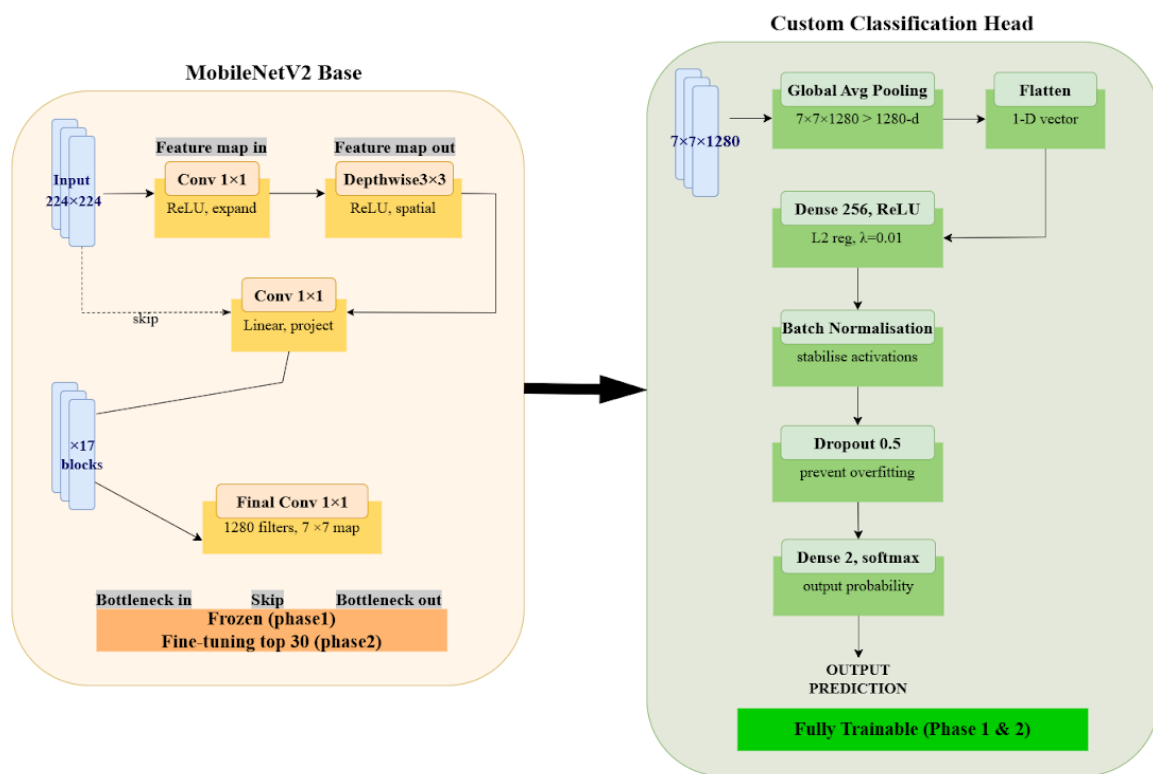


Figure 4.4.2 Visual Model Architecture Diagram

The visual interpretation model forms another core component within the multimodal interpretation feature, responsible for automatically classifying cat body language into predefined categories such as friendly_active and aggressive_defensive. This visual

classification model is built on transfer learning using MobileNetV2 as the pretrained backbone. This is because a model pretrained on ImageNet already encodes rich low-level features that can be generalised well to any visual classification task including behaviour recognition. Fine-tuning only the upper layers on domain-specific data allows the model to adapt to the target classes without requiring millions of labelled data [33].

The full architecture is divided into two logical blocks which are the frozen MobileNetV2 base and a trainable custom classification head. First, the MobileNetV2 is a lightweight convolutional network designed for mobile and embedded apps, which is very suitable for this project [34]. Its core innovation is the inverted residual block with a linear bottleneck. Each block first expands the channel dimension with a pointwise convolution and applies a depthwise separable convolution to extract spatial features with minimal parameters. Then, project them back to a smaller channel dimension. This design provides a strong accuracy to latency trade-off compared to standard convolutions [2]. The base accepts a $224 \times 224 \times 3$ RGB input and produces a $7 \times 7 \times 1280$ feature map after 17 inverted residual blocks and a final 1×1 convolution.

Second, a Custom Classification Head is stacked on top of the base. A Global Average Pooling (GAP) layer collapses 7×7 spatial dimensions into a single 1280-dimensional vector, discarding positional information and dramatically reducing parameter count compared to a Flatten layer [35]. This is followed by a fully connected Dense layer with 256 neurons, ReLU activations and L2 regularization to penalise large weights. Batch Normalisation accountable for stabilising the activations after each mini-batch and a Dropout layer with rate 0.5 randomly deactivates half the neurons during training to prevent coadaptation [36][37]. The final output layer contains two neurons with Softmax activation, producing a probability distribution over the two target classes.

Both models are training in two phases. In phase 1, the MobileNetV2 base is entirely frozen and only the Custom Classification head is trained using the Adam optimiser at the default learning rate. This allows the head to converge on meaningful features before any base weights are disturbed [38]. In phase 2, the top 30 layers of the MobileNetV2 base are unfrozen and the entire network is fine-tuning jointly at a much lower learning rate to adapt the high-level feature detectors to cat body language images without catastrophic forgetting of ImageNet knowledge [38].

To sum up, this model uses the MobileNetV2 network as a backbone that has been trained and uses a lightweight custom classification head to classify the body language of cats into predefined classes. The use of transfer learning enables strong feature extraction with limited training data, while the two-phase training strategy improves performance by first training the classifier head and then fine-tuning selected layers for better domain adaptation. Therefore, this app strikes a good balance between accuracy, computational efficiency and generalisation, making it suitable for use in real-time or other resource-constrained applications.

4.5 Implementation Issues and Challenges

One of the challenges is the limited availability of training data. The cat sound dataset and the visual body language dataset are small. This made it difficult for the models to generalise well beyond the training examples, with the risk of overfitting being a constant concern throughout the entire development process. Besides that, the dual-stream video interpretation pipeline introduced a noticeable processing delay that affected the user experience, especially on slower mobile connections. Every video submission required the backend to simultaneously extract audio, process frames and run two deep learning models before producing a result. This makes the total inference time considerably longer than a single-model approach would have required.

Furthermore, video file size turned out to be a more practical problem than initially anticipated. Users recording pet videos on modern smartphones often produced files in the range of 50–150MB, causing slow upload speeds over mobile data and adding significant delay before backend processing could even begin. Throughout development and testing, the system's dependence on multiple external services introduced a layer of fragility. Supabase, Gemini API and the ngrok tunnel were all critical components, and any one of them experiencing downtime or latency spikes would cascade into visible failures within the app.

Overall, the challenges encountered reflect the practical realities of building an AI-powered mobile application on a small-team, limited-resource development cycle. Data scarcity, inference latency, file size constraints and external service dependencies are all problems that exist in real production systems and working through each of them produced a more robust and thoughtfully designed application.

4.6 Summary

This chapter outlines the overall design of the pet care app, including system workflow, database structure and module interactions. After going through the user authentication process, the application provides several functions like food and health analysis, emotion interpretation, reminders, expense tracking, community interaction and price alert system based on frontend to backend workflow that is strictly defined.

The Supabase application will manage the data while BLoC, repository and API layers connect the components together. In the emotion interpretation model design section, the audio will be classified as a hybrid CNN-BiLSTM model and the visual will be classified as a MobileNetV2 model. This will result in a highly efficient, highly scalable and intelligent personalised pet care solution at the end of the design process.

CHAPTER 5: System Implementation and Testing

5.1 Setting and Configuration

5.1.1 Frontend Setup

The frontend is built with Flutter and Dart, with Visual Studio Code serving as the main development environment. The Flutter and Dart extensions in VS Code made day-to-day tasks like debugging and formatting much smoother. For dependency management, all external packages, including `flutter_bloc` for state management and `http` for network requests, are declared and tracked through the `pubspec.yaml` file. To keep the codebase manageable as the project grows, the folder structure is organised following the Clean Architecture principle. This separates concerns into distinct layers such as screens, services and bloc, while also isolating reusable modules like `packages/user_repository` into their own directories.

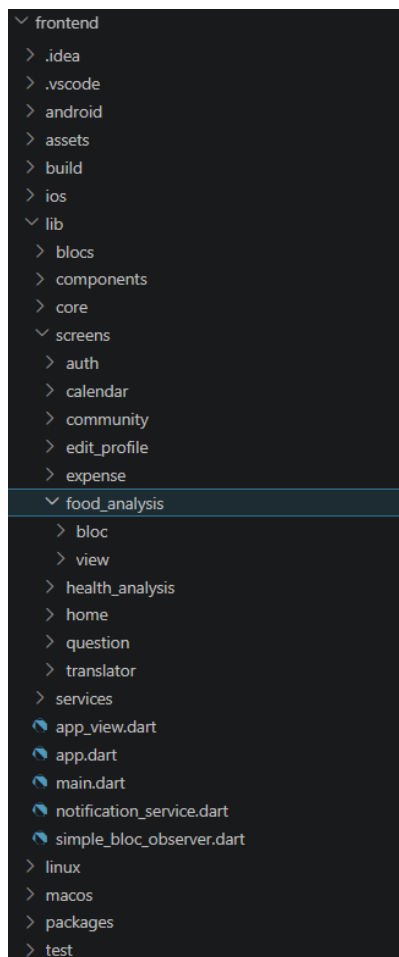


Figure 5.1.1

Screenshot of IDE's File Explorer



Figure 5.1.2

Screenshot of pubspec.yaml File

5.1.2 Backend Setup

On the backend side, Python is chosen as the development language, with FastAPI as the framework. FastAPI is a natural fit because it is able to handle synchronous operation when dealing with AI models and complex business logic. Before writing any code, a Python virtual environment is set up to keep the project's dependencies self-contained and avoid any conflicts with other packages on the app. All required libraries including fastapi, unicorn and other relevant machine learning packages are listed in a requirements.txt file so the environment could be reproduced consistently. During development, the server was run locally using Unicorn which supports live reloading on code changes. A particularly useful side effect of using FastAPI with Uvicorn is the auto-generated Swagger UI at /docs, which makes it easy to test each endpoint interactively without needing a separate tool.

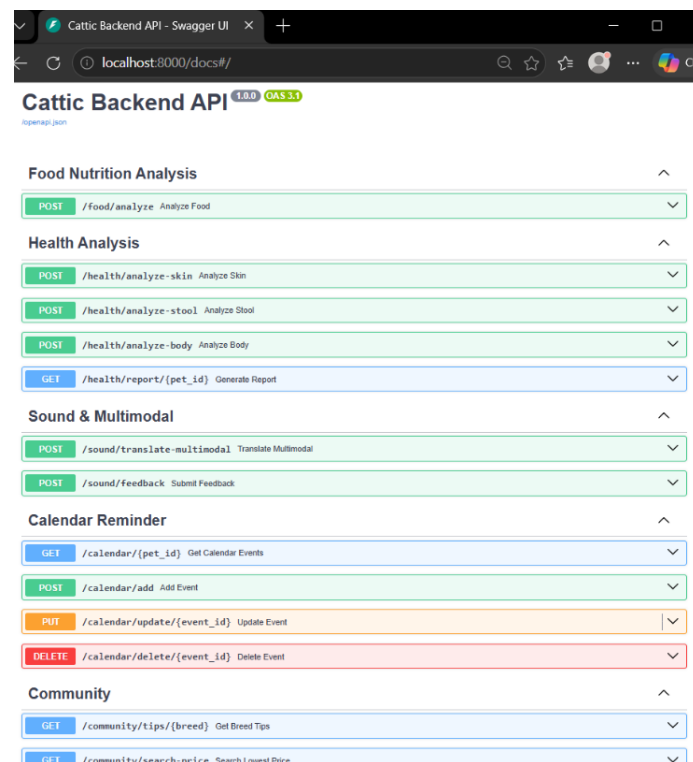


Figure 5.1.3 FastAPI Swagger UI

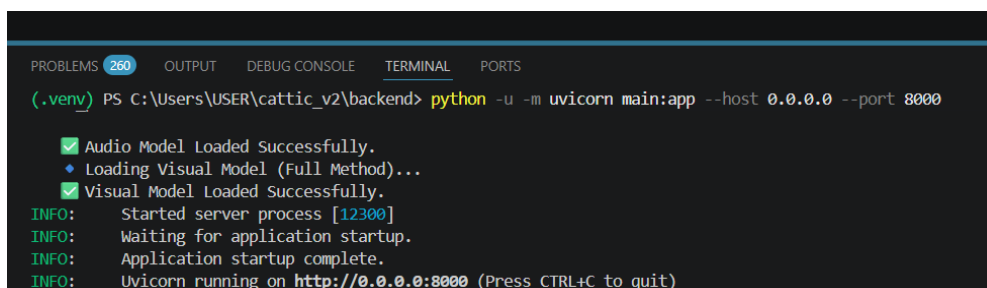


Figure 5.1.4 Screenshot of Showing Uvicorn Terminal Run Successfully

5.1.3 Database Setup

For the database and cloud services, the project relied on Supabase as a Backend-as-a-Service (BaaS) platform. Supabase handled three main responsibilities which are PostgreSQL database, user authentication and file storage. Setup started by creating a new project on Supabase dashboard and pulling the project URL and anonymous API key into the application’s environment configuration.

The database schema is created by running SQL scripts that define core tables such as UserProfile, Pet and HealthLog. Each of these tables is mapped directly to the project's Entity-Relationship Diagram. Beyond the relational tables, Supabase’s Object Storage is also configured with separate buckets to handle different types of media such as pet profile image, pet sound video and expense receipt.

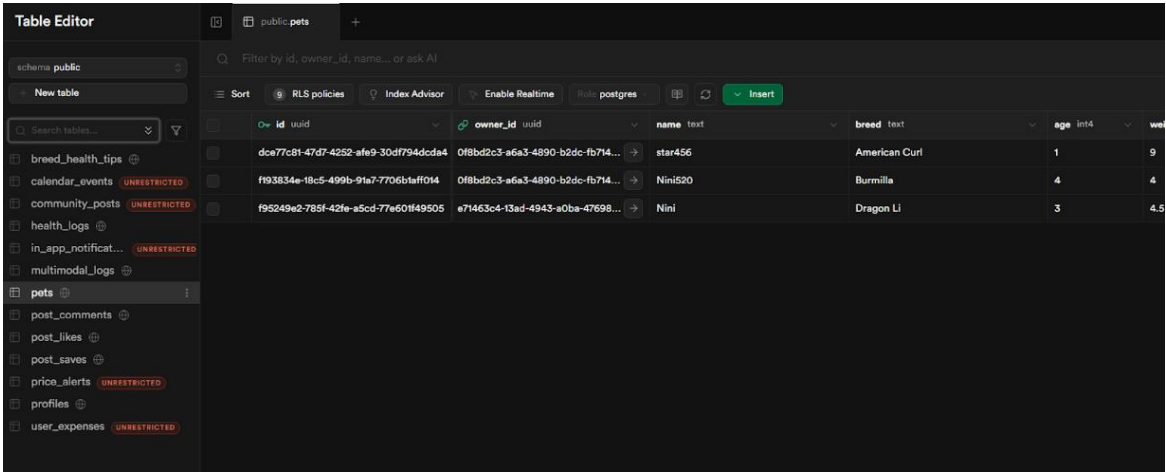


Figure 5.1.5 Screenshot of Pet Table

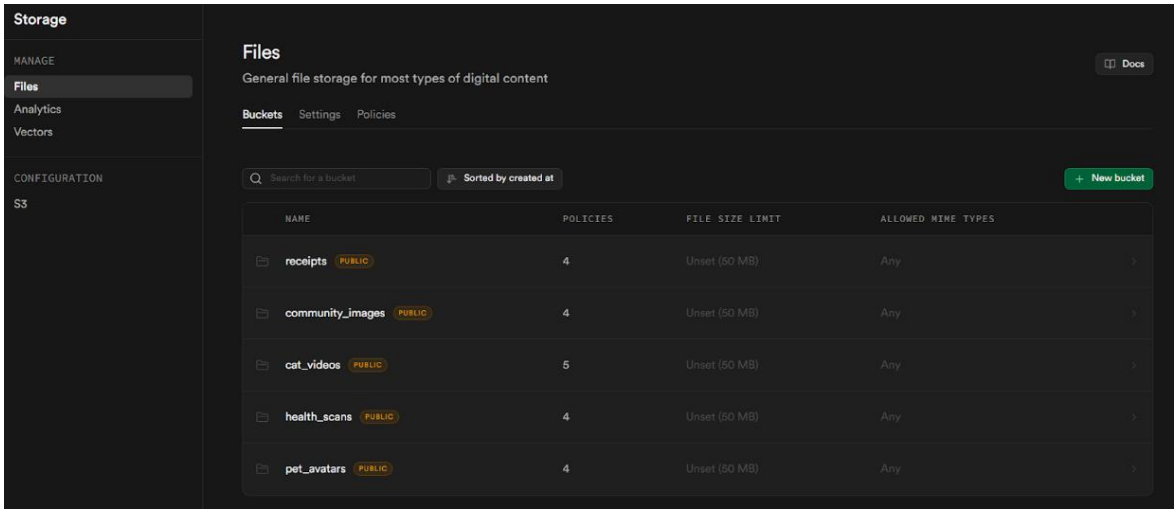


Figure 5.1.6 Screenshot of Supabase Storage Dashboard

5.1.4 Communication Bridge – Ngrok

One practical issue that came up early in development is getting the mobile frontend to talk to the backend. When the FastAPI server is running on localhost, a physical device on a different network is not able to communicate directly. Therefore, a reverse proxy tool named Ngrok is used to create a public HTTPS tunnel pointing to the local server port. Running ngrok generates a temporary URL that forwards traffic from the internet straight to the local FastAPI instance. This URL is then dropped into the Flutter app's API service layer as the base endpoint. This setup meant the full request-response cycle from the mobile app through the tunnel to the local backend and back. It worked seamlessly during testing, without needing to push the backend to any cloud infrastructure prematurely.



```

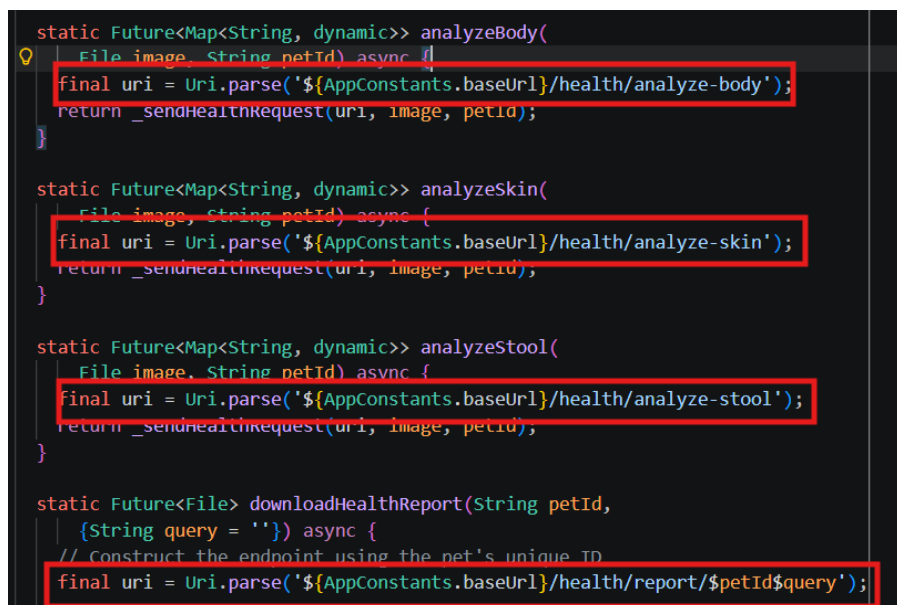
ngrok

One gateway for every AI model. Available in early access *now*: https://ngrok.com/r/ai

Session Status      online
Account             Xinyi (Plan: Free)
Version             3.36.0
Region              Asia Pacific (ap)
Latency             27ms
Web Interface       http://127.0.0.1:4040
Forwarding           https://cometary-nonaltruistically-deja.ngrok-free.dev -> http://localhost:8000

Connections          ttl    opn    rt1    rt5    p50    p90
                   0      0      0.00   0.00   0.00   0.00
  
```

Figure 5.1.7 Screenshot of Active Ngrok Session



```

static Future<Map<String, dynamic>> analyzeBody(
  File image, String petId) async {
  final uri = Uri.parse('${AppConstants.baseUrl}/health/analyze-body');
  return _sendHealthRequest(uri, image, petId);
}

static Future<Map<String, dynamic>> analyzeSkin(
  File image, String petId) async {
  final uri = Uri.parse('${AppConstants.baseUrl}/health/analyze-skin');
  return _sendHealthRequest(uri, image, petId);
}

static Future<Map<String, dynamic>> analyzeStool(
  File image, String petId) async {
  final uri = Uri.parse('${AppConstants.baseUrl}/health/analyze-stool');
  return _sendHealthRequest(uri, image, petId);
}

static Future<File> downloadHealthReport(String petId,
  {String query = ''}) async {
  // Construct the endpoint using the pet's unique ID
  final uri = Uri.parse('${AppConstants.baseUrl}/health/report/$petId$query');
}
  
```

Figure 5.1.8 Example of Base URL Setup

5.2 System Operation

5.2.1 Sign In, Sign Up and Question Screen Development

Before using the application, users will be redirected to the Sign-In screen, where they are required to enter their email and password. When typing, the password will be hidden by default. Users can tap on the Eye icon on the right side to have a full view of the password. After entering the credentials and clicking Sign In Button, Supabase Authentication will validate the existing user. If successfully validated, the app will navigate users to Home Screen. However, if a red error message appears, it means the email or password is incorrect. Users have to retype the accurate email or password.

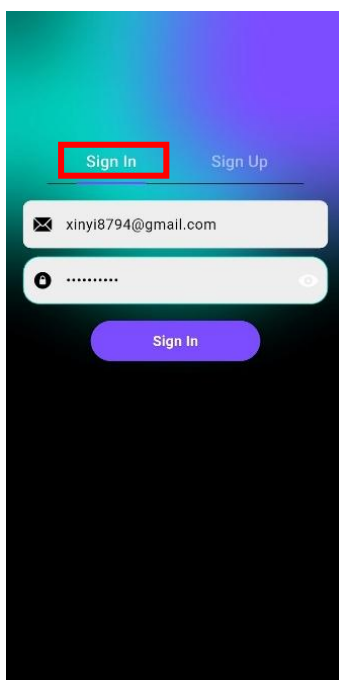


Figure 5.2.1 Sign In Screen

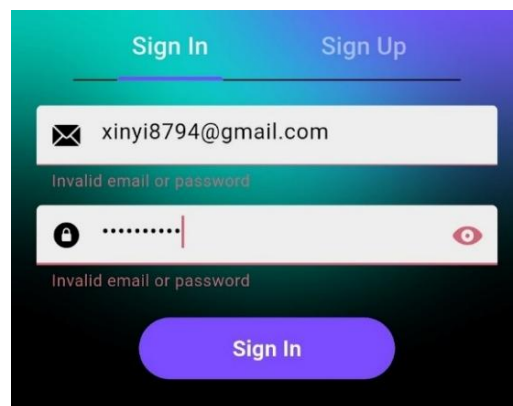


Figure 5.2.2 Error Message Appears When Email or Password Incorrect

```
class SignInBloc extends Bloc<SignInEvent, SignInState> {
  final UserRepository _userRepository;

  SignInBloc(this._userRepository) : super(SignInInitial()) {
    // Sign In
    on<SignInRequired>((event, emit) async {
      emit(SignInProcess());
      try {
        await _userRepository.signIn(event.email, event.password);

        emit(SignInSuccess());
      } catch (e) {
        emit(SignInFailure());
      }
    });
    // Sign Out
    on<SignOutRequired>((event, emit) async => await _userRepository.logOut());
    on<SignOutRequired>((event, emit) async {
      await _userRepository.logOut();
      emit(SignOutSuccess()); // <-- new state
    });
  }
}
```

Figure 5.2.3 Sign In Bloc

As new users, the option to switch to Sign Up Screen is available in order to create a new account by entering an email and password. For the password, users are forced to fulfil a strict format to ensure the security of the app. If the entered password fully follows the format, all the reminders will become green. Otherwise, the missing format reminders will remain white colour if the entered password has missing requirements. Once both fields are completed correctly, users can click on the Sign Up Button to enter Home Screen.

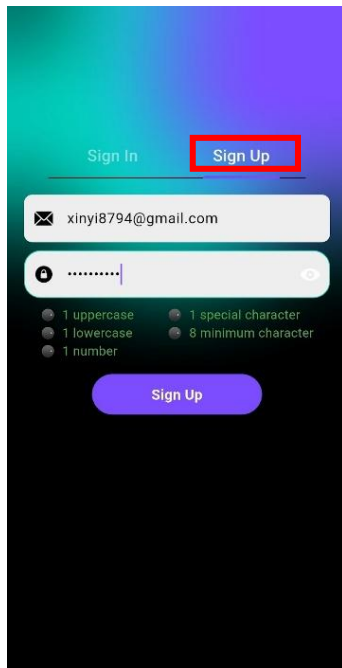


Figure 5.2.4 Sign Up Screen When Password Format Correct

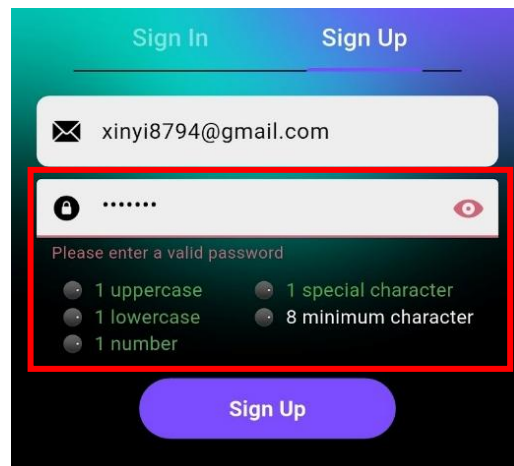


Figure 5.2.5 Format Reminder Remain White Colour If Password Format Wrong

```
class SignUpBloc extends Bloc<SignUpEvent, SignUpState> {
  final UserRepository _userRepository;

  SignUpBloc(this._userRepository) : super(SignUpInitial()) {
    on<SignUpRequired>((event, emit) async {
      emit(SignUpProcess());
      try {
        // 1. Sign Up (Auth)
        // Returns a user with the new valid Supabase ID
        await _userRepository.signUp(event.user, event.password);

        // 1. Just Sign Up (Auth only)
        await _userRepository.signUp(event.user, event.password);

        // 2. Emit Success immediately so we navigate to Question Screen
        emit(SignUpSuccess());

        emit(SignUpSuccess());
      } catch (e) {
        print("=== SIGN UP ERROR ===");
        print("Error Type: ${e.runtimeType}");
        print("Error Message: $e");
        print("Stack Trace: ${StackTrace.current}");
        emit(SignUpFailure());
      }
    });
  }
}
```

Figure 5.2.6 Sign Up Bloc

However, before viewing the Home Screen, new users will be brought to the Question Screen to answer 5 questions:

1. Pet Name
2. Pet Breed
3. Pet Age
4. Pet Weight
5. Pet Allergy

In the pet age question page, users can select the age of pet either in years or in months. If the pet has allergies, the user can tick the provided checkbox and type the name of the allergenic food. Each page will show the progress of the question screen and users can tap either Next Button or Back Button to navigate to the page. At the end, there will be a Submit Button. After clicking on the Submit Button, the account will be successfully created, and the pet details will be saved in Supabase Database for future use. Once done the creation, new users will be navigated to Home Screen.

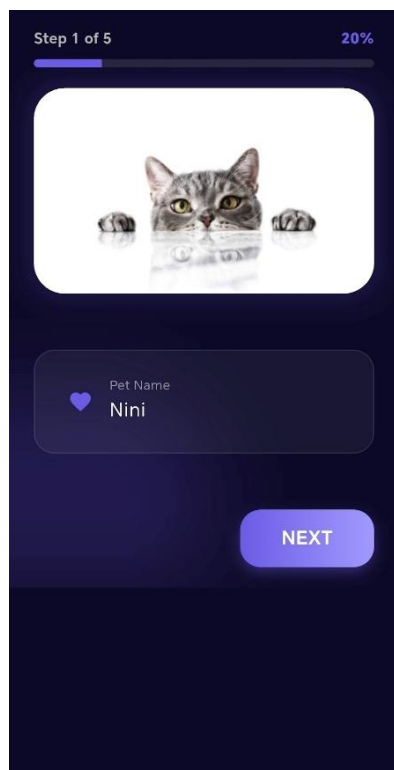


**Figure 5.2.7 First Question Screen
(Pet Name)**

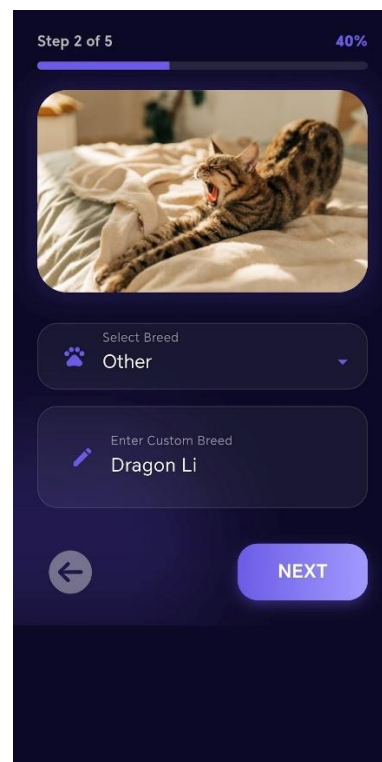


**Figure 5.2.8 Second Question Screen
(Pet Breed)**

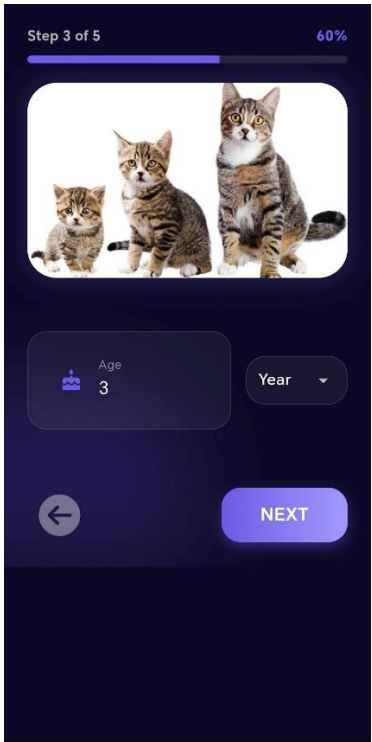


Figure 5.2.9
Third Question Screen
(Pet Age)

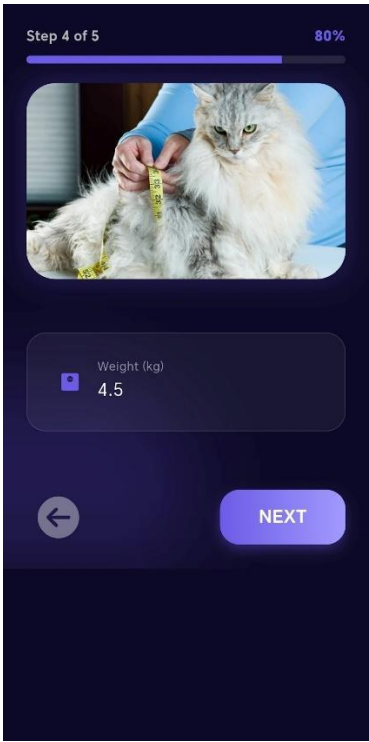


Figure 5.2.10
Fourth Question Screen
(Pet Weight)

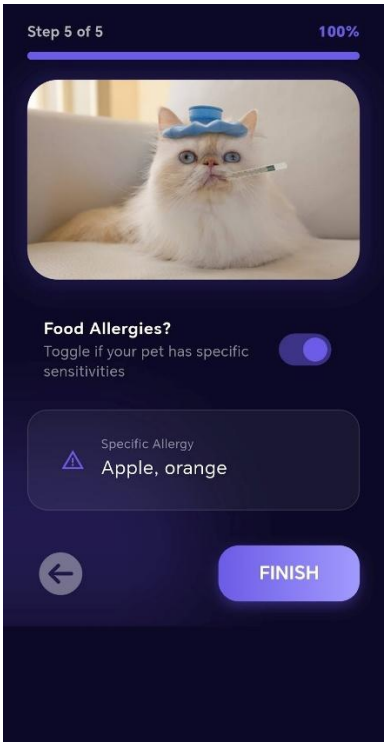


Figure 5.2.11
Fifth Question Screen
(Pet Allergy)

Authentication	Users
MANAGE	Q Email address Search by email All columns Sorted by user ID Add user
Users	UID Display name Email Phone Providers Provider type
OAuth Apps	0f8bd2c3-e6a3-4890-b2dc-fb71488f829e - test@example.com - Email -
NOTIFICATIONS	e71463c4-13ad-4943-a0ba-476989a103ab - xinyi8794@gmail.com - Email -
Email	
CONFIGURATION	
Policies	
Sign In / Providers	
OAuth Server	
Sessions	
Rate Limits	
Multi-Factor	
URL Configuration	
Attack Protection	

Figure 5.2.12 Account Successfully Created in Supabase Authentication

5.2.2 Home Screen and Logout Feature Development

On the Home Screen, users can view their pet’s details displayed at the top of the screen. A Switch selection is shown in the pet details section. For users that have multiple pets, they can tap it to add another pet or switch to another pet profile. The Home Screen also presents six feature cards: Food Nutrition Analysis, Health Analysis, Emotion Interpretation, Pet Reminder,

Expense Tracker and Edit Profile. Tapping any card will navigate users to the corresponding screen. At the bottom navigation bar, there will also be a community selection developed. Users can click on it or swap to the right to enter the community platform. To logout, users can tap the Exit icon at the top right corner of the Home Screen.

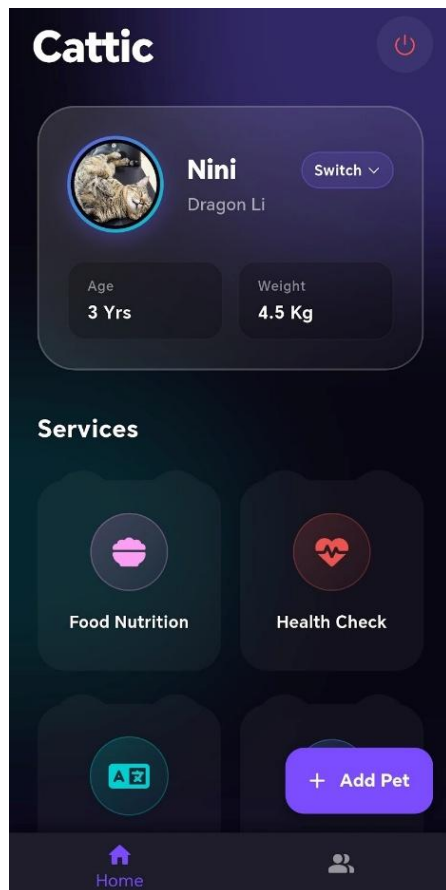


Figure 5.2.13 Home Screen

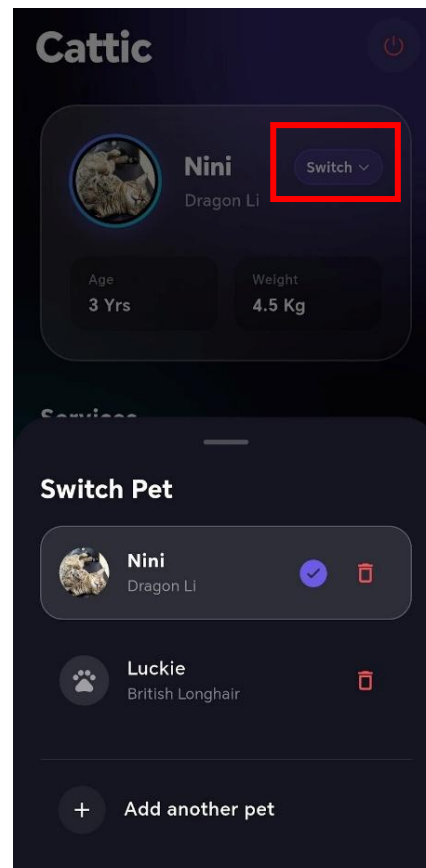


Figure 5.2.14 Switch to Another Pet

5.2.3 Pet Profile Edit Feature Development

To update pet details, users can also press on the Edit Data on the Home Screen. After clicking, the app will navigate users to the Edit Profile Screen. In this screen, users are able to edit their pet information and click on Save Changes Button to save the latest details in Supabase Database. Once updated successfully, there will be a success message. All the fields should not be empty, except the pet allergy's field. If there is an empty field, the save button will not be able to click. To exit this page, users can click on the Cancel button or return icon.

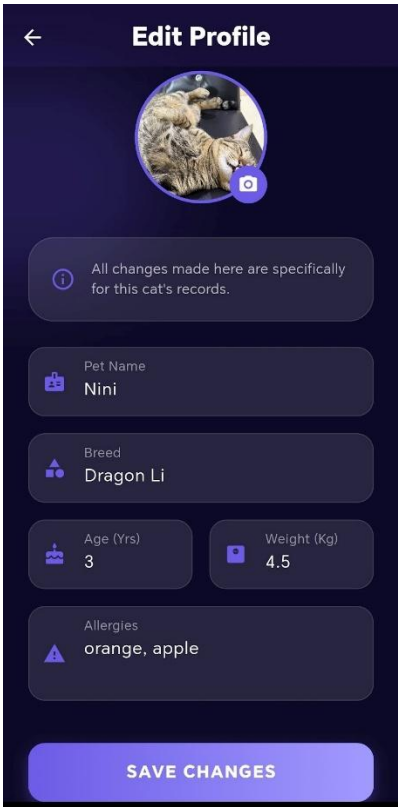


Figure 5.2.15 Edit Profile Screen

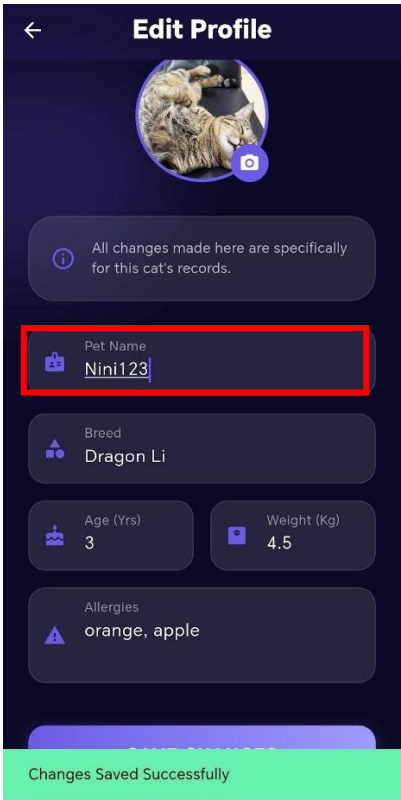


Figure 5.2.16
Successfully Edit Message

The screenshot shows a database interface with the 'public.pets' table selected. The table has columns for id, name, breed, age, weight, and allergies. The row for the pet named 'Nini' is highlighted with a red box.

id	name	breed	age	weight	allergies
c62affad-29e8-438c-b1cc-a1e48c038544	Luckie	British Longhair	2	3.5	["Apple"]
dce77c81-47d7-4252-afe9-30df794dcd4	star456	American Curl	1	9	["milk"]
f193834e-18c5-499b-91a7-7706b1aff014	Nini520	Burmilla	4	4	["apple"]
f95249e2-785f-42fe-a5cd-77e60ff49505	Nini	Dragon Li	3	4.5	["orange","apple"]

Figure 5.2.17 Pet Name Before Change in Database

The screenshot shows the same database interface as Figure 5.2.17, but the pet name has been updated to 'Nini123'. The row for 'Nini123' is highlighted with a red box.

id	name	breed	age	weight	allergies
c62affad-29e8-438c-b1cc-a1e48c038544	Luckie	British Longhair	2	3.5	["Apple"]
dce77c81-47d7-4252-afe9-30df794dcd4	star456	American Curl	1	9	["milk"]
f193834e-18c5-499b-91a7-7706b1aff014	Nini520	Burmilla	4	4	["apple"]
f95249e2-785f-42fe-a5cd-77e60ff49505	Nini123	Dragon Li	3	4.5	["orange","apple"]

Figure 5.2.18 Pet Name After Change in Database

5.2.4 Pet Food Nutrition Analysis Feature Development

To analyse pet food, users can enter the Pet Food Nutrition Screen by clicking on “Food Nutrition” in Home Screen. After clicking this feature, the app will display a Pet Food Nutrition Analysis Screen to users. On this screen, users are able to choose whether to take or upload an image for analysis. The uploaded images can be categorized into two categories which are commercial food and homemade food. Once uploaded, Gemini API will process the image and generate results for users. The analysis will provide four sections of information depending on each pet’s details, which include food description, suitability, nutrition facts and recommendations. To enhance the accuracy of nutritional facts of commercial food, a file named “catfooddb_extend.json” was created and collected. If the app detects commercial food, it will match the name of the food and retrieve the nutritional information from the file.

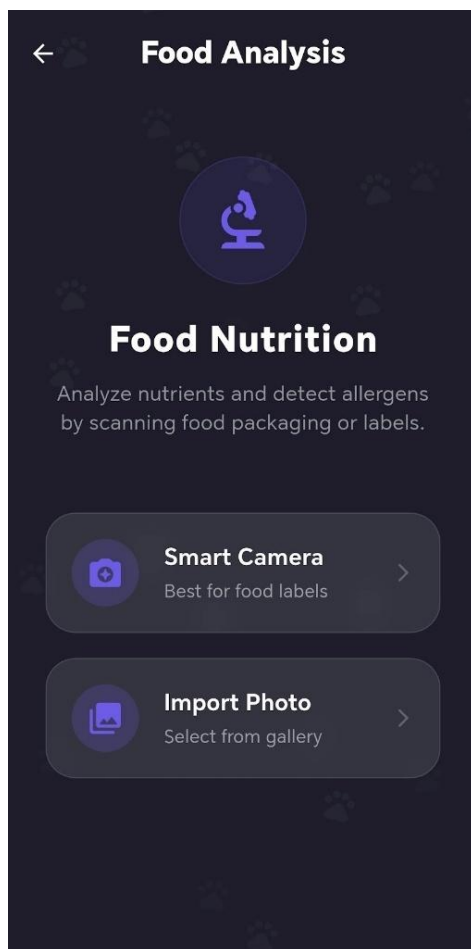


Figure 5.2.19

Food Nutrition Analysis Screen

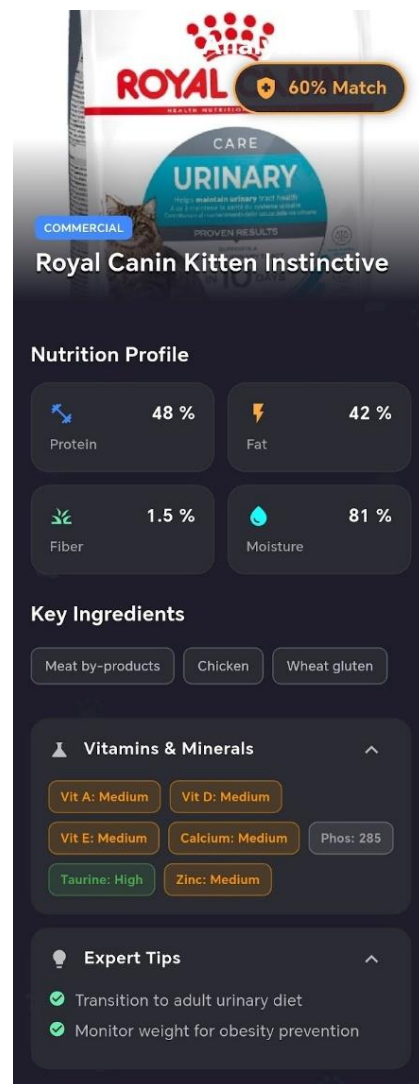


Figure 5.2.20 Food Analysis Result

```

Future<void> _performAnalysis(
    File imageFile,
    String userId,
    Emitter<FoodAnalysisState> emit,
) async {
    try {
        // Initial uploading state
        emit(FoodAnalysisUploading(
            progressMessage: "Uploading image...",
        ));
        // Simulate upload progress (optional enhancement)
        await Future.delayed(const Duration(milliseconds: 500));
        emit(FoodAnalysisUploading(
            progressMessage: "Analyzing food composition...",
        ));
        // Call API with retry mechanism
        final result = await ApiService.analyzeFood(
            imageFile,
            userId,
            maxRetries: 2, // Will try up to 2 times total
        );
        // Success - emit result
        emit(FoodAnalysisSuccess(result));
    } on TimeoutException catch (e) {
        print("${e.message}");
        emit(FoodAnalysisFailure(
            "⚠️ Analysis timed out\n\n"
            "Please try:\n"
            "• Taking a clearer, well-lit photo\n"
            "• Checking your internet connection\n"
            "• Moving to a location with better signal\n"
            "• Trying again in a moment",
            canRetry: true,
        ));
    } on SocketException catch (e) {

```

Figure 5.2.21

Image Uploading State Transition

```

@router.post("/analyze")
async def analyze_food(
    file: UploadFile = File(...),
    pet_id: str = Form(...),
):
    print(f"[REQUEST STARTED] Analyzing Food for Pet: {pet_id}")
    start_time = time.time()

    try:
        print(f"[STEP 1] Reading and Compressing Image...")
        raw_image_bytes = await file.read()
        image_bytes = compress_image(raw_image_bytes)

        print(f"[STEP 2 & 4] Fetching Pet & Identifying Image Simultaneously...")

        model = genai.GenerativeModel('models/gemini-flash-latest')
        fast_config = genai.types.GenerationConfig(
            temperature=0.0,
            response_mime_type="application/json"
        )

        id_prompt = """
        Identify if this food is "Commercial" or "Homemade".
        Rule: Extract a MAXIMUM of 3 visual ingredients.
        Return strict JSON exactly like this, with NO comments and NO trailing commas:
        {
            "type": "Commercial",
            "brand": "Brand Name",
            "variety": "Flavor",
            "visual_ingredients": ["ingredient1", "ingredient2"]
        }
        """

        async def fetch_pet():
            return await asyncio.to_thread(
                lambda: supabase.table('pets').select('id').eq('id', pet_id).single().execute()
            )

        async def identify_image():
            return await asyncio.to_thread(
                lambda: model.generate_content(
                    [image_bytes],
                    generation_config=fast_config,
                )
            )

        pet_data, response_1 = await asyncio.wait_for(
            asyncio.gather(fetch_pet(), identify_image()),
            timeout=30.0
        )

```

Figure 5.2.22

Parallel Fetching of Pet Data and Image Identification

```

# --- WEILPER: Search Commercial DB ---
def search_database(brand_query, variety_query):
    if not COMMERCIAL_DB: return None

    query = f"{brand_query} {variety_query}".strip()

    # OPTIMIZATION 1: Create a dictionary for instant exact-match lookups
    db_dict = {p['name'].lower(): p for p in COMMERCIAL_DB}

    # OPTIMIZATION 2: Check for an exact match first (Lightning Fast!)
    if query.lower() in db_dict:
        print(f" -> Exact Match Found: '{query}'")
        return db_dict[query.lower()]

    # OPTIMIZATION 3: If no exact match, filter by brand to make the pool tiny
    brand_matches = [p for p in COMMERCIAL_DB if brand_query.lower() in p['brand'].lower()]
    search_pool = brand_matches if brand_matches else COMMERCIAL_DB
    # Map the search pool names to their objects so we don't have to loop again later
    pool_dict = {p['name']: p for p in search_pool}
    # Run the rapidfuzz process
    best_match = process.extractOne(query, pool_dict.keys())

    # OPTIMIZATION 4: Lower the threshold slightly to 80 to catch more matches and skip the 40-second Gemini generation more often!
    if best_match and best_match[1] >= 80:
        match_name = best_match[0]
        print(f" -> Fuzzy Match Found: '{match_name}' (Score: {best_match[1]})")
        return pool_dict[match_name] # Instant O(1) retrieval!

    print(f" -> No database match for '{query}'")
    return None

```

Figure 5.2.23 Commercial Food Database Fuzzy Matching Function

To improve safety monitoring, the app will automatically show an alert message if the food contains harmful or allergic ingredients. It will mainly focus on detecting general cat toxic food and pet's specific allergies. As an example in Figure 5.2.24, the user's pet is allergic to apple and the uploaded image contains apple and grape, so the alert message will show apple is allergic to the user's pet and grape is dangerous to serve as a food to pet. This helps users quickly identify risks and avoid unsafe foods.



Figure 5.2.24 Allergy Alert Message



Figure 5.2.25 Allergic Food Detected

5.2.5 Pet Health Analysis Feature Development

Users can click on the Health Analysis on Home Screen to enter the Pet Health Analysis Screen. Before proceeding to taking or uploading images, the app will bring users to a selection screen in order to let users choose a desired type of health analysis. There will be three types of analysis which are pet skin analysis, pet stool analysis and pet body weight analysis. Users can

press one of them and upload the specific health images to the app. After successfully uploaded, Gemini API will process the image and provide results for users.

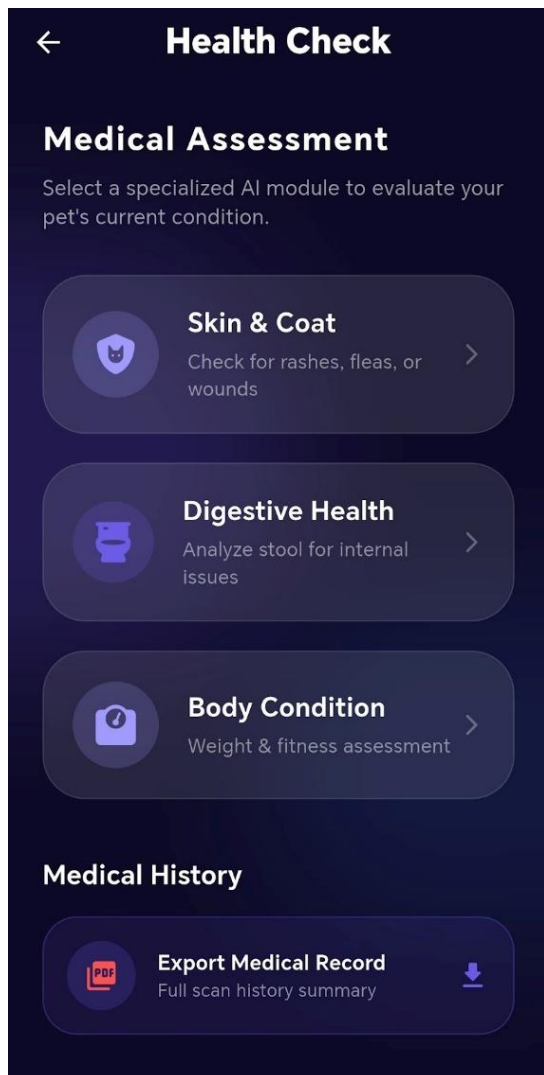


Figure 5.2.26 Health Analysis Screen

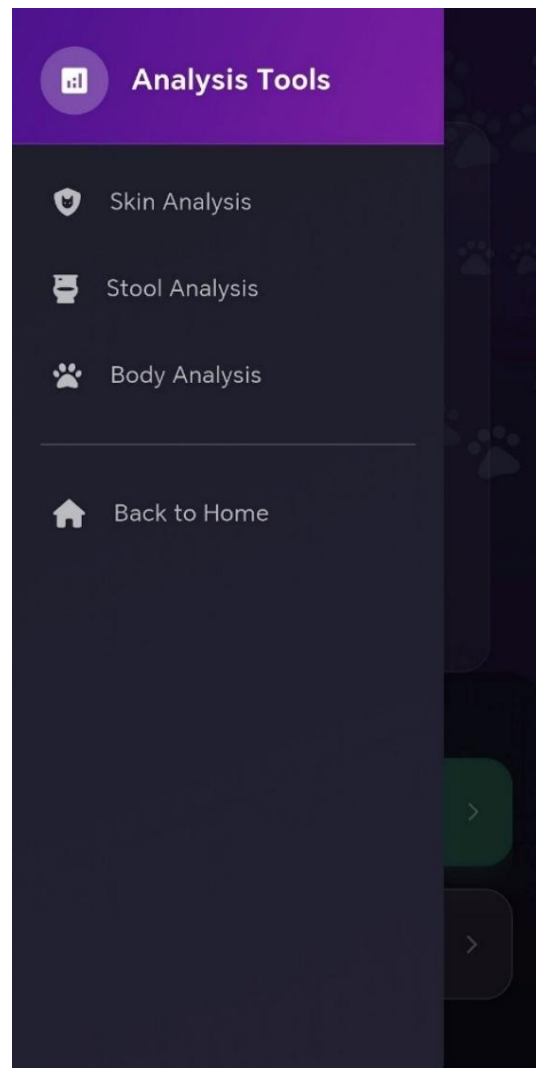


Figure 5.2.27 Navigation Menu

For skin analysis, the app will display skin condition, sign of allergy, severity level and recommendations. For stool analysis, 5 sections of information including stool colour, texture, presence of mucus, severity score and recommendations will appear after complete analysis. For body weight analysis, the app will display body condition, possible health risk and recommendation to users. The app provides previous report data as contextual input to the AI model, which may help guide the analysis's final severity score and status. After completely reviewing the results, users can click on the hamburger menu that implements at the top-left corner, navigating to another health analysis type or Home Screen.

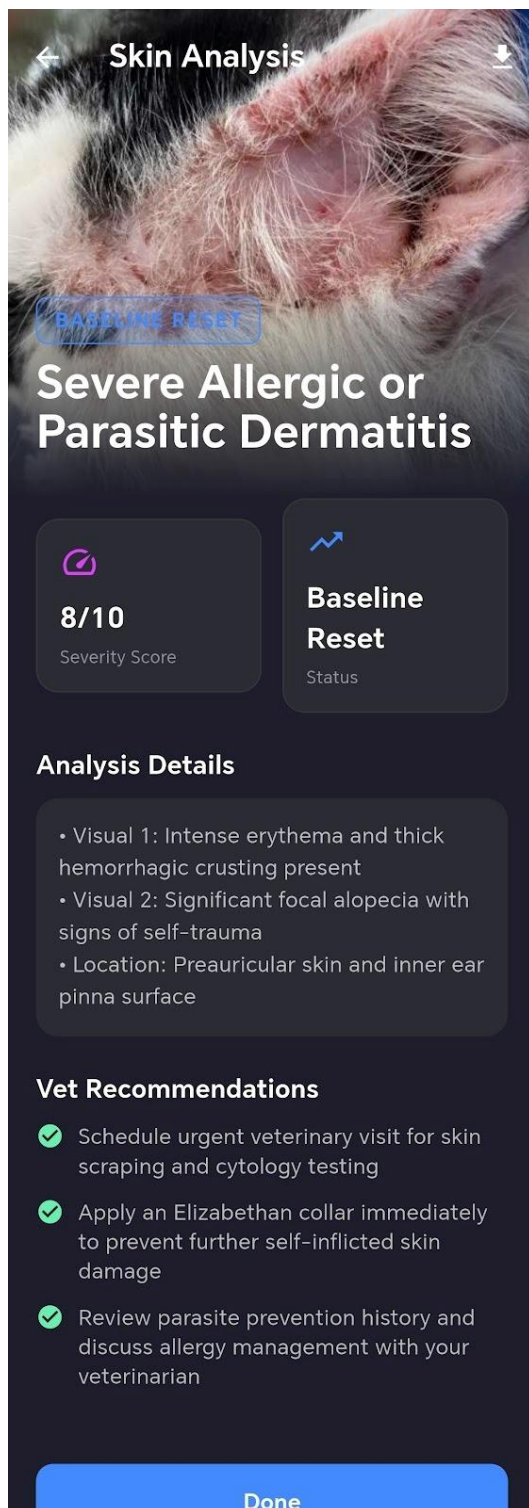


Figure 5.2.28 Skin Analysis Results

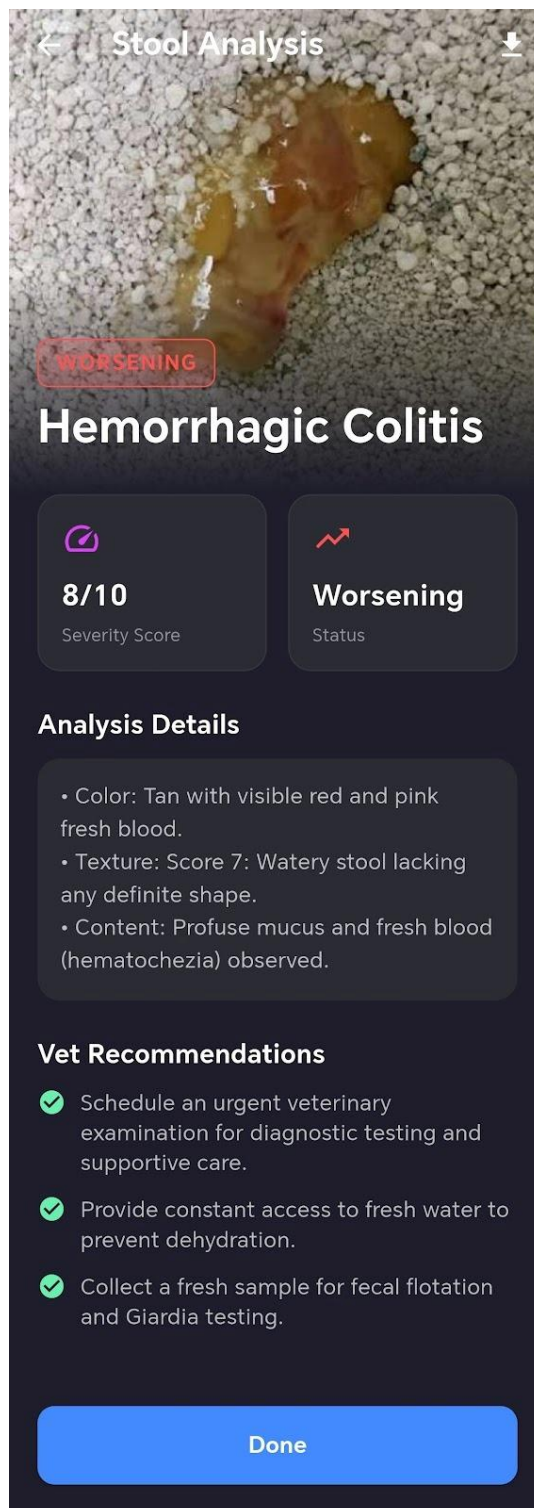


Figure 5.2.29 Stool Analysis Results

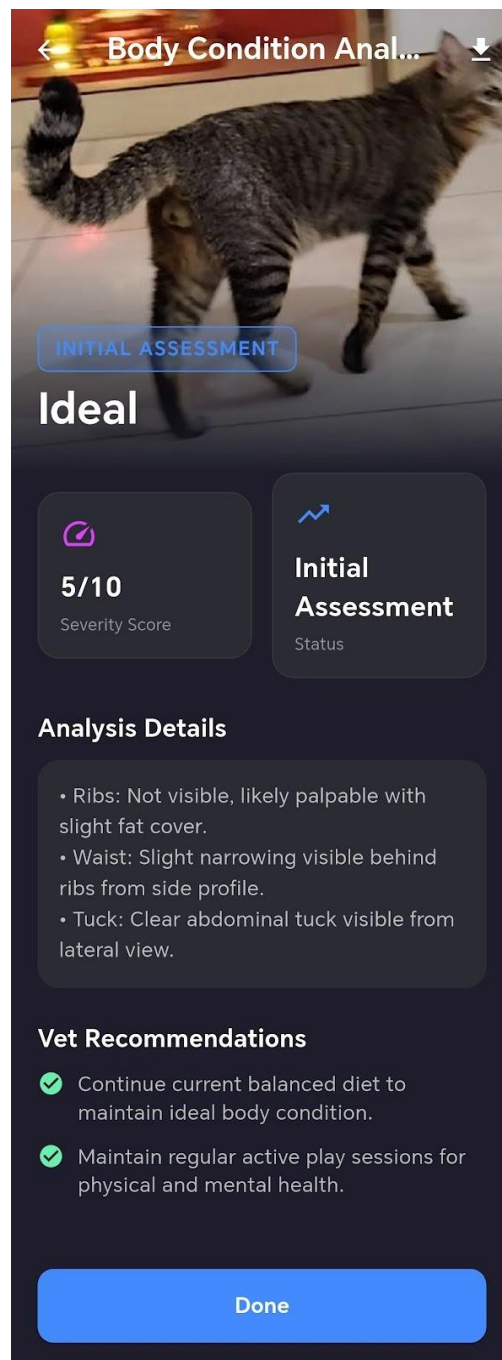


Figure 5.2.30 Body Size Analysis Result

To download the pet health report, there will be two alternative methods. First, users can select the Medical History selection that developed at the bottom of the selection screen. Alternatively, users can also click on the Download icon at the top right corner of each analysis result screen. Once clicking, a window will pop out to ask the user to select a suitable time interval for the report. Users can click on it and an email will be sent to users.

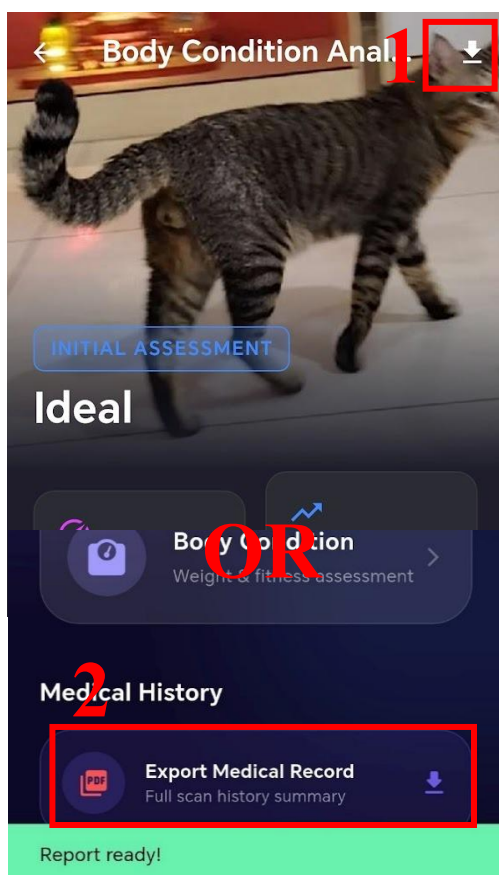


Figure 5.2.31 Download Report

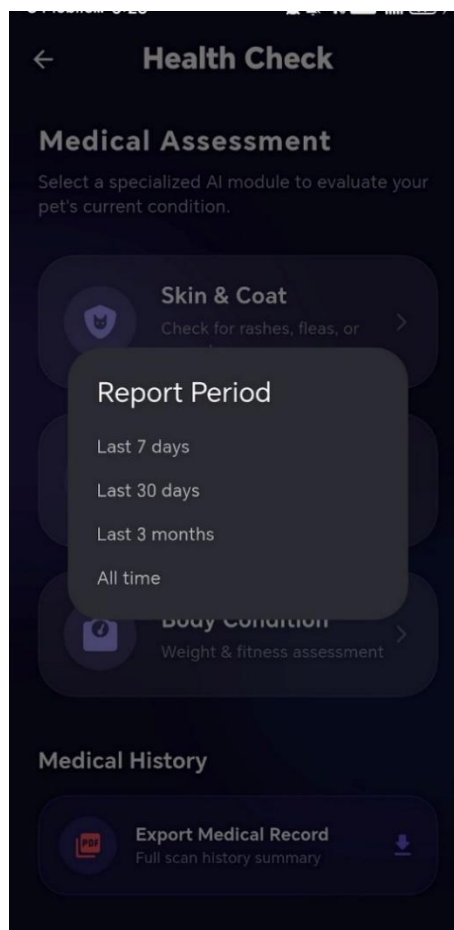


Figure 5.2.32 Report Period Selection

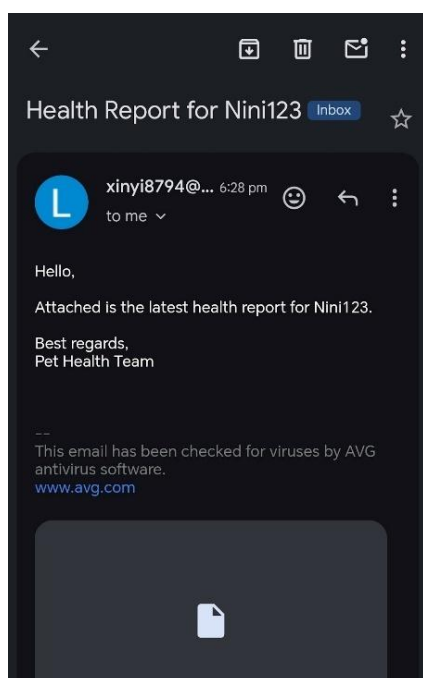


Figure 5.2.33
Health Report Send via Email

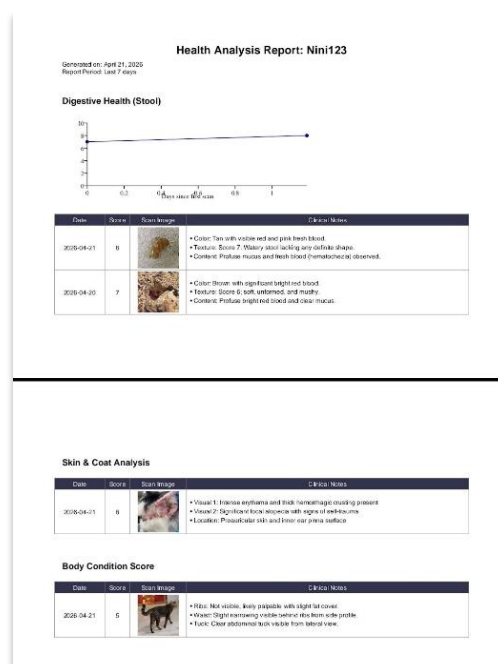


Figure 5.2.34 Health Report Sample

```

@router.get("/report/{pet_id}")
async def generate_report(
    pet_id: str,
    background_tasks: BackgroundTasks,
    # New optional params - Flutter passes these as query strings
    days: Optional[int] = None, # e.g. ?days=30 for last 30 days
    start_date: Optional[str] = None, # e.g. ?start_date=2025-01-01
    end_date: Optional[str] = None, # e.g. ?end_date=2025-03-31
):
    print(f"\n [START] Generating PDF Report for Pet: {pet_id}")
    try:
        # 1. Fetch data
        pet_name = get_pet_name(pet_id)
        user_email = get_user_email_by_pet_id(pet_id)

        query = supabase.table('health_logs').select('*').eq('pet_id', pet_id)

        # Apply date filter
        now = datetime.now(timezone.utc)
        if days:
            cutoff = (now - timedelta(days=days)).isoformat()
            query = query.gte('created_at', cutoff)
            report_period = f"Last {days} days"
        elif start_date and end_date:
            query = query.gte('created_at', start_date).lte('created_at', end_date)
            report_period = f"{start_date} to {end_date}"
        else:
            report_period = "All time"

        response = query.order('created_at', desc=False).execute()
        logs = response.data

        if not logs:
            raise HTTPException(status_code=404, detail="No health logs found for this pet.")

        # 2. Setup memory buffer and pdf styles
        buffer = io.BytesIO()
        doc = SimpleDocTemplate(buffer, pagesize = landscape(letter))
        styles = getSampleStyleSheet()
        body_style = ParagraphStyle('Body', parent=styles['BodyText'], fontSize=10, leading=14)
        elements = []

        elements.append(Paragraph(f"Health Analysis Report: {pet_name}", styles['Title']))
        elements.append(Paragraph(f"Generated on: {datetime.now().strftime('%B %d, %Y')}", styles['Normal']))
        elements.append(Paragraph(f"Report Period: {report_period}", styles['Normal']))
        elements.append(Spacer(1, 20))

        # Group logs by type
        logs_by_type = defaultdict(list)
        for log in logs:
            dt=datetime.fromisoformat(log['created_at'].replace('Z', '+00:00'))
            log['dt_obj']=dt
            log['date_str']=dt.strftime('%Y-%m-%d')
            logs_by_type[log['scan_type']].append(log)

```

Figure 5.2.35 Generate Report Based on Time Interval

5.2.6 Pet Sound Interpretation Feature Development

From the Home Screen, users can tap on AI Translator to enter the Emotion Interpretation Screen. Unlike a simple audio classifier, this feature employs a dual-mode multimodal analysis pipeline which combines an audio classification model with a visual body language model. Before starting analysis, users can choose to take or upload their pet video. Once successfully uploaded, the video will be passed to the FastAPI backend. On the backend side, the app simultaneously processes two streams of information: the audio track is extracted and passed through the custom CNN-BiLSTM sound classification model, while three randomly selected frames from the video are passed through a MobileNetV2-based visual model. The two confidence-weight predictions are then fused to produce a final combined translation result. There will have 4 different situation examples:

1. If Audio and Visual result is same:

- Audio: happy, Visual: friendly_active → “Your cat is feeling very happy and sociable!”
- Audio: angry, Visual: aggressive_defensive → “Your cat is very upset. Please give them space.”

2. If Audio and Visual result is conflict:

- Visual’s Confidence > Audio’s Confidence → Take the visual prediction result as final prediction
- Visual’s Confidence < Audio’s Confidence → Take the audio prediction result as final prediction

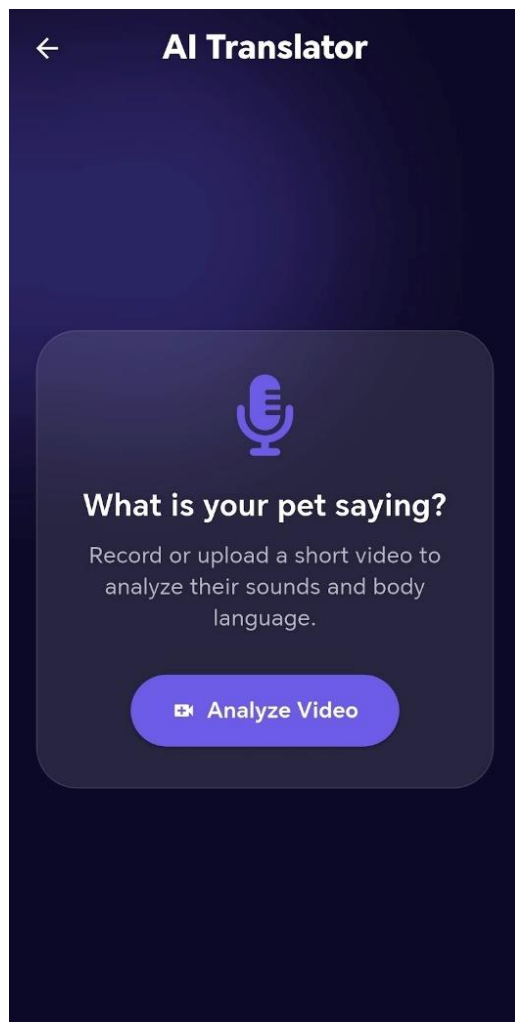


Figure 5.2.36

Pet Emotion Interpretation Screen

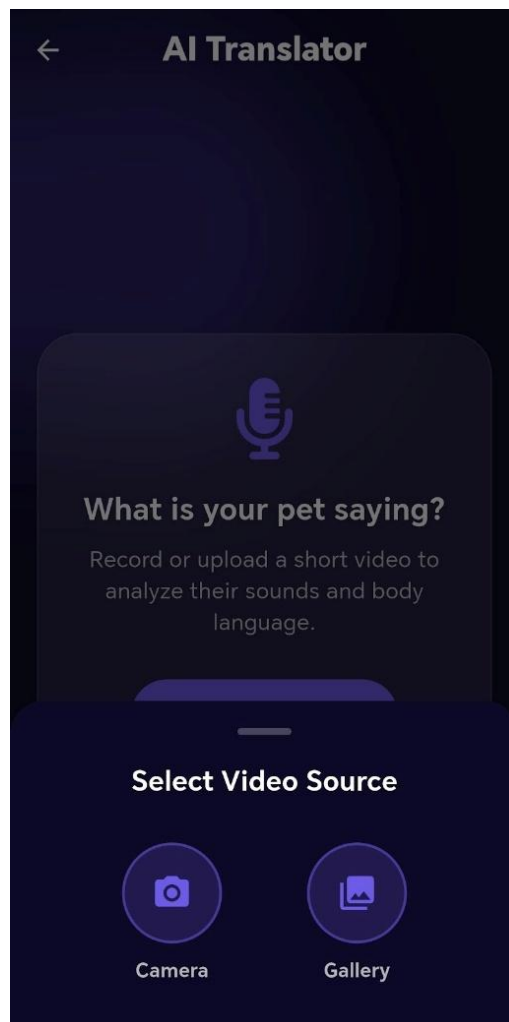


Figure 5.2.37

Video Source Upload Method

```
# =====
# 🎵 1. AUDIO PREDICTION (Smart Routing)
# =====
audio_preds = None
audio_label = "Audio AI Offline"
pet_audio_path = os.path.join(MODEL_DIR, "pets", f"{pet_id}_audio.h5")

if AUDIO_LOADED or os.path.exists(pet_audio_path):
    if os.path.exists(pet_audio_path):
        print(f" 🌟 [SMART ROUTER] Loading CUSTOM Audio Model for {pet_id}")
        custom_audio_model = build_audio_model(len(AUDIO_LABELS))
        custom_audio_model.load_weights(pet_audio_path)
        audio_features = extract_audio_features(audio_path)
        audio_preds = custom_audio_model.predict(audio_features, verbose=0)
    else:
        print(f" 🌐 [SMART ROUTER] Using GLOBAL Audio Model")
        audio_features = extract_audio_features(audio_path)
        audio_preds = audio_model.predict(audio_features, verbose=0)

audio_idx = int(np.argmax(audio_preds))
audio_label = AUDIO_LABELS.get(audio_idx, "Unknown Sound")
```

Figure 5.2.38 Audio Model Processing

```
# =====
# 📺 2. VISUAL PREDICTION (Smart Routing)
# =====
avg_visual_preds = None
visual_label = "Unknown Body Language"
pet_visual_path = os.path.join(MODEL_DIR, "pets", f"{pet_id}_visual.h5")

if VISUAL_LOADED or os.path.exists(pet_visual_path):
    visual_features = extract_visual_features(video_path)

    if visual_features is not None:
        if os.path.exists(pet_visual_path):
            print(f" 🌟 [SMART ROUTER] Loading CUSTOM Visual Model for {pet_id}")
            custom_visual_model = keras.models.load_model(pet_visual_path)
            visual_preds = custom_visual_model.predict(visual_features, verbose=0)
        else:
            print(f" 🌐 [SMART ROUTER] Using GLOBAL Visual Model")
            visual_preds = visual_model.predict(visual_features, verbose=0)

        avg_visual_preds = np.mean(visual_preds, axis=0)
        visual_idx = int(np.argmax(avg_visual_preds))
        visual_label = VISUAL_LABELS.get(visual_idx, "Unknown Body Language")
    else:
        visual_label = "No Video Extracted"
else:
    import random
    visual_label = random.choice(list(VISUAL_LABELS.values()))
```

Figure 5.2.39 Visual Model Processing

```

# =====
# 3. HYBRID FUSION LOGIC
# =====
audio_conf = float(np.max(audio_preds)) if audio_preds is not None else 0.0
visual_conf = float(np.max(avg_visual_preds)) if avg_visual_preds is not None else 0.0

print(f" 🎧 [CONFIDENCE] Audio: {audio_conf:.2f} | Visual: {visual_conf:.2f}")

RULEBOOK = {
    ("Happy", "friendly_active"): "Your cat is feeling very happy and sociable!",
    ("Angry", "aggressive_defensive"): "Your cat is very upset. Please give them space.",
}

pair = (audio_label, visual_label)

if pair in RULEBOOK:
    print(f" 📖 [RULEBOOK] Match found for {pair}")
    final_text = RULEBOOK[pair]
else:
    if visual_conf >= audio_conf:
        display_label = visual_label.replace('_', ' ').title()
        final_text = f"Based on body language, your cat seems {display_label}."
    else:
        final_text = f"That sound indicates your cat is feeling {audio_label}."

print(f" 🗨️ [FINAL RESULT] {final_text}")

```

Figure 5.2.40 Rulebook for Final Prediction

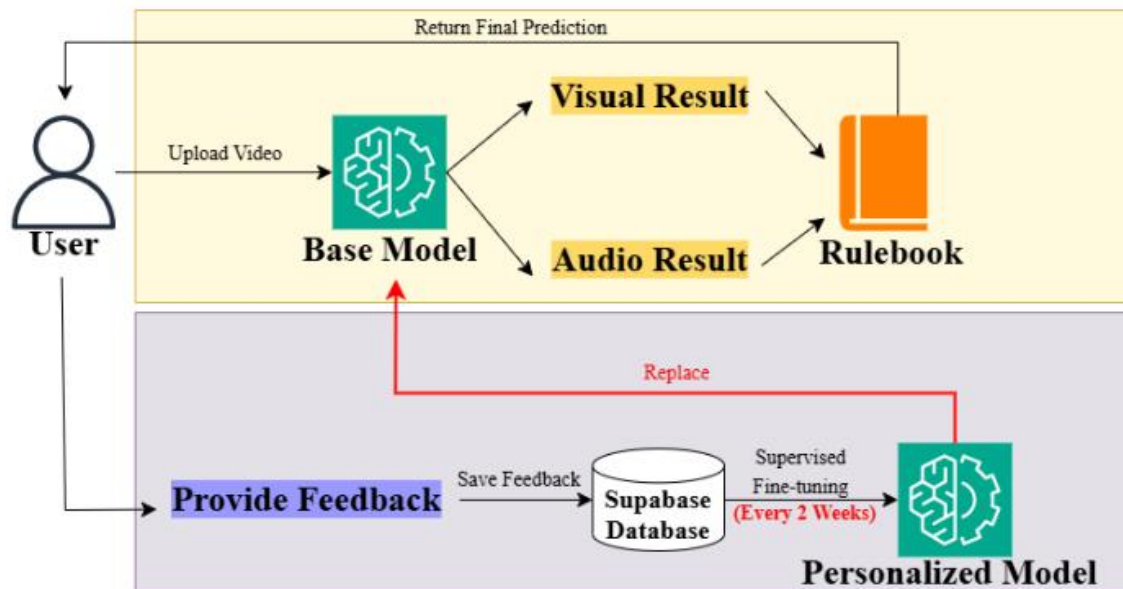


Figure 5.2.41 Pet Emotion Interpretation Flow

Once the result is displayed, users can submit feedback by clicking on the feedback section. If the prediction is correct, users can click on the Correct Button. Otherwise, if the prediction is incorrect, users can click on the Wrong Button and select the answer that they believe is correct. All of the feedback will be saved into the Supabase Database. On a bi-weekly schedule, the saved feedback, where `is_correct` is not null, will be retrieved from the database. A supervised fine-tuning session is triggered by using the collected feedback that is provided by users. A selective layer-freezing strategy is applied before training: for the audio model, only the CNN

front-end layers are frozen while the BiLSTM and dense layers remain trainable; for the visual model, the early convolution blocks are frozen while the higher-level feature blocks and custom classification head are kept trainable. The feedback corrections are weighted at 1.5 and processed with more gradient steps than confirmations to emphasise learning from mistakes. Once training is complete, all layers are unfrozen and the updated model are saved as {pet_id} audio.h5 and {pet_id}_visual.h5 in the models/pets/ directory, effectively replacing the base model for that specific pet on all future predictions. As a result, the sound interpretation feature will become more personalised and accurate over the time.



Figure 5.2.42

Emotion Interpretation Result

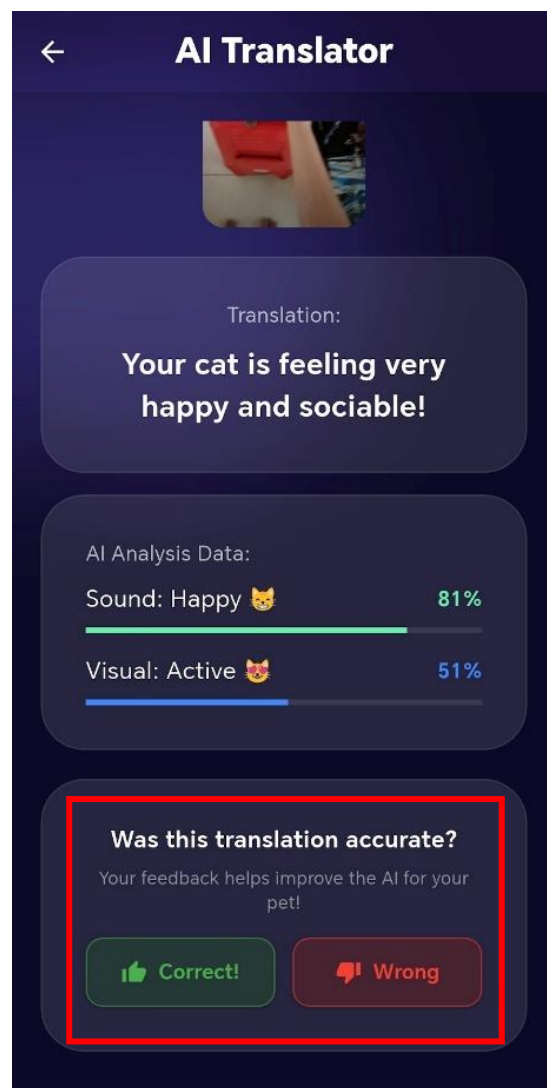


Figure 5.2.43

Feedback Section



Figure 5.2.44 Correct Feedback

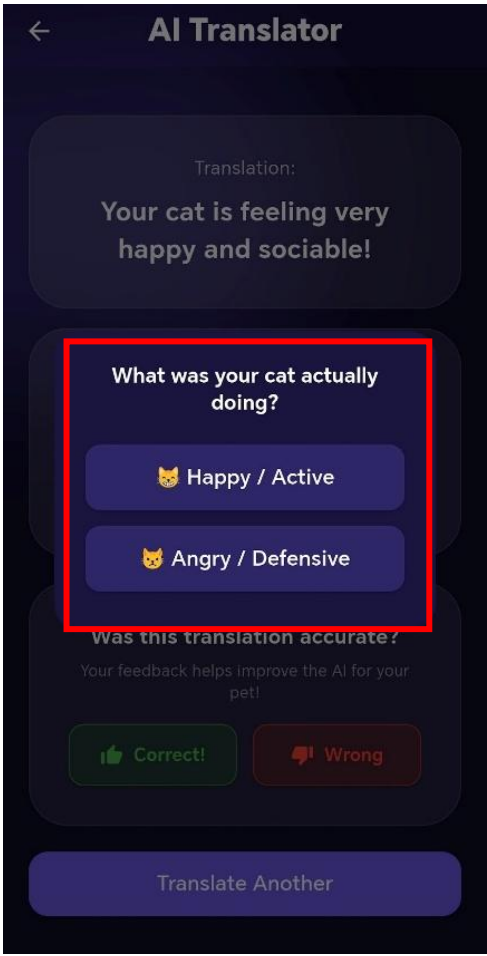


Figure 5.2.45 Correction Label
When Predict Wrongly

A screenshot of the Supabase web interface showing a table named 'public.multimodal_logs'. The table has columns: id, video_url, audio_label, visual_label, final_prediction, user_correction, and is_correct. The 'is_correct' column is highlighted with a red box. The table contains six rows of data, with the last five rows having 'is_correct' values of TRUE. The first row has 'is_correct' as FALSE.

id	video_url	audio_label	visual_label	final_prediction	user_correction	is_correct
76ab32ee-609d-4461-bd8b-d57df8df125e	https://pqhmobospoktvzhnlfoj.supabase.	Angry	aggressive_def	Your cat is very upset. Please give them s	NULL	FALSE
3239f249-de7c-465e-a9ec-e33da8aaf415	https://pqhmobospoktvzhnlfoj.supabase.	Happy	friendly_active	Your cat is feeling very happy and sociabl	NULL	TRUE
7a9c0b49-2bef-4f30-8b59-8c57261da665	https://pqhmobospoktvzhnlfoj.supabase.	Happy	friendly_active	Your cat is feeling very happy and sociabl	NULL	TRUE
9d071bda-b416-4253-9179-2aa0b7af7dcb	https://pqhmobospoktvzhnlfoj.supabase.	Happy	friendly_active	Your cat is feeling very happy and sociabl	NULL	TRUE
c99c88f0-736f-4522-afcb-55b35c508148	https://pqhmobospoktvzhnlfoj.supabase.	Happy	friendly_active	Your cat is feeling very happy and sociabl	NULL	TRUE
35c94471-27d1-4dba-9f64-b1eb9227a5c5	https://pqhmobospoktvzhnlfoj.supabase.	Happy	friendly_active	Your cat is feeling very happy and sociabl	NULL	TRUE

Figure 5.2.46 Screenshot of Supabase Multimodal Table

```
# *****
# 5. USER FEEDBACK ENDPOINT (FIXED)
# *****
@router.post("/feedback")
async def submit_feedback(data: FeedbackData):
    print(f"📄 [FEEDBACK] Log ID {data.log_id}: {'✅ Correct' if data.is_correct else '❌ Incorrect'}")

    try:
        if data.is_correct:
            update_payload = {
                "is_correct": True
            }
            print(f"🔥 Confirmed correct. Original labels kept for reinforcement training.")
        else:
            if data.correct_audio_label and data.correct_visual_label:
                corrected_audio = data.correct_audio_label
                corrected_visual = data.correct_visual_label
            else:
                original = supabase.table("multimodal_logs").select(
                    "audio_label, visual_label"
                ).eq("id", data.log_id).execute()

                if not original.data:
                    return {"error": "Log not found"}

                orig_audio = original.data[0].get("audio_label", "")
                if "Happy" in orig_audio:
                    corrected_audio = "Angry"
                    corrected_visual = "aggressive_defensive"
                else:
                    corrected_audio = "Happy"
                    corrected_visual = "friendly_active"

            update_payload = {
                "is_correct": False,
                "user_correct": corrected_audio, # The TRUE audio label
                "user_correct_visual": corrected_visual # The TRUE visual label
            }
            print(f"💡 Saved correction + Audio: {corrected_audio}, Visual: {corrected_visual}")

            supabase.table("multimodal_logs").update(update_payload).eq("id", data.log_id).execute()

        return {
            "status": "Success",
            "message": "Feedback saved. Model will learn from this in the next training cycle."
        }
    }
```

Figure 5.2.47 Feedback Logic

```
# --- 5. Training loop
# Process corrections first (highest learning value), then confirmations
sorted_logs = corrections + confirmations

for log in sorted_logs:
    temp_video = "temp_r1_video.mp4"
    temp_audio = "temp_r1_audio.wav"

    try:
        download_video(log['video_url'], temp_video)

        from moviepy.video.io.VideoFileClip import VideoFileClip
        clip = VideoFileClip(temp_video)
        clip.audio.write_audiofile(temp_audio, logger=None)
        clip.close()

        audio_features = extract_audio_features(temp_audio)
        visual_features = extract_visual_features(temp_video, num_frames=3)

        if log['is_correct']:
            train_audio_label = log['audio_label']
            train_visual_label = log['visual_label']
            audio_steps = 1
            visual_steps = 1
            sample_weight = 1.0
            signal_type = "REINFORCE"
        else:
            train_audio_label = log.get('user_correct') or _flip_audio(log['audio_label'])
            train_visual_label = log.get('user_correct_visual') or _flip_visual(log['visual_label'])
            audio_steps = 3 # audio: 3 steps fine
            visual_steps = 1 # visual: 1 step only - avoids oscillation with large model
            sample_weight = 1.5
            signal_type = "CORRECT"

        target_audio_idx = AUDIO_LABEL_MAP.get(train_audio_label, 0)
        for _ in range(audio_steps):
            with tf.GradientTape() as tape:
                a_preds = audio_model(audio_features, training=False)
                a_loss = loss_fn([target_audio_idx], a_preds) * sample_weight
                a_grads = tape.gradient(a_loss, audio_model.trainable_variables)
                audio_optimizer.apply_gradients(zip(a_grads, audio_model.trainable_variables))

        if visual_features is not None:
            target_visual_idx = VISUAL_LABEL_MAP.get(train_visual_label, 0)
            v_targets = [target_visual_idx] * len(visual_features)
            for _ in range(visual_steps):
                with tf.GradientTape() as tape:
                    v_preds = visual_model(visual_features, training=False)
                    v_loss = loss_fn(v_targets, v_preds) * sample_weight
                    v_grads = tape.gradient(v_loss, visual_model.trainable_variables)
                    visual_optimizer.apply_gradients(zip(v_grads, visual_model.trainable_variables))

        print(f"📡 [{signal_type}] audio_steps={audio_steps} visual_steps={visual_steps} "
```

Figure 5.2.48 Supervised Fine-tuning Training

```

  backend
  > _pycache_
  cat_translator
  models
  > pets
    f95249e2-785f-42fe-a5cd-77e601f49505_audio.h5
    f95249e2-785f-42fe-a5cd-77e601f49505_visual.h5
    cat_sound_classifier.h5
    cat_visual_model.h5
    mfcc_mean.npy
    mfcc_std.npy

```

Figure 5.2.49 Personalised Model Created

5.2.7 Pet Calendar Reminder Feature Development

Users can access the Pet Calendar Reminder from the Home Screen. Upon entering, the screen displays a monthly calendar view where existing pet-related events are visually marked on their respective dates. If users click on a date, they are able to see the scheduled pet-related event for that date which includes title, category, clinic name and description if they exist.

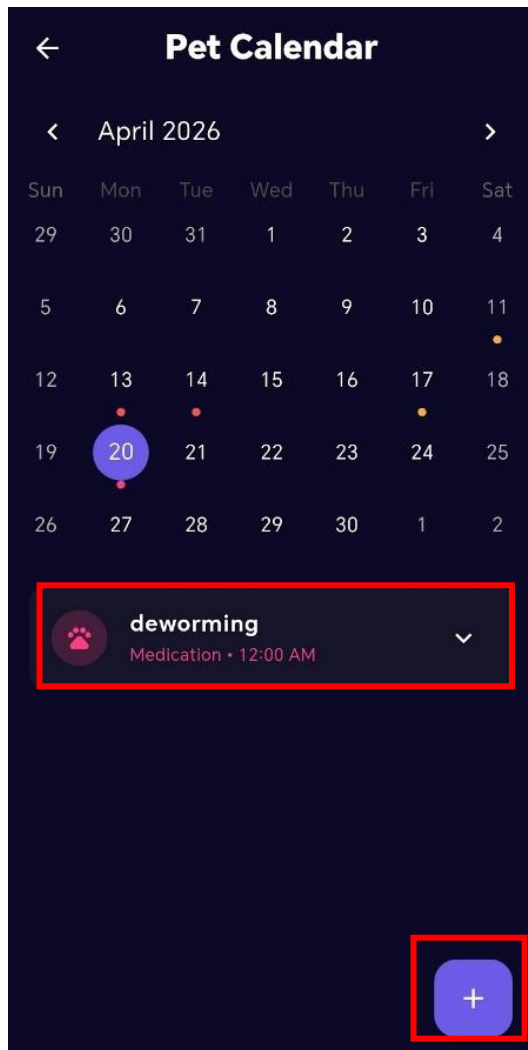


Figure 5.2.50

Pet Calendar Reminder Screen

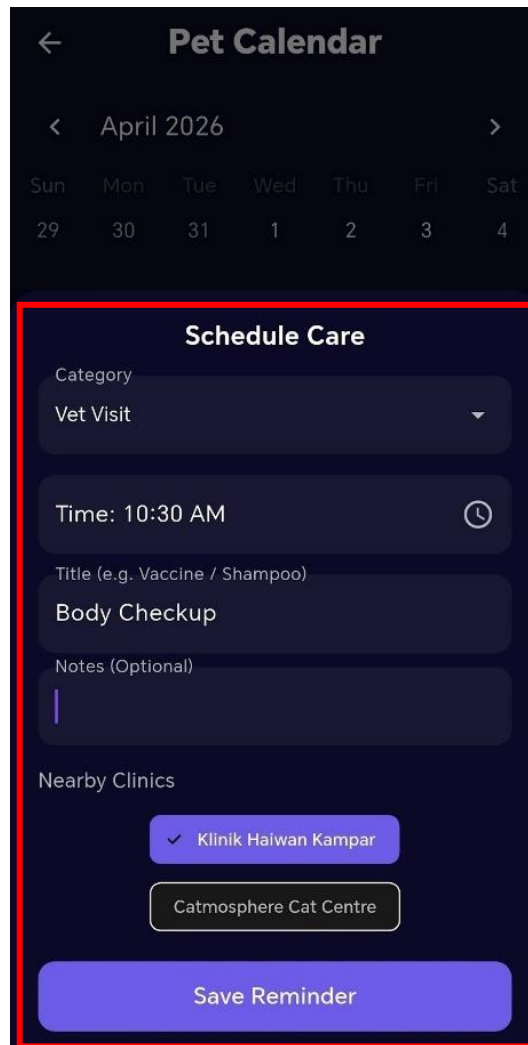


Figure 5.2.51

Activity Addition Form

To add a new event, users are able to tap the Add Event Button which opens a simple form to enter event title, date, description and category (General, Bathing, Vet Visit, Vaccination, Medication). They can also optionally select an automatically displayed clinic selection that is within a 15km range. When submitted, the app sends the data to the backend to be stored in the Supabase Database. Meanwhile, a confirmation message is shown upon success.

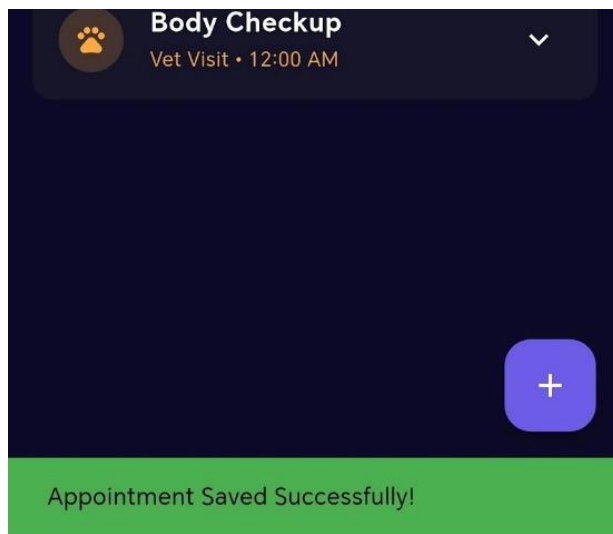


Figure 5.2.52
Activity Successfully Saved Message

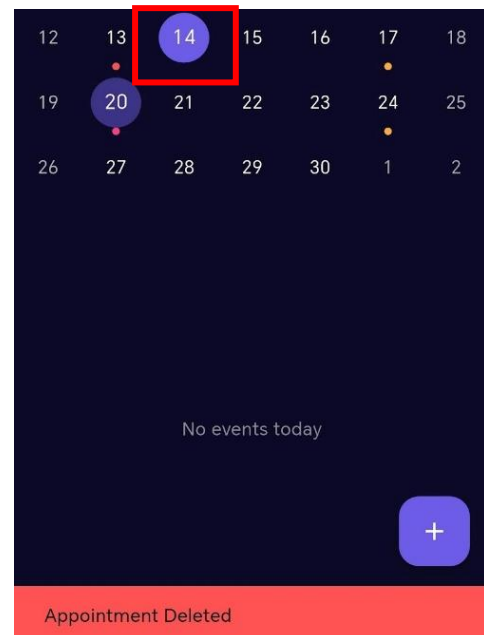


Figure 5.2.53
Activity Successfully Deleted Message

To help users always keep track of some important pet-care event, the app will automatically generate the next scheduled date under events such as vaccinations and baths. Users can also update or delete existing events directly from the event card. To remind users of the upcoming pet-related events, a notification will be delivered to users a day before the scheduled event.

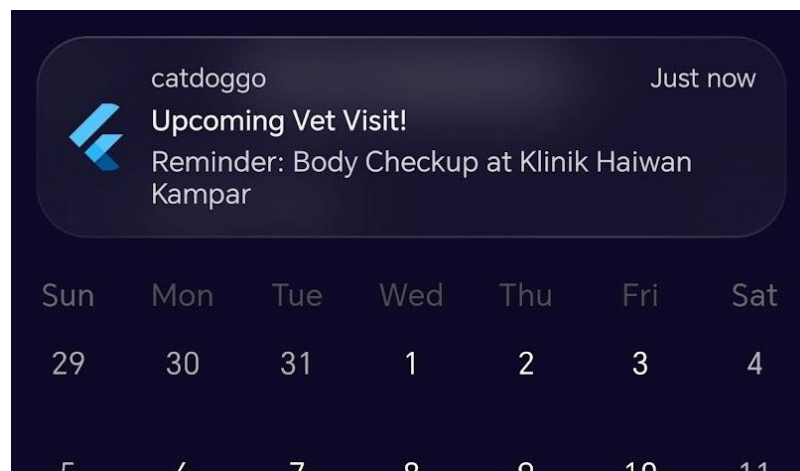


Figure 5.2.54 Reminder Notification

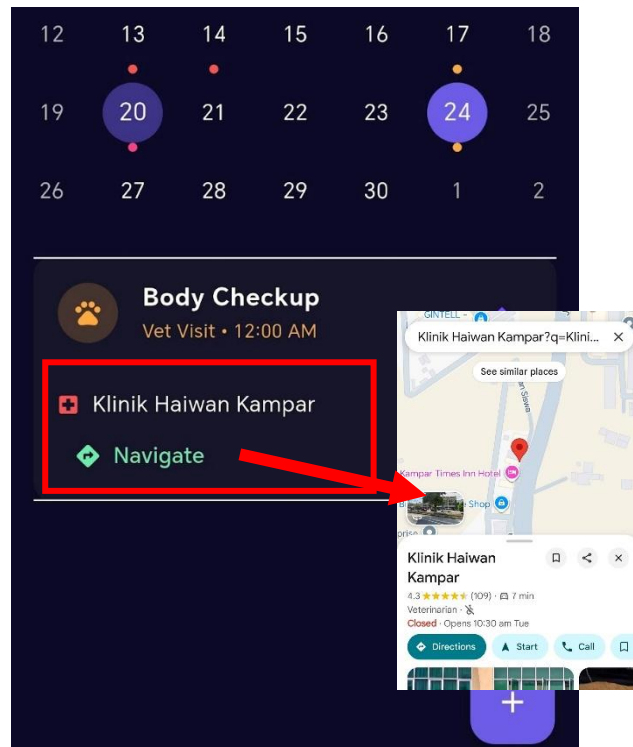


Figure 5.2.55 Navigation to Vet Clinic

```
# --- 1. GET EVENTS ---
@router.get("/{pet_id}")
async def get_calendar_events(pet_id: str):
    try:
        response = supabase.table("calendar_events").select("*").eq("pet_id", pet_id).execute()
        return {"events": response.data}
    except Exception as e:
        print(f"Error fetching calendar events: {e}")
        return {"error": str(e)}

# --- 2. ADD EVENT ---
@router.post("/add")
async def add_event(event: EventData):
    try:
        # Convert pydantic model to dictionary and insert
        response = supabase.table("calendar_events").insert(event.model_dump()).execute()
        return {"status": "success", "data": response.data}
    except Exception as e:
        print(f"Error adding event: {e}")
        return {"error": str(e)}

# --- 3. UPDATE EVENT ---
@router.put("/update/{event_id}")
async def update_event(event_id: str, event: EventData):
    try:
        response = supabase.table("calendar_events").update(event.model_dump()).eq("id", event_id).execute()
        return {"status": "success", "data": response.data}
    except Exception as e:
        print(f"Error updating event: {e}")
        return {"error": str(e)}

# --- 4. DELETE EVENT ---
@router.delete("/delete/{event_id}")
async def delete_event(event_id: str):
    try:
        supabase.table("calendar_events").delete().eq("id", event_id).execute()
        return {"status": "success", "message": "Event deleted successfully"}
    except Exception as e:
        print(f"Error deleting event: {e}")
        return {"error": str(e)}
```

Figure 5.2.56 Calendar Event Backend Logic

```

try {
  await NotificationService.scheduleAppointmentReminder(
    id: Random().nextInt(100000),
    title: "Upcoming $_selectedCategory!",
    body:
      "Reminder: $title${_selectedClinic?['name'] != null ? ' at ${_selectedClinic!['name']}' : ''}",
    appointmentTime: fullTime,
  );
} catch (notifError) {
  debugPrint("Failed to schedule notification: $notifError");
}

if (_selectedCategory == 'Vaccination') {
  await _createFollowUp(
    "Vaccination Booster",
    "Follow-up for: $title",
    fullTime.add(const Duration(days: 21)),
    'Vaccination',
  );
} else if (_selectedCategory == 'Bathing') {
  await _createFollowUp(
    "Monthly Bath",
    "Time for the next scrub!",
    fullTime.add(const Duration(days: 30)),
    'Bathing',
  );
}
}

```

Figure 5.2.57 Notification Setup and Autogenerated Next Scheduled Event

5.2.8 Pet Expense Tracker Development

The Pet Expense Tracking application is an easy way for pet owners to track all of their expenses such as food purchases, vet consultations, grooming sessions and medication costs which are related to their pets. Users are able to access this feature by clicking the Expense Tracker on Home Screen. Once entering the screen, users will be directed to a financial summary at the top of the screen followed by a series of individual expenses below in chronological order. The screen also supports filtering expenses by category or date range, allowing users to review spending patterns over time.

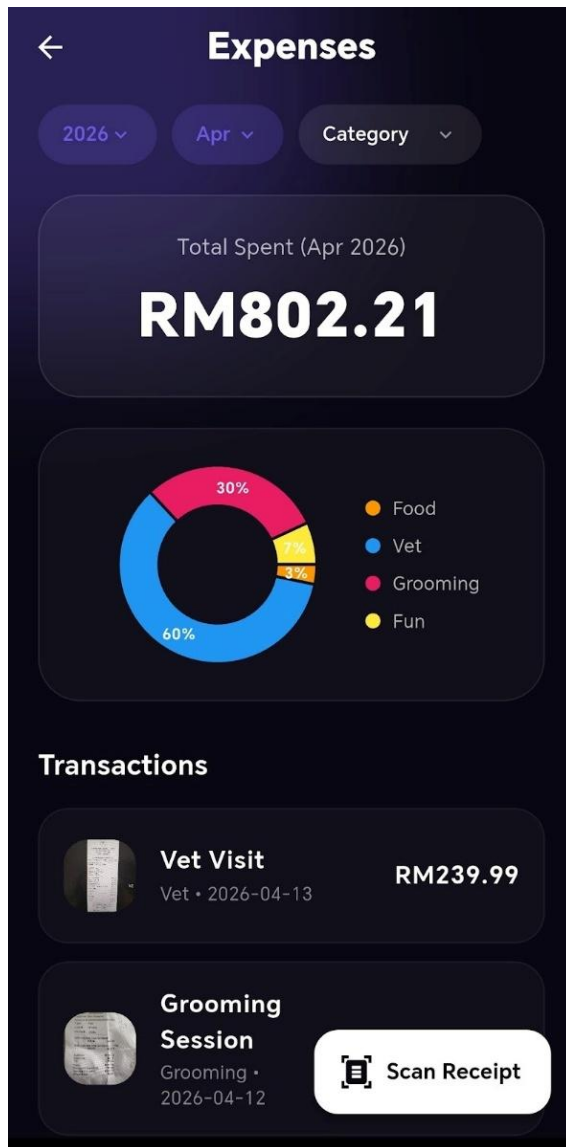


Figure 5.2.58 Expense Tracker Screen

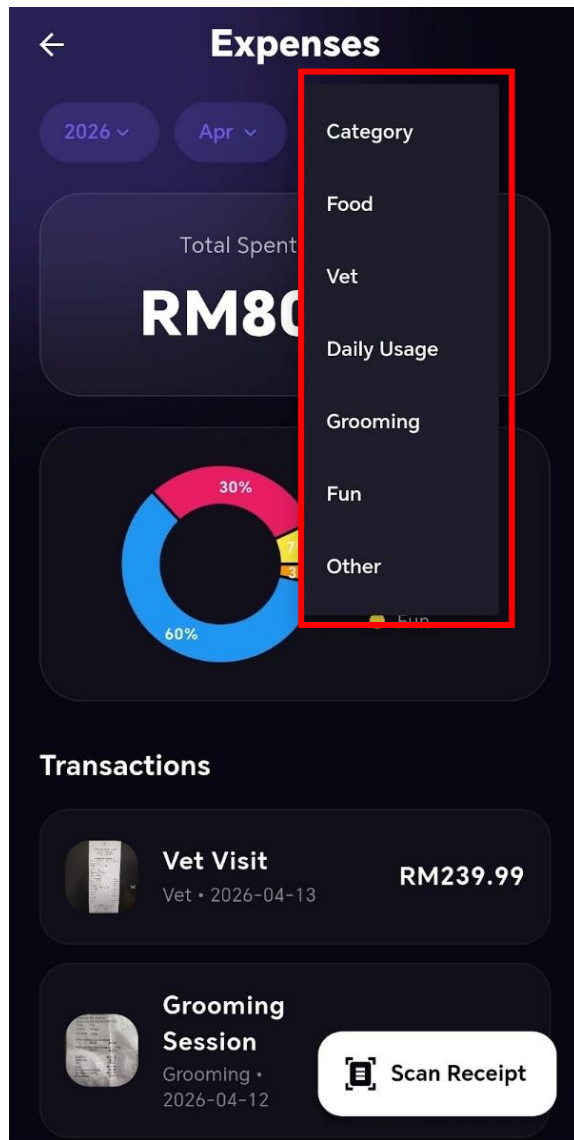


Figure 5.2.59 Category Filtering

To log a new expense, users can simply tap the Add Expense Button and upload the purchased receipt. After uploading, the application will extract the category and total amount automatically. If the expense involves multiple categories, users can use the Split feature to allocate it among each of the categories. Any incorrect auto-generated information can also be adjusted manually before saving. Once confirmed, all of the information will be stored in the Supabase database. By clicking on any individual expense, users can edit or delete any entry immediately and the changes will reflect in the database instantly via the backend API.

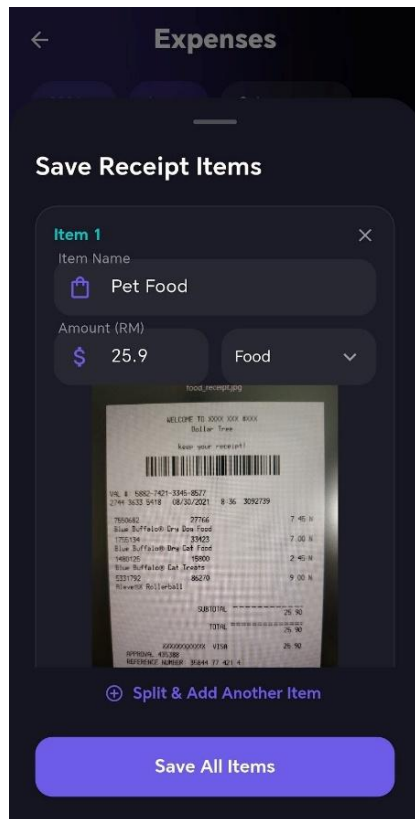


Figure 5.2.60 New Expense Creation

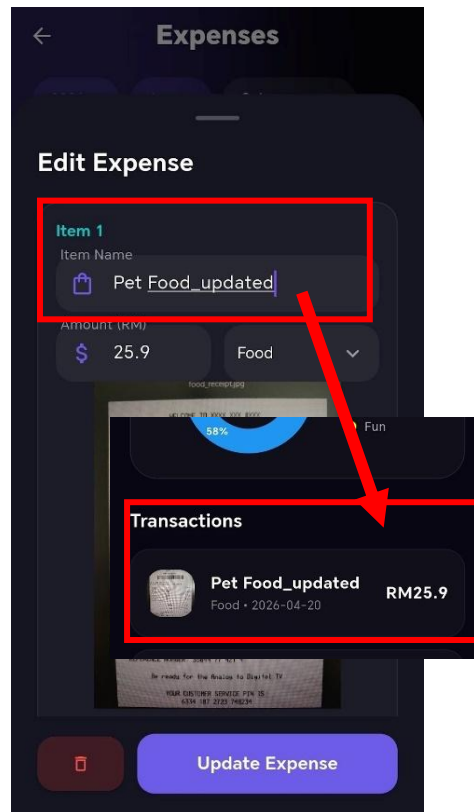


Figure 5.2.61 Expense Updation

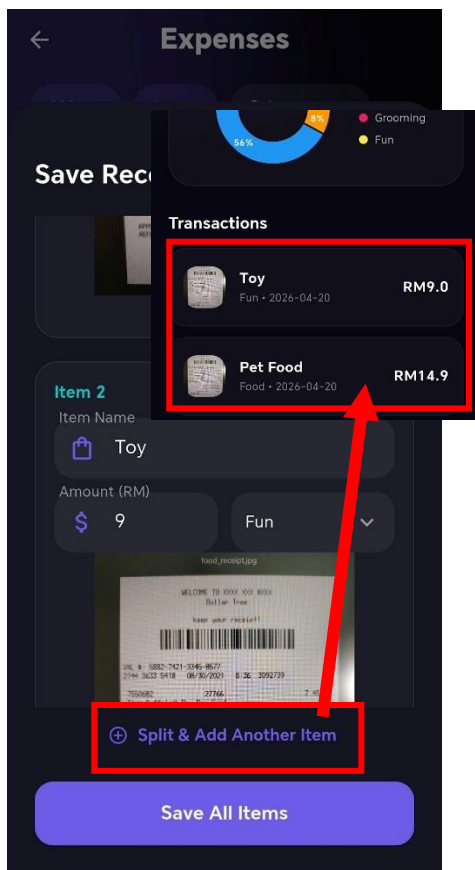


Figure 5.2.62 Expense Splitting

```
Future<void> _processReceipt() async {
  final picker = ImagePicker();
  final pickedFile = await picker.pickImage(source: ImageSource.camera);
  if (pickedFile == null) return;
  setState(() => _isLoading = true);
  try {
    final inputImage = InputImage.fromFilePath(pickedFile.path);
    final textRecognizer =
      TextRecognizer(script: TextRecognitionScript.latin);
    final RecognizedText recognizedText =
      await textRecognizer.processImage(inputImage);
    String rawText = recognizedText.text.toLowerCase();
    textRecognizer.close();

    String guessedItemName = "Scanned Item";
    double guessedPrice = 0.0;
    String guessedCategory = "Other";

    RegExp priceRegex = RegExp(r'\d+\.\d{2}');
    Iterable<RegExpMatch> matches = priceRegex.allMatches(rawText);
    for (final match in matches) {
      double? price = double.tryParse(match.group(0)!);
      if (price != null && price > guessedPrice) guessedPrice = price;
    }

    if (rawText.contains('royal canin') ||
        rawText.contains('food') ||
        rawText.contains('treat')) {
      guessedCategory = "Food";
      guessedItemName = "Pet Food";
    } else if (rawText.contains('clinic') ||
        rawText.contains('vet') ||
        rawText.contains('vaccine') ||
        rawText.contains('checkup')) {
      guessedCategory = "Vet";
      guessedItemName = "Vet Visit";
    } else if (rawText.contains('litter') ||
        rawText.contains('tofu') ||
        rawText.contains('shampoo') ||
        rawText.contains('cloth')) {
      guessedCategory = "Daily Usage";
      guessedItemName = "Daily Supplies";
    } else if (rawText.contains('toy') ||
        rawText.contains('wand') ||
        rawText.contains('catnip')) {
      guessedCategory = "Fun";
      guessedItemName = "Pet Toys";
    } else if (rawText.contains('groom') ||
        rawText.contains('cut') ||
        rawText.contains('bath')) {
      guessedCategory = "Grooming";
      guessedItemName = "Grooming Session";
    }

    setState(() => _isLoading = false);
    showAddEditDialog(
```

Figure 5.2.63

Detail Autogenerated Logic

5.2.9 Pet Community Interaction Development

The Community Screen is accessible from the bottom navigation bar in the Home Screen. It serves as a social hub where users can browse breed-specific health tips, interact with other pet owners and discover useful pet care information. When users open the Community Screen, the backend will pass the user's pet breed to search for the breed-specific health tips in the database. The personalised tips which cover genetic risks, nutrition needs and grooming tips will be displayed on the top of the screen. If a matching record exists, those tailored tips are returned directly. If the breed is not in the database, a set of generic care reminders is returned as a fallback.

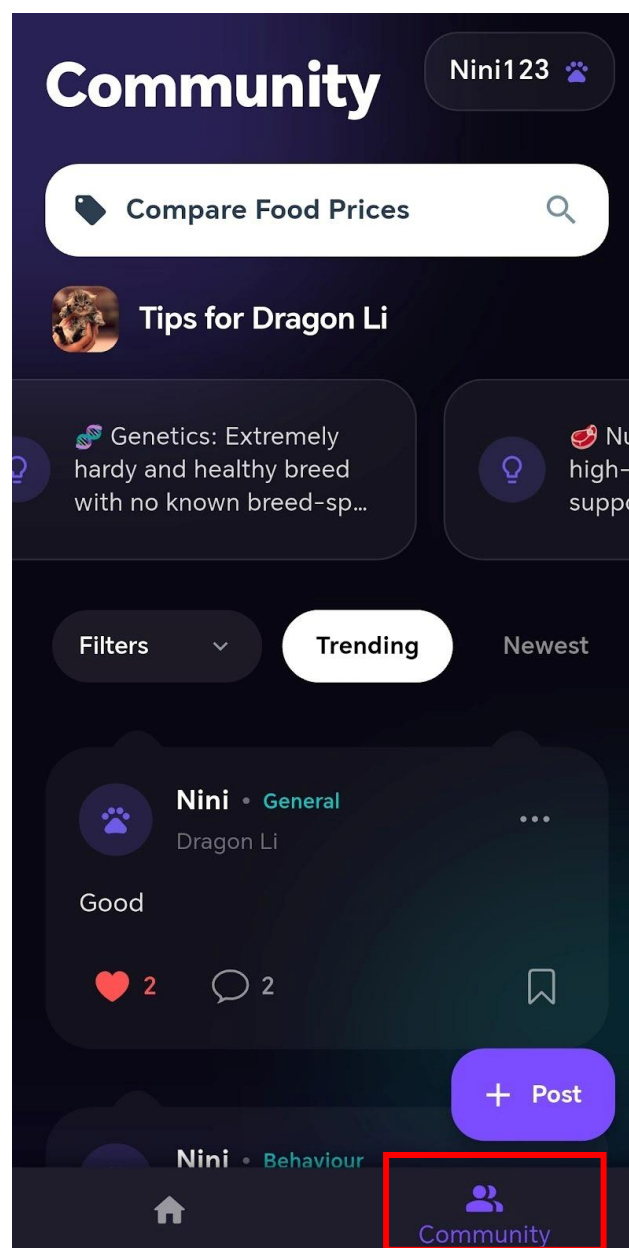
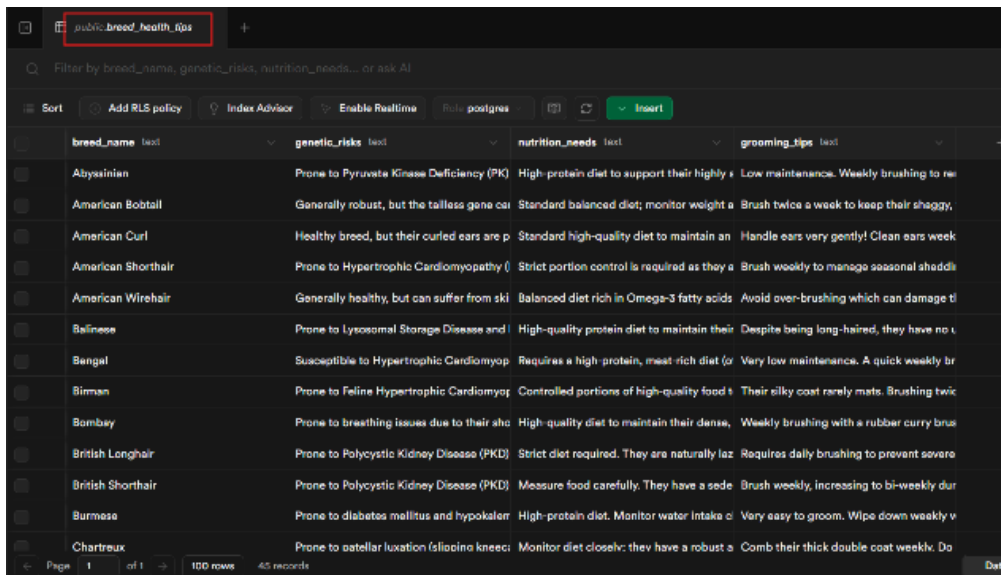


Figure 5.2.64 Community Interaction Screen



breed_name	genetic_risks	nutrition_needs	grooming_tips
Abyssinian	Prone to Pyruvate Kinase Deficiency (PK)	High-protein diet to support their highly a	Low maintenance. Weekly brushing to re
American Bobtail	Generally robust, but the tailless gene ca	Standard balanced diet; monitor weight a	Brush twice a week to keep their shaggy,
American Curl	Healthy breed, but their curled ears are p	Standard high-quality diet to maintain an	Handle ears very gently! Clean ears week
American Shorthair	Prone to Hypertrophic Cardiomyopathy (	Strict portion control is required as they a	Brush weekly to manage seasonal shedd
American Wirehair	Generally healthy, but can suffer from ski	Balanced diet rich in Omega-3 fatty acids	Avoid over-brushing which can damage ti
Balinese	Prone to Lysosomal Storage Disease and	High-quality protein diet to maintain their	Despite being long-haired, they have no c
Bengal	Susceptible to Hypertrophic Cardiomyop	Requires a high-protein, meat-rich diet (o	Very low maintenance. A quick weekly br
Birman	Prone to Feline Hypertrophic Cardiomyop	Controlled portions of high-quality food	Their silky coat rarely mats. Brushing twic
Bombay	Prone to breathing issues due to their sho	High-quality diet to maintain their dense,	Weekly brushing with a rubber curry brus
British Longhair	Prone to Polycystic Kidney Disease (PKD)	Strict diet required. They are naturally laz	Requires daily brushing to prevent severe
British Shorthair	Prone to Polycystic Kidney Disease (PKD)	Measure food carefully. They have a sede	Brush weekly, increasing to bi-weekly dur
Burmese	Prone to diabetes mellitus and hypokalem	High-protein diet. Monitor water intake o	Very easy to groom. Wipe down weekly v
Chartreux	Prone to patellar luxation (flopping knee)	Monitor diet closely: they have a robust a	Comb their thick double coat weekly. Do

Figure 5.2.65 Breed-Specific Tips Database

```

router.get("/tips/{breed}")
async def get_breed_tips(breed: str):
    try:
        # 1. Fetch exact health data from your own Supabase database
        db_response = supabase.table("breed_health_tips").select("*").ilike("breed_name", f"%{breed}%").execute()

        # Default generic tips if breed isn't in your database yet
        tips = [
            f"Make sure your {breed} gets plenty of fresh water everyday!",
            "Regular vet checkups are the key to a long, healthy life.",
            "Keep their coat clean with regular brushing."
        ]

        # If the breed IS in your database, use your custom medical tips!
        if len(db_response.data) > 0:
            breed_info = db_response.data[0]
            tips = [
                f"🧬 Genetics: {breed_info['genetic_risks']}",
                f"🍖 Nutrition: {breed_info['nutrition_needs']}",
                f"🐾 Grooming: {breed_info['grooming_tips']}"
            ]

        # 2. Get the cute picture from The Cat API
        api_key = os.getenv("PET_API_KEY")
        base_url = "https://api.thecatapi.com/v1/images/search"
        api_response = requests.get(base_url, headers={"x-api-key": api_key})

        image_url = None
        if api_response.status_code == 200:
            image_data = api_response.json()
            if len(image_data) > 0:
                image_url = image_data[0].get("url")

        # 3. Return everything to Flutter!
        return {
            "tips": tips,
            "breed_image": image_url
        }

    except Exception as e:
        print(f"Error fetching data: {e}")
        return {"tips": ["Keep your pet hydrated!"], "breed_image": None}

```

Figure 5.2.66 Specific Breed Tips Logic

Users can also post updates, share photos and comment on other posts. All the community posts are stored in the Supabase Database and the feed is rendered dynamically from the backend response. This creates an interactive space for pet owners to exchange experiences and advice.

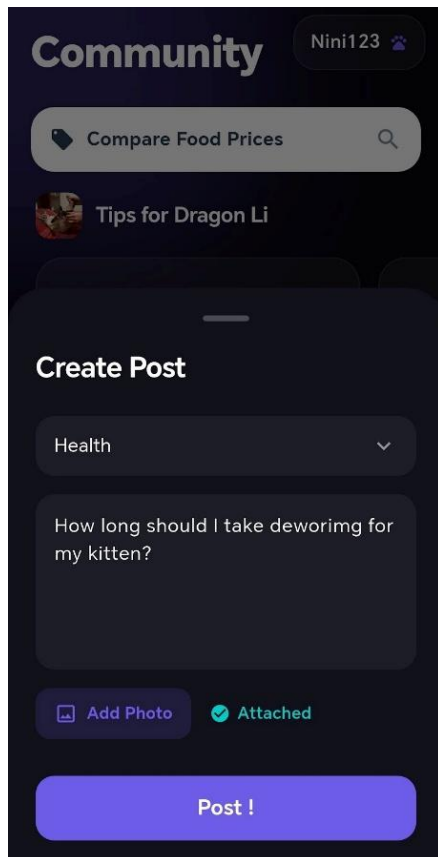


Figure 5.2.67 Post Creation

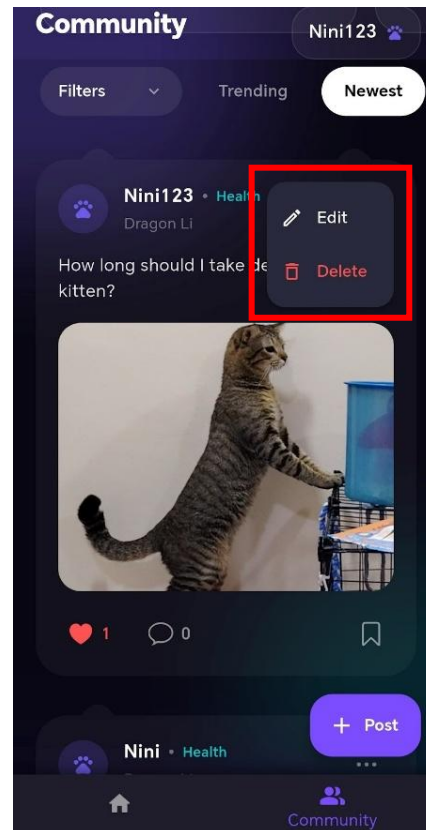


Figure 5.2.68 Edition or Deletion

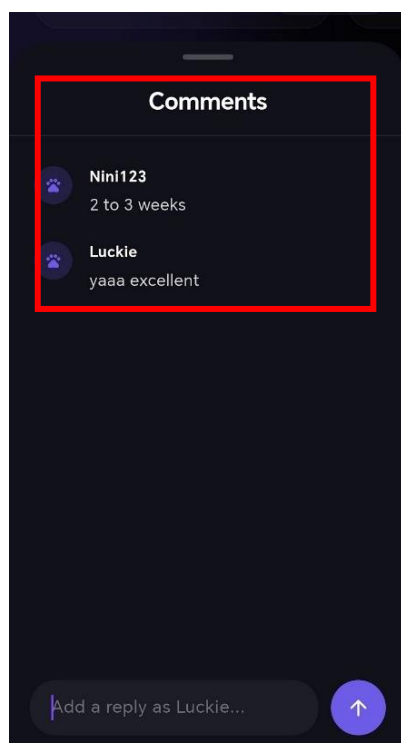


Figure 5.2.69
Provide Comment to Posts

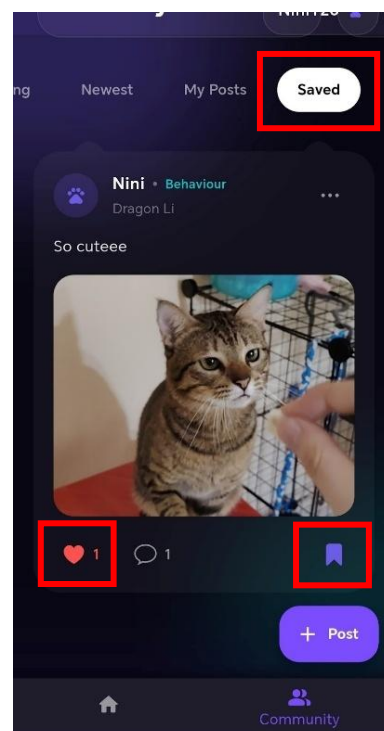


Figure 5.2.70
Saved and Like Posts

5.2.10 Price Tracker Development

The Price Tracker feature is accessible within the Community Screen, allowing users to search for pet products and compare prices across multiple online platforms in real-time. When users submit a product search query, the backend will forward the query to the SerpAPI Google Shopping engine with the location set to Malaysia. The top 3 lowest priced products will then be displayed in the app for users to click on the link to purchase the product.

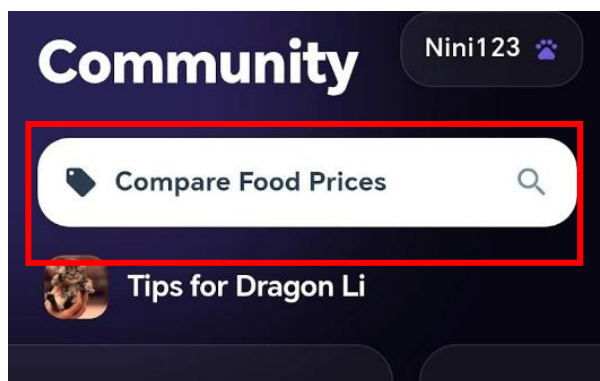


Figure 5.2.71

Search Bar for Price Comparison

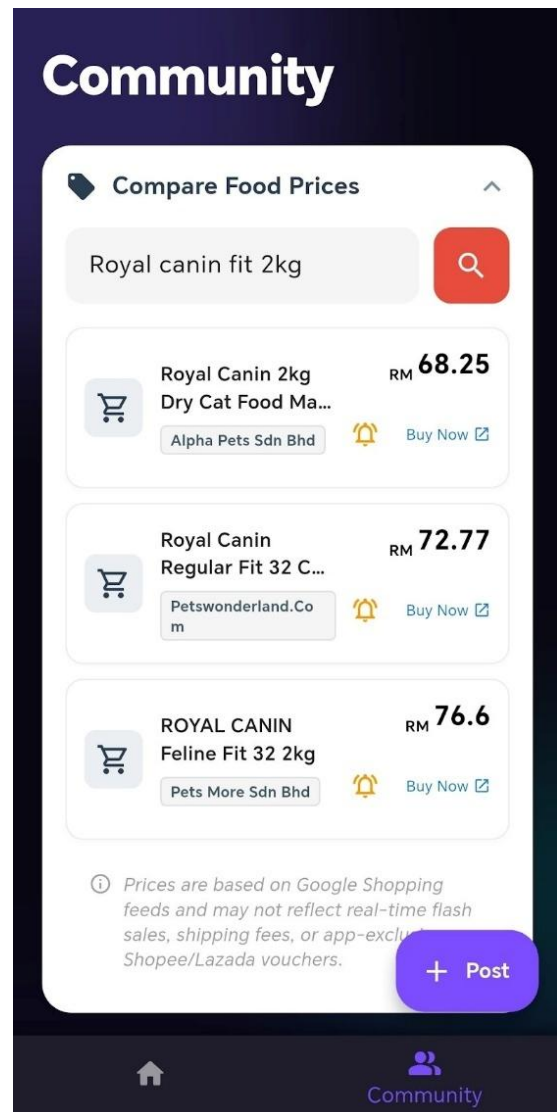


Figure 5.2.72

Top 3 Lowest-Price Product



Figure 5.2.73 Redirect User to Online Store

Users can also set a price alert for the desired product by tapping the Alert icon on a result card. After clicking, there will be a window pop out to ask for the target price from users. Users can enter their target price and click on the Save Button. Once submitted, the target price, product name, store link and search query will be saved into the Supabase Database. A background scheduler will start running a checking job every day at midnight. If the price of selected product drops to the target price or lower, the app will create a notification to alert users.

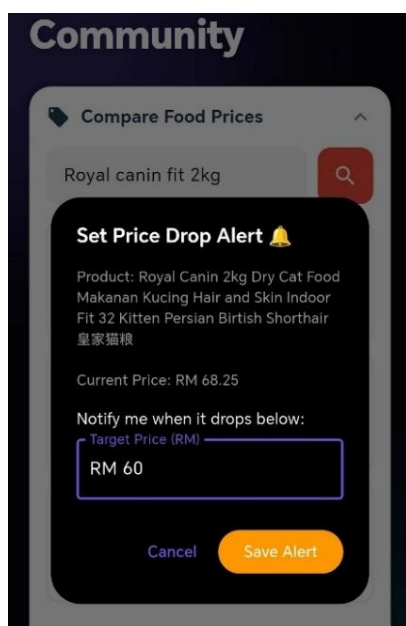


Figure 5.2.74

Target Price of Price Drop Alert

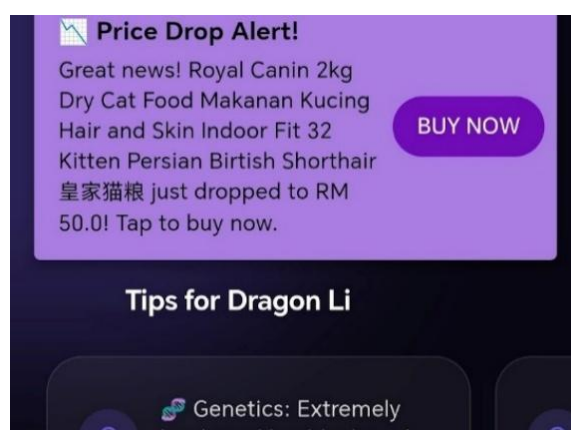


Figure 5.2.75

Notification Shown When Price Drop

```

@router.get("/search-price")
def search_lowest_price(query: str):
    search_term = query.strip()
    serpapi_key = os.getenv("SERPAPI_KEY")

    url = "https://serpapi.com/search"
    params = {
        "engine": "google_shopping",
        "q": search_term,
        "location": "Malaysia",
        "hl": "en",
        "gl": "my",
        "api_key": serpapi_key
    }

    response = requests.get(url, params=params)
    data = response.json()

    if "shopping_results" not in data:
        return {"error": "No products found."}

    shopping_results = data["shopping_results"]
    valid_items = [item for item in shopping_results if "extracted_price" in item]
    valid_items.sort(key=lambda x: x["extracted_price"])
    print("--- SPYING ON GOOGLE SHOPPING RESULTS ---")
    for item in valid_items:
        print(f"Store Found: {item.get('source', 'Unknown')} | Price: RM {item.get('extracted_price')}")
    print("-----")

```

Figure 5.2.76 Query Searching Using SerpAPI

```

@router.post("/set-alert")
async def set_price_alert(alert: PriceAlertRequest):
    """Saves the user's target price to the database"""
    try:
        data = {
            "user_id": alert.user_id,
            "search_query": alert.search_query,
            "product_name": alert.product_name,
            "target_price": alert.target_price,
            "store_link": alert.store_link
        }
        supabase.table("price_alerts").insert(data).execute()
        return {"status": "success", "message": "Alert saved!"}
    except Exception as e:
        return {"status": "error", "message": str(e)}

```

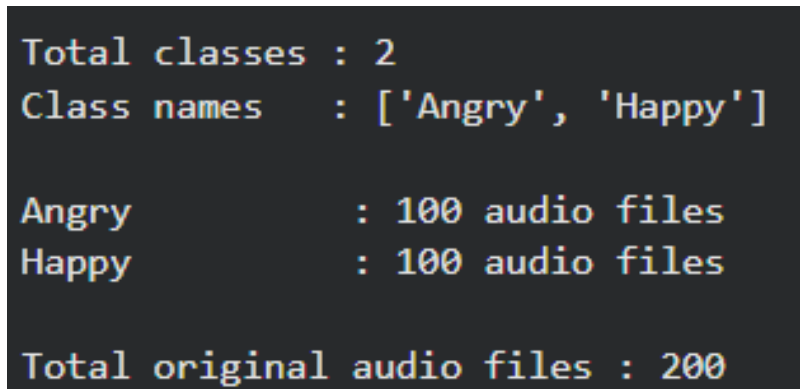
Figure 5.2.77 Price Dop Alert Logic

5.3 Pet Emotion Interpretation Model Training

5.3.1 Sound Interpretation Model

5.3.1.1 Dataset

The sound classification model is trained on a custom-compiled cat audio dataset consisting of two behavioural sound categories: Angry and Happy. The dataset contained 100 audio files per class, giving a total of 200 original recordings. All files are in .mp3 or .wav format. The two-class structure is deliberately kept balanced to prevent models from developing a bias toward the more represented category during training.



```
Total classes : 2
Class names   : ['Angry', 'Happy']

Angry         : 100 audio files
Happy         : 100 audio files

Total original audio files : 200
```

Figure 5.3.1 Dataset Summary in Sound Model

5.3.1.2 Data Preprocessing

MFCC Feature Extraction

Raw audio waveforms cannot be fed directly into a neural network in a meaningful way. Therefore, each audio file is converted into a Mel-Frequency Cepstral Coefficient (MFCC) representation. This is a compact and frequency-domain feature that makes the model suitable for classifying vocal and tonal patterns in pet sounds. Each file is loaded at a standardised sampling rate of 22050Hz using Librosa library. 40 MFCC coefficients are extracted per frame and the resulting matrix is transposed to produce a time-series array of shape (timesteps, 40). To satisfy the network's fixed input size, every MFCC sequence is either zero-padded at the end or truncated to a consistent length of 130 timesteps, providing a final feature shape of (130,40) per sample.

```

N_MFCC = 40
SR = 22050
# — 3a. Feature extraction —————
def extract_mfcc(audio, sr=SR, n_mfcc=N_MFCC):
    """Convert raw audio waveform → MFCC matrix (time_steps × n_mfcc)."""
    mfcc = librosa.feature.mfcc(y=audio, sr=sr, n_mfcc=n_mfcc)
    return mfcc.T          # shape: (time_steps, 40)

# — 3b. Fixed-length padding / truncation —————
def process_mfcc(mfcc_data, max_len=MAX_LEN):
    """Pad (zeros) or truncate MFCC to a fixed length for the NN."""
    if mfcc_data.shape[0] < max_len:
        pad_width = max_len - mfcc_data.shape[0]
        return np.pad(mfcc_data, ((0, pad_width), (0, 0)), mode='constant')
    return mfcc_data[:max_len, :]

```

Figure 5.3.2 MFCC Function in Sound Model

Z-Score Normalisation

After completed extraction, all MFCC features are normalised using Z-score standardisation, subtracting the training set mean and dividing by the standard deviation. This ensures the model receives inputs with a consistent numerical scale which stabilises gradient updates and speeds up convergence during training. The normalisation statistics are saved as `mfcc_mean.py` and `mfcc_std.npy` to make sure identical scaling can be applied at inference time.

```

# — 3e. Z-score normalisation —————
mfcc_mean = np.mean(x_raw, axis=0)
mfcc_std  = np.std(x_raw, axis=0)
x_scaled  = (x_raw - mfcc_mean) / (mfcc_std + 1e-6)
print("\n MFCC normalised (mean=0, std=1 per coefficient)")

```

Figure 5.3.3 Z-Score Normalisation in Sound Model

Data Augmentation

With only 100 dataset per class, training a deep learning model without data augmentation will increase the opportunity to get an overfitting result. Therefore, to address this problem, each audio clip is passed through 3 additional augmentation transformations to get totally 4 version per original file:

1. Unmodified original recording
2. Gaussian noise recording (factor = 0.02)
3. Pitch shift up recording (+2 semitones)

4. Pitch shift down (-2 semitones)

This expansion increases the dataset from 200 to 800 samples in total and also maintains the balance of class perfectly at 400 samples per class. Each augmented variant was processed through the same MFCC extraction and padding pipeline to get the identical input shape (130,40).

```
# — 3c. Augmentation pipeline —————
def load_and_augment(file_path):
    try:
        audio, sr = librosa.load(file_path, sr=SR)
        return [
            process_mfcc(extract_mfcc(audio, sr)),
            process_mfcc(extract_mfcc(
                audio + np.random.normal(0, 0.02, len(audio)), sr)),
            process_mfcc(extract_mfcc(
                librosa.effects.pitch_shift(audio, sr=sr, n_steps=2), sr)),
            process_mfcc(extract_mfcc(
                librosa.effects.pitch_shift(audio, sr=sr, n_steps=-2), sr)),
        ]
    except Exception:
        return []
```

Figure 5.3.4 Data Augmentation Function in Sound Model

```
=====
SECTION 3 – DATA PREPROCESSING (MFCC + AUGMENTATION)
=====

Loading and augmenting audio files...
Total samples after augmentation : 800
MFCC feature shape per sample    : (130, 40) → (130 time-steps × 40 coefficients)
```

Figure 5.3.5 Final Dataset Size in Sound Model

5.3.1.3 Dataset Splitting

The dataset is divided into 3 non-overlapping subsets using a stratified two-stage splitting strategy. Stratification is applied at both stages to guarantee the balance of class across all 3 subsets. In the first stage, 70% of data is assigned to the training set, the remaining 30% held aside as a temporary pool. In the second stage, the temporary pool is split equally, producing a validation set (15%) and test set (15%). The training set is used to update model weights during backpropagation. The validation set is used to monitor generalisation performance at the end of each epoch, serving as the signal for both early stopping and learning rate reduction. The test set is kept entirely unseen through all training and tuning decisions. It is only accessed once at the final evaluation stage to produce an unbiased performance report.

```
# Stage 1 – 70% train, 30% temp
x_train, x_temp, y_train_labels, y_temp_labels = train_test_split(
    x_scaled, y_raw, test_size=0.30, stratify=y_raw, random_state=SEED)

# Stage 2 – split temp equally → 15% val, 15% test
x_val, x_test, y_val_labels, y_test_labels = train_test_split(
    x_temp, y_temp_labels, test_size=0.50,
    stratify=y_temp_labels, random_state=SEED)
```

Figure 5.3.6 Dataset Splitting in Sound Model

Training	(70%)	:	560 samples
Validation	(15%)	:	120 samples
Test	(15%)	:	120 samples
Total		:	800 samples

Figure 5.3.7 Dataset Distribution in Sound Model

5.3.1.4 Model Training

The model follows a CNN-BiLSTM architecture with an attention mechanism. The network begins with a convolutional block which is a Conv1D layer with 64 filters, kernel size 3, ReLU activation and L2 regularisation, followed by batch normalisation, max pooling with a pool size of 2 and a dropout layer at 0.4 rate. This block is responsible for extracting short-range spectral patterns from MFCC input.

```
input_shape = x_train.shape[1:]
inputs      = layers.Input(shape=input_shape)

# — CNN block —
x = layers.Conv1D(64, 3, padding='same', activation='relu', kernel_regularizer=regularizers.l2(0.001))(inputs)
x = layers.BatchNormalization()(x)
x = layers.MaxPooling1D(2)(x)
x = layers.Dropout(0.4)(x)
```

Figure 5.3.8 CNN Block

The output is then passed into a BiLSTM with 128 units per direction, totaling 256 units of hidden state. This sequence bidirectionally allows the model to draw context from both earlier and future time steps simultaneously, which is particularly valuable for sound classification as the beginning and end of vocalisation will carry meaning. Internal LSTM dropout of 0.3 applied to reduce recurrent co-adaption.

```
# — BiLSTM block —————
x = layers.Bidirectional(
    layers.LSTM(128, return_sequences=True, dropout=0.3, recurrent_dropout=0.0)
)(x)
```

Figure 5.3.9 BiLSTM Block

Then, an Attention Mechanism is processed after the BiLSTM block. A single dense layer with a Tanh activation produces a scalar attention score for each 130 timesteps and then normalised into a probability distribution using softmax. These weights are multiplied element-wise with the BiLSTM outputs, effectively amplifying the most informative frames and suppressing background noise or silent padding. Its outputs are summed across the time dimension to produce a single fixed-length content vector that represents the audio's most important signature.

```
# — Attention mechanism —————
att = layers.Dense(1, activation='tanh')(x)
att = layers.Flatten()(att)
att = layers.Activation('softmax')(att)
att = layers.RepeatVector(256)(att)
att = layers.Permute([2, 1])(att)
out = layers.Multiply()([x, att])
out = layers.Lambda(lambda z: K.sum(z, axis=1))(out)
```

Figure 5.3.10 Attention Mechanism

This context vector is passed through a Dense Layer of 64 units with ReLU activation and L2 regularisation, followed by batch normalisation, dropout at 0.5 and a final softmax output layer to produce a probability distribution over two classes. The model is compiled with Adam optimiser at a learning rate of 0.0005 and categorical cross-entropy loss.

```
# — Dense classification block —————
x = layers.Dense(64, activation='relu', kernel_regularizer=regularizers.l2(0.001))(out)
x = layers.BatchNormalization()(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(len(labels_list), activation='softmax')(x)

model = models.Model(inputs, outputs)
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.0005),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

Figure 5.3.11 Dense Layer and Optimiser

Training is conducted for up to 60 epochs with a batch size of 32, supported by two callbacks to improve performance. Early stopping is one of the callbacks with a patience of 12 epochs to monitor validation loss and restore the best weights once training stops. Besides, learning rate reduction halves the learning rate (down to minimum of $1e-6$) whenever validation loss fails to improve for 5 consecutive epochs. As shown in Figure 5.3.11, the model converged by approximately epoch 28, with training accuracy reaching 92% and validation accuracy stabilising at 88 to 89%. The small gap between the two curves suggests good generalisation, as the model performs consistently on unseen data without significant overfitting. On the loss graph, both training and validation loss decrease rapidly at the beginning. As training progresses, the loss curve continues to decline more gradually and remain relatively close to each other. This also indicates that the model is not memorising the training data, but instead learning generalisable features.



Figure 5.3.12 Sound Model Training Loss and Accuracy Curves

```
early_stop = EarlyStopping(monitor='val_loss', patience=12, restore_best_weights=True, verbose=1)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=5, min_lr=1e-6, verbose=1)

history = model.fit(
    x_train, y_train,
    batch_size = BATCH_SIZE,
    epochs = EPOCHS,
    validation_data = (x_val, y_val),
    callbacks = [early_stop, reduce_lr],
    verbose = 1
)
```

Figure 5.3.13 Model Training in Sound Model

Layer (type)	Output Shape	Param #	Connected to
input_layer_2 (InputLayer)	(None, 130, 40)	0	-
conv1d_4 (Conv1D)	(None, 130, 64)	7,744	input_layer_2[0]...
batch_normalizatio... (BatchNormalizatio...	(None, 130, 64)	256	conv1d_4[0][0]
max_pooling1d_2 (MaxPooling1D)	(None, 65, 64)	0	batch_normalizat...
dropout_2 (Dropout)	(None, 65, 64)	0	max_pooling1d_2[...
bidirectional_2 (Bidirectional)	(None, 65, 256)	197,632	dropout_2[0][0]
dense_6 (Dense)	(None, 65, 1)	257	bidirectional_2[...
flatten_2 (Flatten)	(None, 65)	0	dense_6[0][0]
activation_2 (Activation)	(None, 65)	0	flatten_2[0][0]
repeat_vector_2 (RepeatVector)	(None, 256, 65)	0	activation_2[0][...
permute_2 (Permute)	(None, 65, 256)	0	repeat_vector_2[...
multiply_2 (Multiply)	(None, 65, 256)	0	bidirectional_2[...] permute_2[0][0]
lambda (Lambda)	(None, 256)	0	multiply_2[0][0]
dense_7 (Dense)	(None, 64)	16,448	lambda[0][0]
batch_normalizatio... (BatchNormalizatio...	(None, 64)	256	dense_7[0][0]
dropout_3 (Dropout)	(None, 64)	0	batch_normalizat...
dense_8 (Dense)	(None, 2)	130	dropout_3[0][0]
Total params: 222,723 (870.01 KB)			
Trainable params: 222,467 (869.01 KB)			
Non-trainable params: 256 (1.00 KB)			

Figure 5.3.14 Sound Model Architecture

5.3.1.5 Performance Evaluation

The trained model is evaluated on the held out test set of 120 which had remained completely unseen throughout training. The results demonstrated strong and balanced performance across all evaluation dimensions.

Classification Report

The overall test accuracy reached 89%. For the Angry class, precision is 0.90 and recall is 0.88, producing an F1-score of 0.89. For Happy classroom precision scored at 0.89 and recall is 0.90, yielding an F1-score of 0.89. Both macro averaged and weighted-averaged F1-score settled at 0.89, indicating that the model performs consistently across all classes. It also shows that the model is reliable in correctly identifying both emotions while maintaining low misclassification rates.

	precision	recall	f1-score	support
Angry	0.90	0.88	0.89	60
Happy	0.89	0.90	0.89	60
accuracy			0.89	120
macro avg	0.89	0.89	0.89	120
weighted avg	0.89	0.89	0.89	120

Figure 5.3.15 Classification Report in Sound Model

Confusion Matrix

The confusion matrix demonstrates that the model performs well across both emotion classes. Out of 60 Angry test samples, there are 53 correctly classified, with the other seven being incorrectly identified as Happy. Similarly, for 60 Happy test samples, 54 are correctly identified while 6 are wrongly classified as Angry. In this case, the model produced an uneven distribution of errors between class types. Both accuracy levels show that the model can consistently distinguish between emotions and produces equal amounts of errors. The high number of correct predictions in both categories confirms that the model has learned meaning features and can generalise effectively to new data.

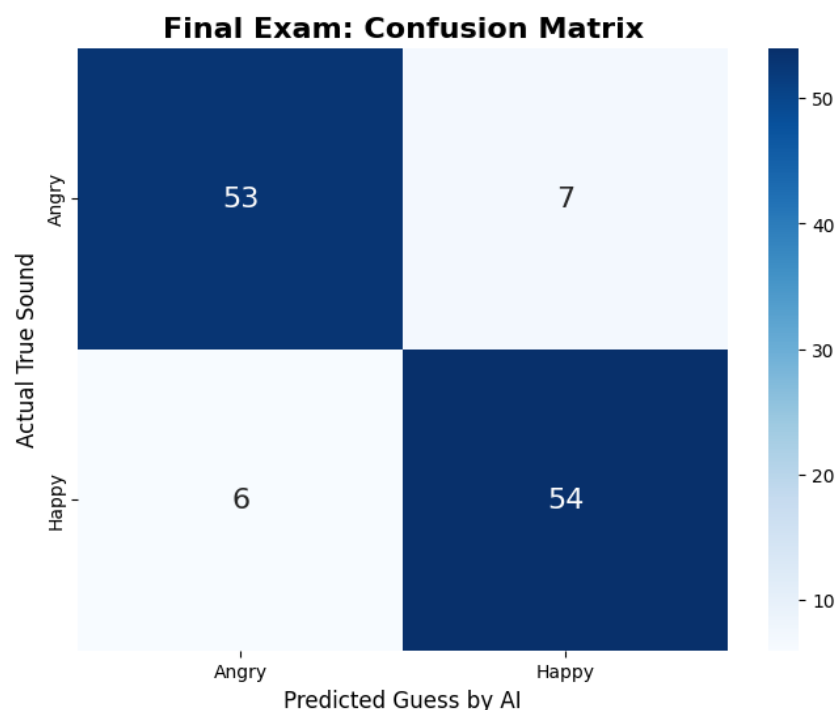


Figure 5.3.16 Confusion Matrix in Sound Model

Receiver Operating Characteristic (ROC) Curve

The ROC curve for both classes show a high Area Under Curve (AUC) score, indicating the model maintains a strong true positive rate across a wide range of decision thresholds. Both Angry and Happy classes achieved an AUC of 0.89. Since a perfect classifier scores 1.0 and a random guess scores 0.5, an AUC of 0.89 suggests that the model has an 89% probability of correctly predicting a ranking chosen positive sample higher than a negative one. The Angry class shows a steeper initial rise at low false positive rate, indicating strong early discriminative performance. In contrast, the true positive rate in the Happy class increases gradually and consistently with the threshold. Overall, both curves tend to move to the upper left corner of ROC space, which reflects good and balanced classification performance for both categories of emotions.

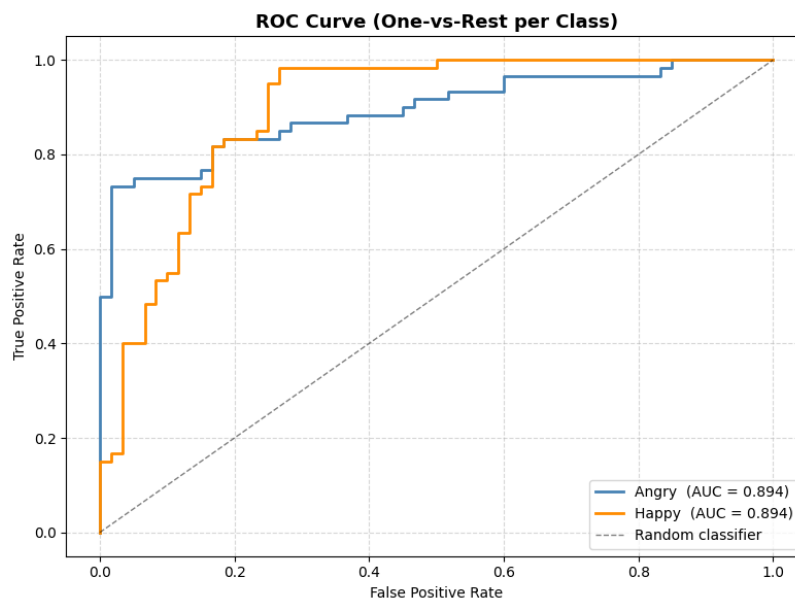


Figure 5.3.17 ROC Curve in Sound Model

Precision-Recall Curve

The precision-recall curve for both classes remains at high precision levels even at high recall threshold, reflecting a strong average precision (AP) score for each class. This is particularly important because the classes are balanced and a high AP in a balanced dataset is a real indicator of the model's quality and not an artifact of class imbalance. The Angry class achieved an AP of 0.92 while the Happy class recorded 0.86, with the Angry curve precision at 1.0 from recall 0.0 to approximately 0.75 before gradually declining. This means that the model almost makes zero false positive Angry predictions at a high confidence threshold. Similarly, the Happy curve

starts slightly lower at 0.84 and declines more gradually across its range. These patterns are in accordance with the above confusion matrix in which the Happy class predicted more false positives than Angry class.

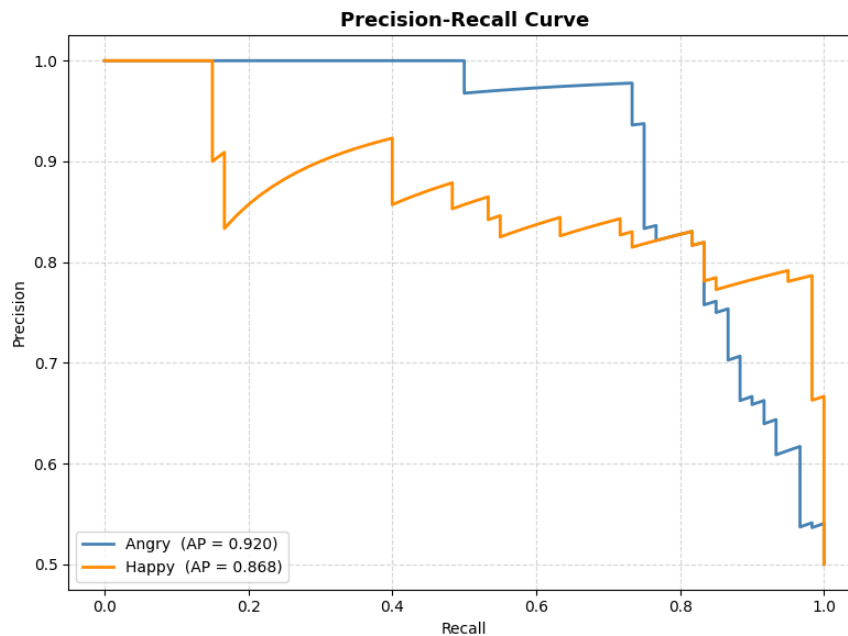


Figure 5.3.18 Precision-Recall Curve in Sound Model

In short, the evaluation results on all 4 metrics consistently confirm that the CNN-BiLSTM with attention model has learned meaningful acoustic patterns and generalises reliably to unseen cat audio samples. These results confirm that the CNN-BiLSTM architecture with the attention mechanism, data augmentation and the improved three-way training split, generated a sound classification model that is accurate and reliable for deployment in real-time sound interpretation feature of the application.

5.3.1.6 Model Comparison

The table below presents a direct side-by-side comparison of the FYP1 baseline model and the improved FYP2 model across all key evaluation metrics.

Table 5.3.1 Model Comparison in Sound Model

Performance Metric	FYP1	FYP2	Improvement
Accuracy	85%	89%	+4%
Precision (weighted)	87%	89%	+2%
Recall (weighted)	85%	89%	+4%
F1-score (weighted)	84%	89%	+5%
Number of classes	10	2	-

Although the number of target classes is reduced from 10 in FYP1 to 2 in FYP2, this decision is deliberate and directly tied to the specific use case of the app, differentiating between Angry and Happy cat vocalisation for sound interpretation feature. More importantly, the performance gains observed cannot be attributed to a reduction in the number of classes alone, as several targeted changes to the training pipeline and model architecture were made in FYP2, yielding a genuinely more accurate and reliable model. Specifically, the unstable Mixup data generator is removed and replaced by a fixed and pre-processed dataset, which removes the erratic gradient signals that caused oscillating training curves in FYP1. The dataset splitting strategy is upgraded from a naive 80/20 train-test split to a proper three-way 70/15/15 stratified split, ensuring that early stopping and learning rate scheduling are guided by a dedicated validation set instead of the test data. Furthermore, L2 regularisation is applied to both convolutional and dense layers and the attention context vector aggregation has been changed from GlobalAveragePooling1D to a direct K.sum() operation to keep the full contribution of high-attention frames. The evaluation results clearly show the combined effect of these changes.

Overall accuracy jumped from 85% to 89%, weighted recall rose from 85% to 89% and the weighted F1-score increased from 84% to 89%. Beyond the aggregate numbers, the per-class

performance in FYP2 was notably more balanced and stable compared to FYP1, where results varied widely across categories. For example, in FYP1's model, Resting and Warning had perfect F1-scores of 1.00 but Paining lagged far behind at 0.57 and Angry was over-predicted with precision down at 0.62 even though recall hit 1.00. However, in FYP2, both classes achieved identical F1-scores of 0.89 with near-symmetric precision and recall values, confirming that the improvements were structural and consistent rather than incidental. This demonstrates that even with a more focused classification scope, a well-regularised training pipeline and a cleaner architecture can produce a model that is meaningfully more accurate, more stable and more evenly balanced across its target categories.

	precision	recall	f1-score	support
Angry	0.62	1.00	0.76	8
Defense	0.86	0.75	0.80	8
Fighting	0.73	1.00	0.84	8
Happy	1.00	0.75	0.86	8
HuntingMind	0.88	0.88	0.88	8
Mating	1.00	0.88	0.93	8
MotherCall	1.00	0.75	0.86	8
Paining	0.67	0.50	0.57	8
Resting	1.00	1.00	1.00	8
Warning	1.00	1.00	1.00	8
accuracy			0.85	80
macro avg	0.87	0.85	0.85	80
weighted avg	0.87	0.85	0.85	80
Accuracy	: 0.8500			
Precision	: 0.8741			
Recall	: 0.8500			
F1 Score	: 0.8498			

Figure 5.3.19 Sound Model Classification Report for FYP1

5.3.2 Visual Interpretation Model

5.3.2.1 Dataset

The visual body language classification model is trained on a custom image dataset containing two behavioural categories: `aggressive_defensive` and `friendly_active`. These two labels represent the two most visually distinguishable cat body postures used in the dual-model sound interpretation pipeline. For example, an aggressive or defensive stance and a friendly or actively engaged posture. All the images are in standard .jpg format and the dataset contain 303 real images per class.

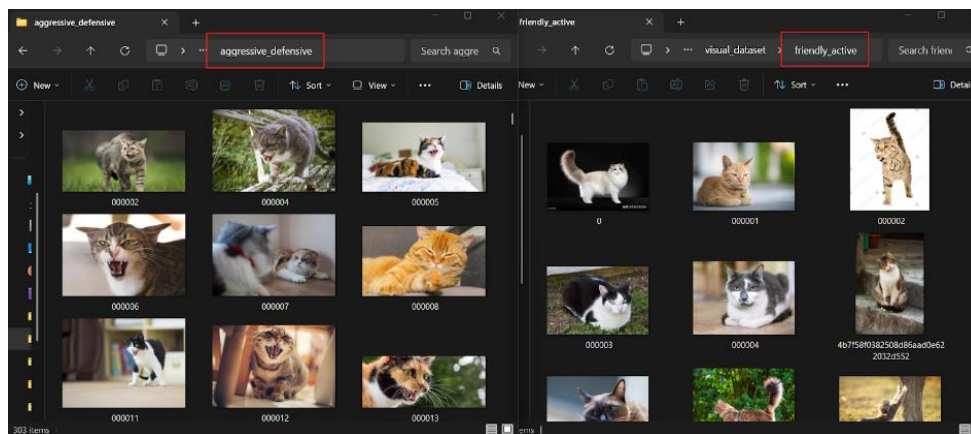


Figure 5.3.20 Visual Dataset

5.3.2.2 Data Preprocessing

Since the model uses MobileNetV2 as its backbone and a convolutional neural network originally trained on the ImageNet dataset, all input images are resized to 224×224 pixels to match the expected input dimension of the architecture. Pixel values are rescaled from their original range of 0-255 to a normalised range of 0.0 to 1.0 by dividing by 255. This helps to stabilize the gradient updates during training and aligns with the preprocessing convention used during MobileNetV2's original training.

To address the limited size of the real image collection and reduce the risk of overfitting, two rounds of augmentation are applied. The first is a synthetic data generation phase applied directly to the raw dataset folders before splitting, using ImageDataGenerator configured with rotation up to 30 degrees, width and height shifts of 0.2, shear of 0.2, zoom of 0.3, horizontal flipping and brightness variation between 0.8 and 1.2. This offline augmentation generated new saved image files until each class reached 600 images, permanently expanding the database on disk.

```
# --- 3. SYNTHETIC DATA GENERATION ---
# create new "fake but realistic" img using transformations
# solve data imbalance/small dataset problem
print("\n🚀 Starting Synthetic Data Generation (Target: 600 per class)...")
syn_augmentor = ImageDataGenerator(
    rotation_range=30,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.3,
    horizontal_flip=True,
    brightness_range=[0.8, 1.2], fill_mode='nearest'
)
```

Figure 5.3.21 Synthetic Data Generation in Visual Model

```
aggressive_defensive      : 600 original images
friendly_active           : 600 original images

Target per class   : 600
Total after aug    : ~1200 images
```

Figure 5.3.22 Target Dataset in Visual Model

The second round of augmentation is applied online during training through the training data generator by using more aggressive settings which have a rotation up to 40 degrees, shifts of 0.25, shear of 0.3, zoom of 0.4, horizontal flipping and brightness between 0.7 and 1.3. This additional online augmentation ensures the model is exposed to new variations in every training epoch, further improving generalisation and preventing the model from memorizing the fixed dataset. The validation and test generators are used only for rescaling with no augmentation to ensure fair and unmodified evaluation.

```
train_datagen = ImageDataGenerator(
    rescale=1./255, rotation_range=40, width_shift_range=0.25,
    height_shift_range=0.25, shear_range=0.3, zoom_range=0.4,
    horizontal_flip=True, brightness_range=[0.7, 1.3], fill_mode='nearest'
)
test_val_datagen = ImageDataGenerator(rescale=1./255)

# Training generator: including augmentation, help model generalize better
train_gen = train_datagen.flow_from_directory(
    os.path.join(SPLIT_DIR, 'train'), target_size=IMG_SIZE, batch_size=BATCH_SIZE, class_mode='categorical', shuffle=True
)
# Val/Test generator: Only rescale ensure fair evaluation
val_gen = test_val_datagen.flow_from_directory(
    os.path.join(SPLIT_DIR, 'val'), target_size=IMG_SIZE, batch_size=BATCH_SIZE, class_mode='categorical', shuffle=False
)
test_gen = test_val_datagen.flow_from_directory(
    os.path.join(SPLIT_DIR, 'test'), target_size=IMG_SIZE, batch_size=BATCH_SIZE, class_mode='categorical', shuffle=False
)
```

Figure 5.3.23 Train Data Generator in Visual Model

5.3.2.3 Dataset Splitting

After synthetic generation, the full 1200 images dataset is divided into 3 subsets using splitfolders library with a stratified ratio split. This split is followed by a 70/15/5 strategy in which 70% of the image is assigned to the training set, 15% to the validation set and the remaining 15% to the test set. A fixed random seed of 42 is used to ensure the split is reproducible. The usage of the training set is for updating model weights, the validation set for monitoring purpose during training and the test set for evaluation purpose.

```
print("\n Splitting expanded dataset into Train (70%), Val (15%), Test (15%)...")
splitfolders.ratio(DATASET_DIR, output=SPLIT_DIR, seed=42, ratio=(0.70, 0.15, 0.15), group_prefix=None)
```

Figure 5.3.24 Dataset Splitting in Visual Model

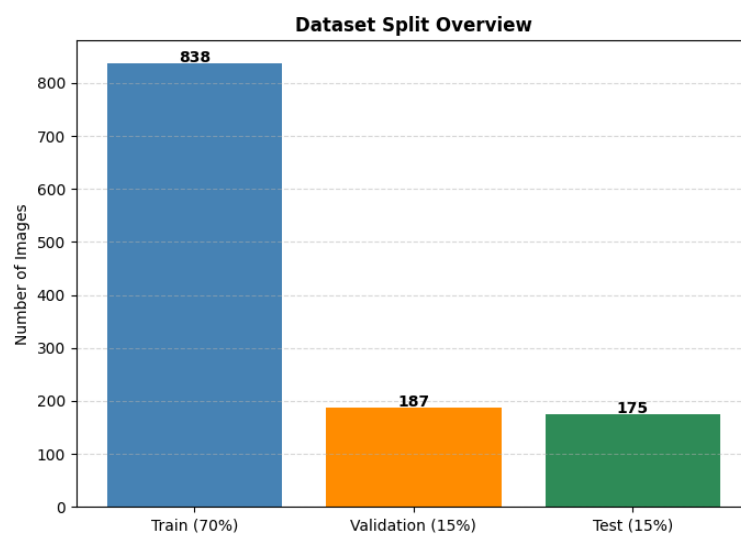


Figure 5.3.25 Splitting Result in Visual Model

5.3.2.4 Model Training

The visual model is built using transfer learning with MobileNetV2 as the pretrained backbone. MobileNetV2 is chosen for its lightweight architecture and strong feature extraction capability. It is originally trained on over million ImageNet images and has already learned rich low-level and mid-level visual features such as edges, texture and shaper that transfer effectively to new classification tasks. The `include_false=False` setting removed the original ImageNet classification head and attached a custom classification head in its place. A Global Average Pooling layer is built to collapse the spatial feature maps into a single vector, followed by a Dense layer of 256 units with ReLU activation and L2 regularisation, Batch Normalisation, Dropout and a final softmax output layer.

```

base_model = MobileNetV2(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
base_model.trainable = False

model = models.Sequential([
    base_model,
    layers.GlobalAveragePooling2D(),
    layers.Dense(256, activation='relu', kernel_regularizer=regularizers.l2(0.01)),
    layers.BatchNormalization(),
    layers.Dropout(0.5),
    layers.Dense(len(class_labels), activation='softmax')
])

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()

```

Figure 5.3.26 Sequential with MobileNetV2 Base and Custom Head

Model: "sequential_1"		
Layer (type)	Output Shape	Param #
=====		
mobilenetv2_1.00_224 (Functional)	(None, 7, 7, 1280)	2257984
global_average_pooling2d_1 (GlobalAveragePooling2D)	(None, 1280)	0
dense_2 (Dense)	(None, 256)	327936
batch_normalization_1 (Batch Normalization)	(None, 256)	1024
dropout_1 (Dropout)	(None, 256)	0
dense_3 (Dense)	(None, 2)	514
=====		
Total params: 2587458 (9.87 MB)		
Trainable params: 328962 (1.25 MB)		
Non-trainable params: 2258496 (8.62 MB)		

Figure 5.3.27 Visual Model Architecture

The training process is carried out in two phases. In phase 1, the MobileNetV2 backbone is frozen entirely and the custom classification head is trained for up to 20 epochs. This prevents the pretrained feature extractor from being disrupted by the large initial gradients from the untrained head. After completing phase 1, the backbone is unfrozen and the last 30 layers are made trainable for fine-tuning, while all earlier layers remain frozen. The learning rate will drop to 1e-5 during phase 2, applying only small and careful adjustment to the pretrained weights. Both phases used Early Stopping with patience of 8 epochs and ReduceLROnPlateau with a reduction factor of 0.5 and patience of 4 epochs to prevent overfitting. As shown in Figure 5.3.28, both training and validation loss curves dropped sharply from around 4.7 and 4.0 respectively at the start of phase 1, converging closely together and stabilising near 1.0 by epoch 15. Upon entering phase 2 at epoch 20, the fine tuning with the reduced learning rate produced a further gradual decline with both losses settling at approximately 0.85 by epoch 35. During both phases, the training and validation curves tracked each other closely with no

visible divergence, confirming that the two-phase training strategy successfully worked to prevent overfitting while allowing the model to progressively refine its feature representations from head training to backbone fine-tuning.

```
early_stop = EarlyStopping(monitor='val_loss', patience=8, restore_best_weights=True, verbose=1)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=4, min_lr=1e-6, verbose=1)

# Phase 1 - frozen backbone
print("\n🚀 Phase 1: Training classification head (backbone frozen)...")
history_p1 = model.fit(train_gen, epochs=20, validation_data=val_gen, callbacks=[early_stop, reduce_lr])

# Phase 2 - fine-tune last 30 layers
print("\n🔧 Phase 2: Fine-tuning last 30 layers of MobileNetV2...")
base_model.trainable = True
for layer in base_model.layers[:-30]:
    layer.trainable = False

model.compile(
    optimizer=tf.keras.optimizers.Adam(1e-5),
    loss='categorical_crossentropy',
    metrics=['accuracy'])
history_p2 = model.fit(train_gen, epochs=15, validation_data=val_gen, callbacks=[early_stop, reduce_lr])
```

Figure 5.3.28 Phase1 and Phase2 Training Blocks

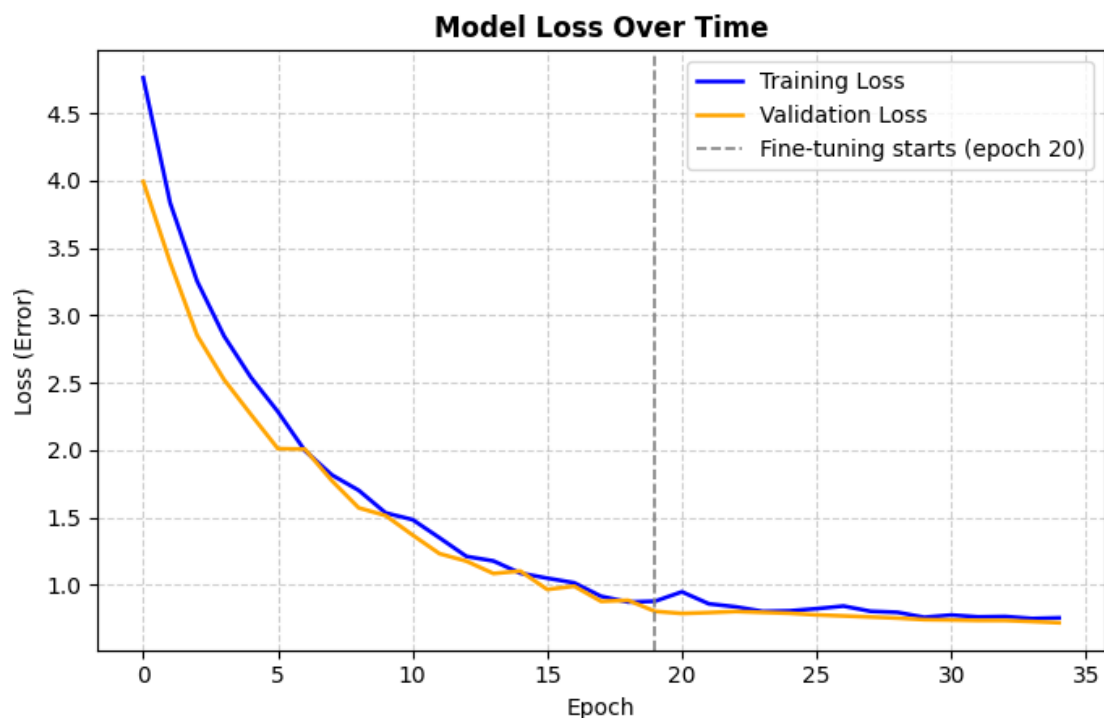


Figure 5.3.29 Visual Model Training Loss Curve

5.3.2.5 Performance Evaluation

The MobileNetV2 model is evaluated on the test set with 86 aggressive_defensive and 89 friendly_active. The results across all four evaluation methods confirm that the model learned meaningful visual features and generalised reliably to new images.

Classification Report

The overall test accuracy reached 87% with both classes achieving well-balanced and strong per-class metrics. The precision and recall of aggressive_defensive class are 0.88 and 0.85 respectively and its F1-score is 0.86. While the friendly_active class had precision of 0.86 and recall of 0.89 resulting in an F1-score of 0.87. Both macro-averaged and weighted-averaged F1-scores settled at 0.87, indicating that the model works well for both categories and is not biased towards one class. The near-identical score between two classes suggest that the model did not develop any directional bias during training despite the slight difference in support counts.

	precision	recall	f1-score	support
aggressive_defensive	0.88	0.85	0.86	86
friendly_active	0.86	0.89	0.87	89
accuracy			0.87	175
macro avg	0.87	0.87	0.87	175
weighted avg	0.87	0.87	0.87	175

Figure 5.3.30 Classification Report in Visual Model

Confusion Matrix

The confusion matrix is a better indicator of the success and failure of the model. Out of 86 aggressive_defensive samples, 73 are correctly identified and 13 are misclassified as friendly_active. Out of 89 friendly_active samples, 79 are classified and 10 are incorrectly predicted as aggressive_defensive. The almost symmetric error structure excludes any systematic bias in predictions of the model. The small number of errors on both sides indicates that the rest of the misclassifications are likely due to truly ambiguous postures, where the visual margin between aggressive and friendly body language is overlaps, rather than from any fundamental weakness in the model.

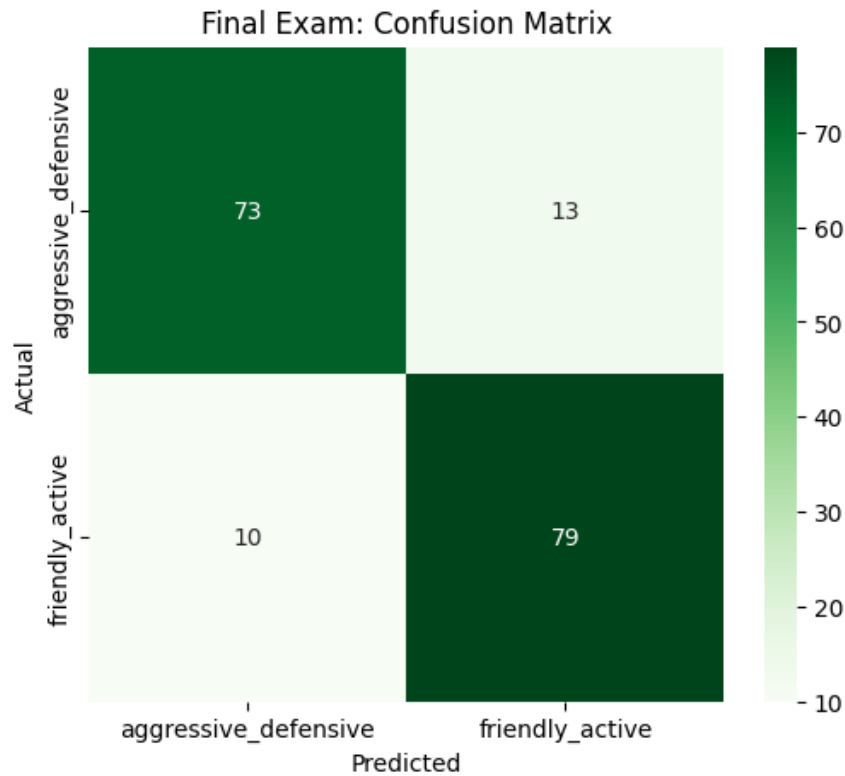


Figure 5.3.31 Confusion Matrix in Visual Model

Receiver Operating Characteristic (ROC) Curve

The ROC curve achieved an identical AUC of 0.91, placing the model firmly in the excellent discrimination range. Both curves bow substantially away from the diagonal random-classifier baseline and consistently occupy the upper-left region of ROC space throughout their entire range. This shows that the model maintains a strong true positive rate across all possible decision thresholds. The aggressive_defensive curve displays a steeper initial rise at very low false positive rates, reflecting that the model is particularly selective and confident when identifying aggressive postures at high confidence thresholds. In addition, the friendly_active curve follows a slightly more gradual but equally sustained path. Both curves eventually converge toward the top-right corner showing strong overall detection ability. These identical AUC scores further prove the model treats both categories fairly because they are consistent with their near-equal F1-scores in the classification report.

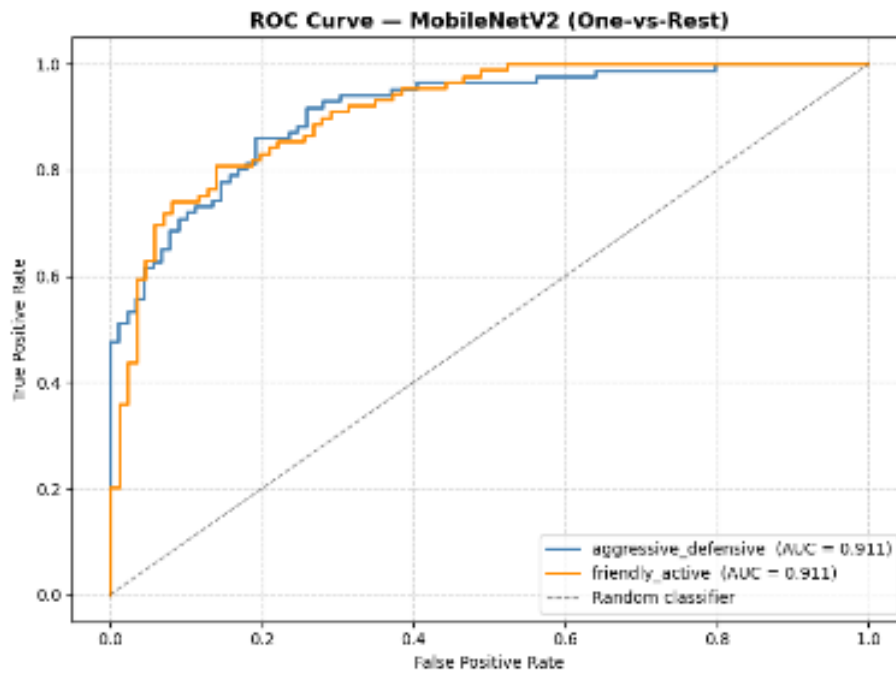


Figure 5.3.32 ROC Curve in Visual Model

Precision-Recall Curve

The Precision-Recall curve supports the results of the ROC analysis, with the `aggressive_defensive` class achieving an Average Precision of 0.917 and `friendly_active` recording 0.911. At the beginning, both curves hold precision at 1.0 from recall 0.0 to approximately 0.20, meaning the model makes virtually zero false positive predictions when operating at high confidence thresholds for either class. Beyond that point, both curves descend gradually but still remain above 0.90 precision all the way to recall 0.2. This is a good sign that the model's probability outputs are well-calibrated rather than throwing random guesses over the floor at the default threshold. The precision of `aggressive_defensive` is a little higher, which is consistent with the confusion matrix, where it got less false positives than `friendly_active`. Overall, both AP scores are higher than 0.5 (random guessing), showing that the model has truly learned useful information visual patterns instead of just guessing.

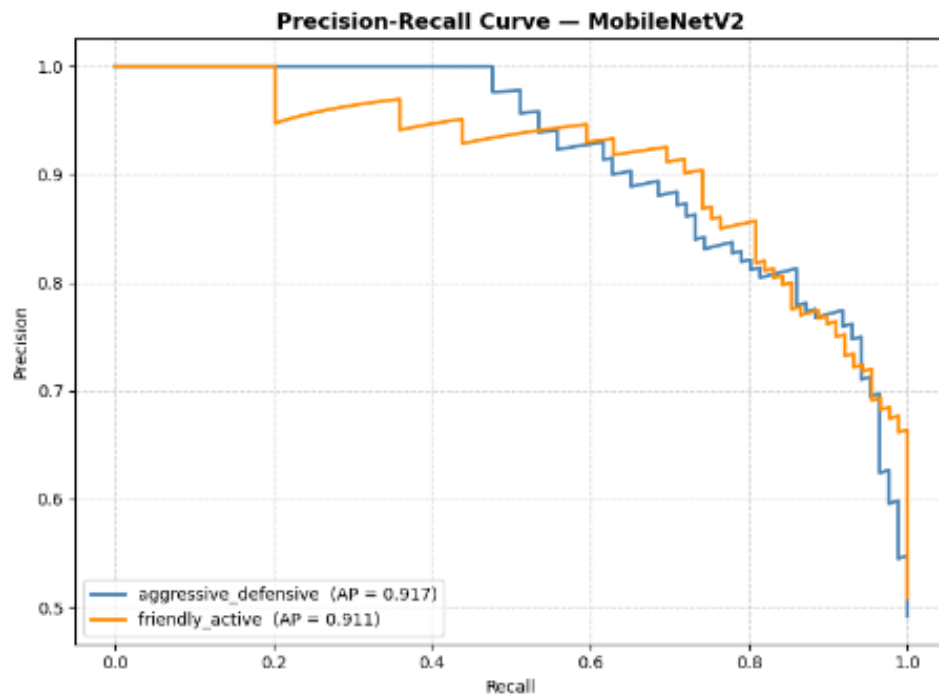


Figure 5.3.33 Precison-Recall Curve in Visual Model

To sum up, all 4 evaluations prove that the MobileNetV2 model successfully learned to distinguish between aggressive_defensive and friendly_active cat body postures with strong and balanced performance across accuracy, precision, recall, AUC and average precision. This proves that the two-phase transfer learning strategy which combined with synthetic data generation and regularisation, produced a model that understands the visual differences between the two behavioural categories rather than simply memorising training examples.

5.3.2.6 Model Comparison

The table below presents a direct side-by-side comparison of InceptionV3 Baseline Model and the MobileNetV2 Model across all key evaluation metrics.

Table 5.3.2 Model Comparison in Visual Model

Performance Metric	InceptionV3	MobileNetV2	Improvement
Accuracy	81%	87%	+6%
Precision (weighted)	81%	87%	+6%
Recall (weighted)	81%	87%	+6%
F1-score (weighted)	81%	87%	+6%

To validate the choice of MobileNetV2 as the production model for the application, it is benchmarked against the InceptionV3 model which has a deeper and more parameter-heavy architecture also pretrained on ImageNet. Both models are trained using the same data splits, identical two-phase training strategy, augmentation pipeline, callback configuration and custom classification head to ensure a fair comparison.

As a result shown in the table, MobileNetV2 outperformed InceptionV3 across every single metric despite being dramatically smaller. On overall test accuracy, MobileNetV2 achieved 87% compared to 81% for InceptionV3, representing an improvement of 6 percentage points on the same unseen test set. A similar improvement is observed in precision, recall and F1-score, all of which increased from 81% to 87%.

Besides that, MobileNetV2 achieved its superior accuracy with only 2,587,458 parameters, while InceptionV3 required 22,328,866, making InceptionV3 roughly 8.6 times larger yet still less accurate. This 88.4% reduction in model size comes with real practical benefits for a mobile application such as faster inference time on device, lower memory usage and quicker model loading.

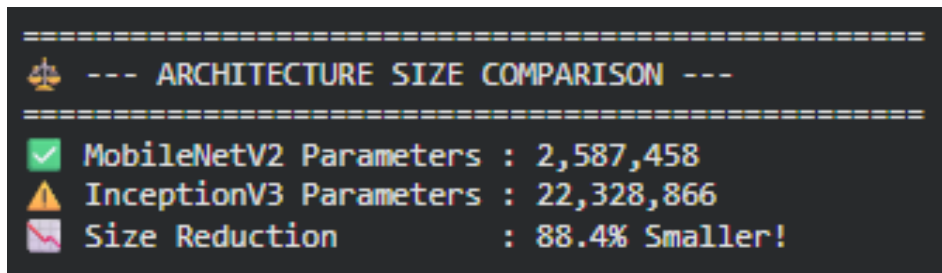


Figure 5.3.34 Parameter Size Comparison

Additionally, the chart comparing validation loss of these two approaches further corroborates this finding. Both models started from a similar loss of around 4.1 at epoch 0, but the InceptionV3 curve showed noticeably more instability. For example, it spiked at epoch 5 and again around epoch 15. On the other hand, MobileNetV2's validation loss descended more smoothly and consistently throughout the entire training run, reaching a lower final value by epoch 35. This difference in stability indicates that the bigger size of InceptionV3 is disadvantage on this small dataset as it causes the model to oscillate more as it fits the data with too many parameters than what the task need. By contrast, MobileNetV2 had a capacity that more closely fit the size of the dataset, yielding more stable gradients and ultimately better generalisation.

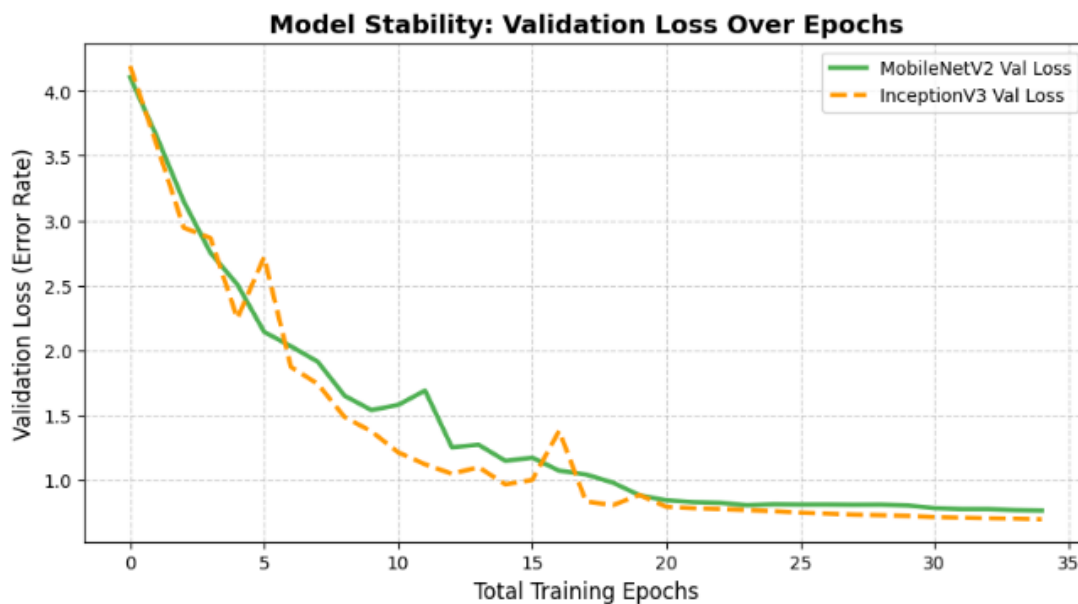
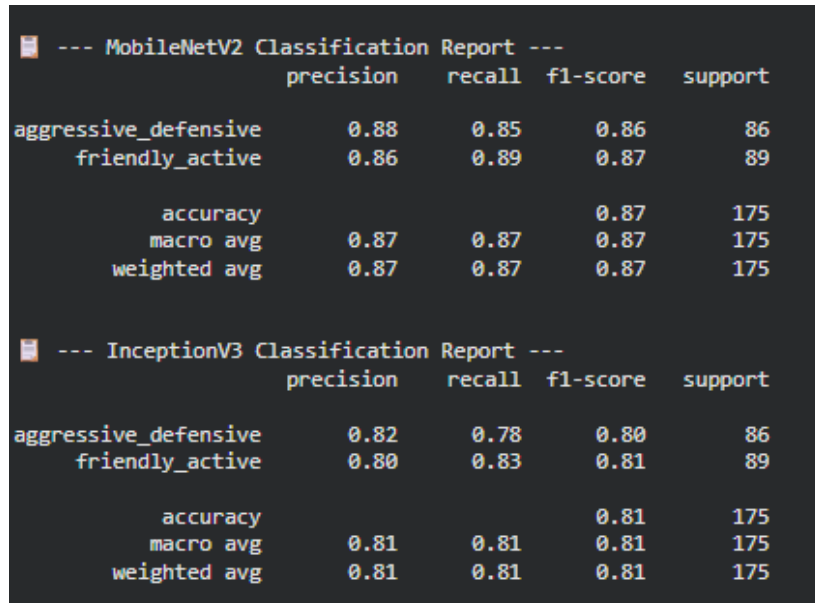


Figure 5.3.35 Validation Loss Comparison

In summary, these results confirm that MobileNetV2 was the right architectural choice for this application because it is not only lighter and faster, but also genuinely learned the visual

features of cat body language more effectively than a heavier alternative under the same training conditions.



```

--- MobileNetV2 Classification Report ---
              precision    recall  f1-score   support

aggressive_defensive      0.88      0.85      0.86         86
friendly_active           0.86      0.89      0.87         89

   accuracy              0.87              175
  macro avg              0.87              175
 weighted avg              0.87              175

--- InceptionV3 Classification Report ---
              precision    recall  f1-score   support

aggressive_defensive      0.82      0.78      0.80         86
friendly_active           0.80      0.83      0.81         89

   accuracy              0.81              175
  macro avg              0.81              175
 weighted avg              0.81              175

```

Figure 5.3.36 Classification Report for InceptionV3 Model

5.3.3 Personalised Model via Transfer Learning

5.3.3.1 Personalisation Strategy Overview

One of the limitations of a globally trained model is that it treats all cats as statistically equivalent. It learns from a broad dataset of general cat vocalisations and body postures, then applies the same internal weights to every pet it encounters. However, in reality, different individual cats may have different vocalisations and body language in expressing their emotion. A general-purpose baseline model which is trained on a broad dataset is difficult in translating the cat's sound and body language in an accurate way.

Therefore, to address this problem, the app treats the baseline model not as a fixed final model but as a starting point. Every time the sound interpretation feature makes a prediction, the user has the option to confirm or correct it. These feedback responses are stored as labelled training records in Supabase multimodal_logs table, each tied to a specific pet_id. Over time, this accumulated feedback builds up into a small but highly targeted dataset that reflects exactly how that particular cat sounds and moves. On a bi-weekly schedule, the application uses the data to fine-tune a copy of the baseline model for a specific pet, gradually shifting its internal weights toward the specific acoustic and visual patterns of that individual pet. Once trained, the personalised model replaces the baseline for all future predictions for that pet and the process continues to improve with every new round of feedback.

5.3.3.2 Transfer Learning Implementation

Retrieve feedback, retrain model(freeze/unfreeze), get a new personalised model and replace baseline model

Retrieve Feedback Process

The fine-tuning process is run on a bi-weekly schedule. Before training begins for any pet, queries that retrieve all feedback records where is_correct is not null is processed. There will be a minimum threshold enforced, where a pet must have at least 5 total feedback entries with at least 2 corrections before fine-tuning is triggered. This guard prevents the model from being updated on small samples which could cause it to perform worse than the baseline model.

```

def run_biweekly_rl():
    print("=" * 60)
    print("Starting Bi-Weekly Supervised Fine-Tuning")
    print("=" * 60)

    response = (
        supabase.table("multimodal_logs")
        .select("*")
        .not_is("is_correct", "null")
        .execute()
    )
    logs = response.data

    if not logs:
        print("No feedback found. Users need to submit thumbs up/down first.")
        return

    correct_count = sum(1 for l in logs if l['is_correct'])
    incorrect_count = sum(1 for l in logs if not l['is_correct'])
    print(f"Total feedback entries : {len(logs)}")
    print(f" Confirmed correct      : {correct_count}")
    print(f" Corrections (wrong)       : {incorrect_count}")

    pet_data = {}
    for log in logs:
        pid = log['pet_id']
        pet_data.setdefault(pid, []).append(log)

    for pet_id, pet_logs in pet_data.items():
        corrections = [l for l in pet_logs if not l['is_correct']]
        if len(pet_logs) < 5 or len(corrections) < 2:
            print(f"\nSkipping {pet_id}: need >= 5 total feedback and >= 2 corrections "
                  f"(have {len(pet_logs)} total, {len(corrections)} corrections).")
            continue
        apply_rl_to_pet(pet_id, pet_logs)

```

Figure 5.3.37 Threshold for Retrieving Records of Fine-Tuning

Transfer Learning Strategy

Before training the model, a selective layer-freezing strategy is implemented in both sound and visual models. For the audio CNN-BiLSTM model, the first 5 layers, which cover the Input, Conv1D, Batch Normalisation, Max Pooling and Dropout, are frozen. The BiLSTM, Attention Mechanism, Dense and Output layer are kept trainable. This is because CNN front-end has already learned stable low-level features from the original training dataset and retraining them on a handful feedback samples will destroy those general patterns. On the other hand, the BiLSTM is the component that is used for recognising temporal vocal sequences and it needs to adapt to a specific pet's unique way of vocalising.

For the visual MobileNetV2 model, the freeze boundary is set at layer 120 inside the MobileNetV2 sub-model. Layers between 0 to 119 are the early convolutional blocks responsible for detecting edges and texture remain frozen, while the layers 120 onwards which include the higher-level feature blocks and the entire custom classification head are kept trainable. This approach preserves the general visual understanding while still giving the model enough capacity to adapt to the specific body language patterns of an individual cat.

```
# AUDIO: freeze only CNN front-end (layers[0:5]); LSTM + Dense must train
for i, layer in enumerate(audio_model.layers):
    layer.trainable = (i >= AUDIO_FREEZE_UNTIL)

# VISUAL: the model is Sequential[MobileNetV2, GAP, Dense, BN, Dropout, Dense]
# It only has 6 top-level layers so VISUAL_FREEZE_UNTIL=120 would freeze ALL.
# Must freeze INSIDE MobileNetV2 (layer[0]) and keep the custom head trainable.
for layer in visual_model.layers[1:]: # GAP, Dense, BN, Dropout, Dense – always train
    layer.trainable = True
mobilenet = visual_model.layers[0] # MobileNetV2 sub-model
mobilenet.trainable = True # must be True for internal flags to work
for i, layer in enumerate(mobilenet.layers):
    layer.trainable = (i >= VISUAL_FREEZE_UNTIL) # freeze early blocks only
```

Figure 5.3.38 Freezing Strategy

Optimiser Setting

There will be two separate Adam Optimisers used. One for the sound model at a learning rate of $5e-5$ and another for the visual model at much smaller $5e-6$. The reason the learning rate for the visual model is smaller than the sound model is because MobileNetV2 has over 2 million trainable parameters, applying the same learning rate as the smaller audio model will cause oscillation rather than smooth convergence on the limited feedback dataset.

```
audio_optimizer = keras.optimizers.Adam(learning_rate=0.00005)
visual_optimizer = keras.optimizers.Adam(learning_rate=0.000005)
loss_fn = keras.losses.SparseCategoricalCrossentropy()
```

Figure 5.3.39 Optimiser Initialisation

Training Personalised Model

The training loop prioritises learning from mistakes over reinforcing what the model predicts correctly. Correction records are trained with a sample weight of 1.5 and 3 gradient steps, while confirmation records use a weight of 1.0 and a single gradient step. Once all feedback records have been processed, all layers are unfrozen before saving to ensure the store weights capture the full model state. The updated audio weights are saved as {pet_id}_audio.h5 and the visual model as {pet_id}_visual.h5 in the models/pet directory. The new personalised will replace the baseline for all future predictions for that pet.

```

def apply_rl_to_pet(pet_id, logs):
    for log in sorted_logs:
        temp_video = "temp_rl_video.mp4"
        temp_audio = "temp_rl_audio.wav"

        try:
            download_video(log['video_url'], temp_video)

            from moviepy.video.io.VideoFileClip import VideoFileClip
            clip = VideoFileClip(temp_video)
            clip.audio.write_audiofile(temp_audio, logger=None)
            clip.close()

            audio_features = extract_audio_features(temp_audio)
            visual_features = extract_visual_features(temp_video, num_frames=3)

            if log['is_correct']:
                train_audio_label = log['audio_label']
                train_visual_label = log['visual_label']
                audio_steps = 1
                visual_steps = 1
                sample_weight = 1.0
                signal_type = "REINFORCE"
            else:
                train_audio_label = log.get('user_correct') or _flip_audio(log['audio_label'])
                train_visual_label = log.get('user_correct_visual') or _flip_visual(log['visual_label'])
                audio_steps = 3
                visual_steps = 1
                sample_weight = 1.5
                signal_type = "CORRECT"

            target_audio_idx = AUDIO_LABEL_MAP.get(train_audio_label, 0)
            for _ in range(audio_steps):
                with tf.GradientTape() as tape:
                    a_preds = audio_model(audio_features, training=False)
                    a_loss = loss_fn([target_audio_idx], a_preds) * sample_weight
                    a_grads = tape.gradient(a_loss, audio_model.trainable_variables)
                    audio_optimizer.apply_gradients(zip(a_grads, audio_model.trainable_variables))

            if visual_features is not None:
                target_visual_idx = VISUAL_LABEL_MAP.get(train_visual_label, 0)
                v_targets = [target_visual_idx] * len(visual_features)
                for _ in range(visual_steps):
                    with tf.GradientTape() as tape:
                        v_preds = visual_model(visual_features, training=False)
                        v_loss = loss_fn(v_targets, v_preds) * sample_weight
                        v_grads = tape.gradient(v_loss, visual_model.trainable_variables)
                        visual_optimizer.apply_gradients(zip(v_grads, visual_model.trainable_variables))

            print(f" [{signal_type}] audio_steps={audio_steps} visual_steps={visual_steps} "
                  f"weight={sample_weight} | "
                  f"Audio: {train_audio_label} | Visual: {train_visual_label}")

        except Exception as e:
            print(f" Failed on log {log.get('id', '?')}: {e}")

```

Figure 5.3.40 Training Personalised Model

5.3.3.3 Proof of Learning

Using 14 test video (7 happy, 7 angry), accuracy and confidence improvement, learning curve, loss convergence to verify that the transfer learning mechanism produces improvement rather than arbitrary weight changes, a test is conducted using a specific pet. A set of 14 labelled test videos (7 Happy samples, 7 Angry samples) is used to evaluate both baseline model and the personalised fine-tuning model on identical inputs.

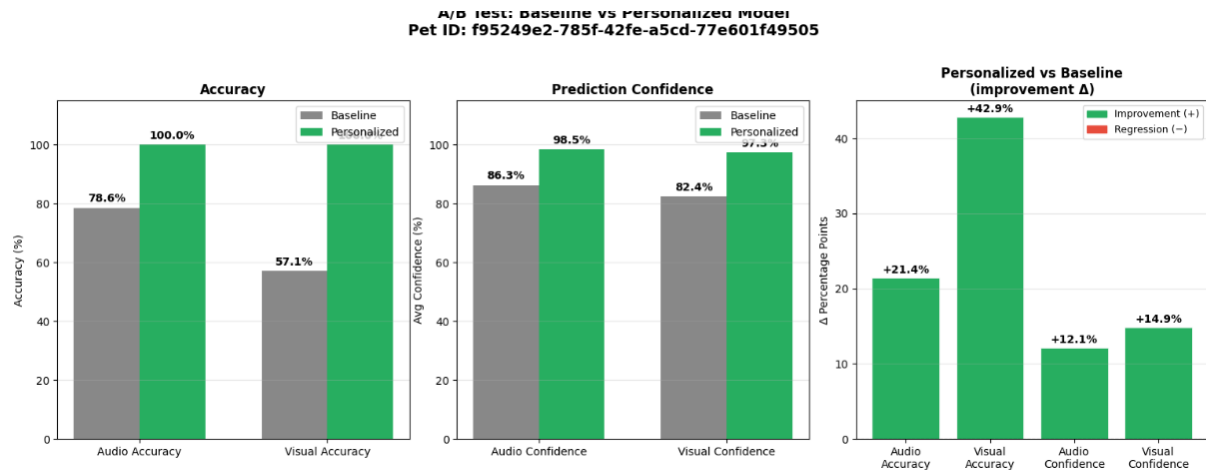


Figure 5.3.41 Accuracy and Confidence Improvement Comparison

As shown in the Figure 5.3.40, an accuracy comparison between baseline model and personalised model is built. The baseline audio model correctly predicted 78.6% of 14 test cases, while the personalised audio model achieved 100% accuracy after fine-tuning, a gain of 21.4%. Besides that, the visual model showed an even more dramatic improvement in which the baseline visual model only achieved 57.1% accuracy, while the personalised visual model also reached 100%, an improvement of 42.9%. This large baseline visual accuracy gap suggests that this particular cat's body language was quite different from the average patterns in the training dataset and the fine-tuning successfully bridged that gap.

Although a 100% accuracy may initially raise concerns regarding overfitting, it is appropriate within this application. For general tasks, perfect accuracy on a small dataset for a general task is an indication the model may have overfitted and may not generalise well to unseen data. However, the purpose of this app is not to generalise across a wide population of cats but to model the behaviour of a specific individual pet. Hence, the model is customised to learn to identify the distinctive vocal and body language patterns of an individual cat. Since the evaluation videos are also from the same cat, perfect accuracy implies the model has adapted very well to the pet. In short, the high accuracy reflects successful personalisation rather than overfitting, demonstrating that the app performs as intended.

Learning and Training Loss Curve

Figure 5.3.41 shows the accuracy learning curve across training epochs for both personalised audio and visual models. Both models start slightly above their respective baseline performance and improve steadily as training progresses. The personalised audio model quickly rises from about 82% to 100% accuracy, while the visual model increases from around 58% to 100%. By approximately epoch 3 to 4, both models converge and maintain stable perfect accuracy for the remaining epochs. This indicates that the models perform consistently after fine-tuning.

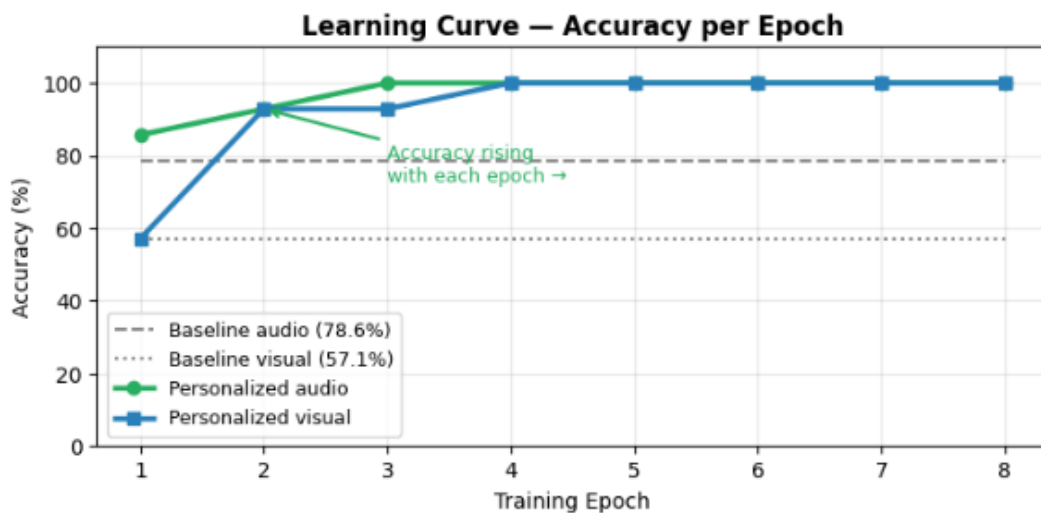


Figure 5.3.42 Learning Curve in Transfer Learning

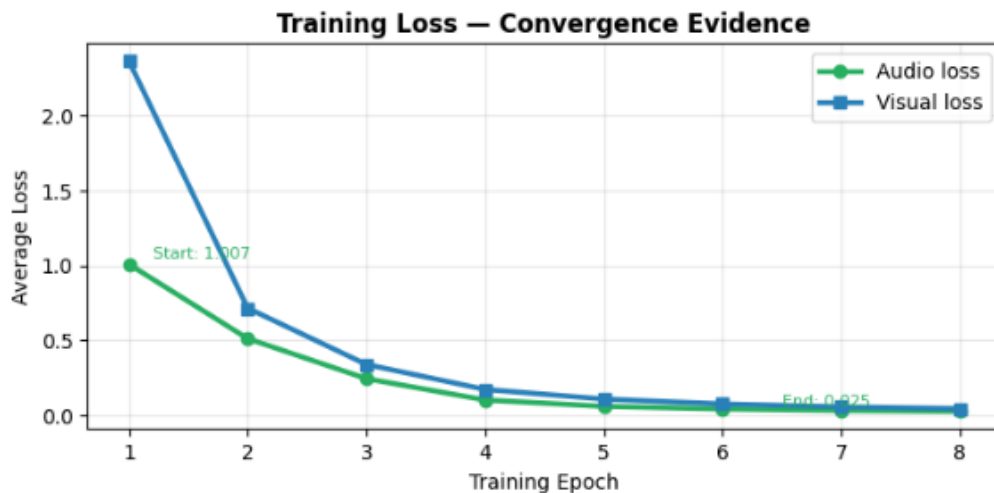


Figure 5.3.43 Training Loss in Transfer Learning

Furthermore, Figure 5.3.42 is a training loss curve that shows both models exhibit a smooth and continuous decrease in loss over time. The audio model reduces from 1.01 to 0.02. While

the visual model drops from about 2.4 to near 0. This shows that the model successfully minimizes error during training.

Together, these curves provide further evidence that the models are genuinely learning rather than memorising. The gradual improvement in accuracy and the smooth minimization in loss indicate stable gradient-based optimisation instead of random or unstable fitting. If the models are simply memorising data, the loss will fluctuate or abruptly without consistent improvement patterns. Hence, the stable convergence in both accuracy and loss prove that the models are learning meaningful patterns from the test data, adapting to the personalised pet behaviour.

5.4 Summary

This chapter focuses on the complete implementation of the Pet Care Application. It provides an in-depth walk-through of how each component of this application has been developed, configured and tested from beginning to end. In the setup section, explanation on how the frontend is developed using Flutter and Dart and backend is built with Python and fastAPI is provided. The use of Supabase for database, user authentication and file storage is also described. Ngrok is used as a temporary communication bridge between the mobile app and the local backend during development.

In the operation section, a development description of all the features, such as Food Nutrition Analysis, Health Analysis, Sound Interpretation, Calendar Reminder, Expense Tracker and Community Interaction, are provided. Each feature is designed to be as easy to use as possible for the user, while also being able to rely on a strong backend for logic. Furthermore, the technical part of this chapter focuses on the two AI models that are developed for the sound interpretation feature of the application. The models included the use of a CNN-BiLSTM for audio classification and a MobileNetV2 for visual recognition of body language. Each model was trained and validated using a process that included preparation of the dataset, augmentation of the dataset and staged training processes. The accuracy of results between the two models were 89% and 87%, respectively. Some evaluation results are also written in this chapter to prove that the models work well.

CHAPTER 6: Conclusion and Recommendation

6.1 Project Review, Discussion and Conclusion

This project creates a mobile device application that assists owners in being able to track and successfully manage their cat's welfare using artificial intelligence. The app is created using Flutter and contains multiple intelligence features that create an easy way for cat owners to understand how to care for their pet's wellbeing.

The main component of this pet care application is the sound interpretation component, which uses a two-model process consisting of a custom-designed CNN-BiLSTM model to classify sound and a MobileNetV2 model to recognise visual body language. During the project, both models were trained, assessed and improved. Ultimately, the CNN-BiLSTM model and the MobileNetV2 model achieved an 89% and an 87% testing accuracy. More importantly, the personalisation model built on top of these models demonstrated that fine-tuning based on individual user feedback is able to push accuracy to 100% for a specific pet. This confirms that the model genuinely adapts over time rather than remaining static. In addition to machine learning models, several features are also offered by the application which include food nutrition and health analysis, calendar reminders for pet-related activities, expense tracker, a community forum and a price comparison feature with price dropping notifications. Each feature has been developed to address actual user needs and are supported by FastAPI backend and utilise Supabase for data storage and authentication.

Overall, the project objectives, which simplify the analysis process, improve app's personalisation and develop an all-in-one integrated smart pet care app, are successfully achieved. This app demonstrates that it is possible to build a meaningful and AI-driven pet care application within the constraints of limited resources.

6.2 Novelties and Contribution

This project introduces many innovative ideas which help the application stand out from other existing pet care applications, especially due to its unique dual-model multimodal approach to sound interpretation. The application applies both audio and body language into one final prediction by using a common confidence-weighted strategy. Since this dual-model approach is not currently typical amongst consumer-oriented pet care applications, it provides a robust solution than single-modelling applications particularly where audio and visual signals may be ambiguous or contradictory.

Additionally, a pet-centric personalisation sound translator is a worthy deliverable of this project. Through user feedback and various fine-tuning based on the baseline model, the app can customise each user's experience and adjust to an individual pet's habits. This approach also offers continuous learning to improve the app's prediction accuracy over time without the user needing to know any technical details, which is totally different from other apps.

Lastly, the inclusion of AI-powered analysis and real-world utility features like expense tracking, community engagement and automated price alerts makes it actually useful rather than just a bunch of AI-generated predictions. With all features integrated into one mobile app, these elements as a whole significantly boost the potential for bringing AI closer to everyday pet ownership in a more meaningful and practical manner.

6.3 Future Work

Although this pet care app is made to work within its intended scope, several improvements can be made in the future. From the AI side, the most impactful improvement is expanding the sound classification model to have more categories beyond its current two class structure. This is because adding emotion categories such as fearful or painful will make the interpreter significantly more useful in real world scenarios. Besides, training models on a larger and diverse dataset also helps to strengthen the generalisation ability. The extending support to other animals such as dogs can also broaden the application's usage. Furthermore, from the systems side, transitioning from ngrok to a higher-level cloud-based backend will be the most important step towards deploying this app into a production environment. This eliminates network instability, improves reliability and allows the application to grow beyond local testing limits. Lastly, optimising the video or image processing pipeline to produce lower latency from the inference step by potentially using GPU-accelerated cloud-based inference. In short, the app can transform from a functional prototype into a more scalable and accurately deployable application through these future improvements. By enhancing both AI model capabilities and the backend infrastructure, the app will provide more reliable performance, support diverse use cases and provide a smoother user experience for pet users within their daily lives.

REFERENCES

- [1] A. M. Beck, S. Barker, N. R. Gee, J. A. Griffin, and R. Johnson, “The Background to Human-Animal Interaction (HAI) Research,” *Human-animal interaction bulletin*, vol. 2018, Dec. 2018, doi: <https://doi.org/10.1079/hai.2018.0015>.
- [2] H. L. Brooks *et al.*, “The power of support from companion animals for people living with mental health problems: A systematic review and narrative synthesis of the evidence,” *BMC Psychiatry*, vol. 18, no. 1, Feb. 2018, doi: <https://doi.org/10.1186/s12888-018-1613-2>.
- [3] F. Del Baldo *et al.*, “Accuracy of a flash glucose monitoring system in cats and determination of the time lag between blood glucose and interstitial glucose concentrations,” *Journal of Veterinary Internal Medicine*, vol. 35, no. 3, pp. 1279–1287, May 2021, doi: <https://doi.org/10.1111/jvim.16122>.
- [4] “The Impact of Pet Sitting on Community Building | Scout,” *Scoutforpets.com*, 2023. <https://www.scoutforpets.com/post/the-impact-of-pet-sitting-on-community-building> (accessed Apr. 24, 2025).
- [5] Md. Tauseef, E. Rathod, S. M. Nandish, and M. G. Kushal, “Advancements in Pet Care Technology: A Comprehensive Survey,” *Advancements in Pet Care Technology: A Comprehensive Survey*, Mar. 2024, doi: <https://doi.org/10.1109/icdecs59733.2023.10503555>.
- [6] P. Pongrácz, C. Molnár, and Á. Miklósi, “Barking in family dogs: An ethological approach,” *The Veterinary Journal*, vol. 183, no. 2, pp. 141–147, Feb. 2010, doi: <https://doi.org/10.1016/j.tvjl.2008.12.010>.
- [7] object Object, “Classification of dog barks: a machine learning approach,” *Core.ac.uk*, 2018, doi: oai:infoscience.tind.io:115481.
https://core.ac.uk/reader/147934074?utm_source=linkout
- [8] S. A. Alowais *et al.*, “Revolutionizing healthcare: The role of artificial intelligence in clinical practice,” *BMC Medical Education*, vol. 23, no. 1, Sep. 2023, doi: <https://doi.org/10.1186/s12909-023-04698-z>.
- [9] Grand View Research, “Pet Care Market Size, Research & Trends | Industry Report, 2018-2025,” *Grand View Research*, 2018. <https://www.grandviewresearch.com/industry-analysis/pet-care-market>

REFERENCES

- [10] “Pawdi - Pet Food Scanner,” *Getpawdi.com*, 2024. <https://www.getpawdi.com/>
- [11] “Pet Food Expert,” *Petfoodexpert.com*, 2025. <https://petfoodexpert.com/?search=royal%20canin&species=dog> (accessed Apr. 24, 2025).
- [12] Ronald-Vlad Creanga, “Pawdi - Pet Food Scanner,” *App Store*, Dec. 19, 2024. <https://apps.apple.com/ro/app/pawdi-pet-food-scanner/id6738991905>
- [13] “Choosing the right food for your dog or cat | Animal Humane Society,” *www.animalhumanesociety.org*. <https://www.animalhumanesociety.org/resource/choosing-right-food-your-dog-or-cat>
- [14] “TTcare, a health care app for pets,” *www.ttcareforpet.com*. <https://www.ttcareforpet.com/>
- [15] “South Korea’s drive to revolutionize pet tech,” *GlobalPETS*, Feb. 26, 2024. <https://globalpetindustry.com/article/south-koreas-drive-to-revolutionize-pet-tech/> (accessed Apr. 24, 2025).
- [16] “Our Solution - World’s first AI-Powered Pet Health Scan App,” *AI For Pets*, 2024. <https://www.aiforpet.com/our-solution/>
- [17] “Early Diagnosis - an overview | ScienceDirect Topics,” *www.sciencedirect.com*. <https://www.sciencedirect.com/topics/nursing-and-health-professions/early-diagnosis>
- [18] I. Belcic and C. Stryker, “Supervised Learning,” *Ibm.com*, Sep. 23, 2021. <https://www.ibm.com/think/topics/supervised-learning>
- [19] J. Stihec, “Self-Supervised Learning Harnesses the Power of Unlabeled Data,” *Shelf*, Jul. 02, 2024. <https://shelf.io/blog/self-supervised-learning-harnesses-the-power-of-unlabeled-data/>
- [20] H. Weaver, “This Free AI App Can Tell You What Your Cat’s Meows Mean—Supposedly,” *Kinship.com*, Jan. 22, 2024. <https://www.kinship.com/cat-behavior/meow-talk-app-news>
- [21] L. A. Ntalampiras *et al.*, “Automatic Classification of Cat Vocalizations Emitted in Different Contexts,” *Animals*, vol. 9, no. 8, p. 543, Aug. 2019, doi: <https://doi.org/10.3390/ani9080543>.

REFERENCES

- [22] C. Arnold, “What is your cat telling you? New technology deciphers meows,” *Animals*, Oct. 23, 2024. <https://www.nationalgeographic.com/animals/article/cats-pets-communication-app-technology>
- [23] “How to train the MeowTalk app for your cat,” *MeowTalk*, 2025. <https://www.meowtalk.app/tutorials>
- [24] Desert Veterinary Clinic, “Desert Veterinary Clinic,” *Desert Veterinary Clinic*, Apr. 17, 2024. <https://desertvet.com/unnoticed-hazards-the-essential-role-of-fecal-analysis-in-pet-health-care/>
- [25] E. Kasırğa, “The importance of stool tests in diagnosis and follow-up of gastrointestinal disorders in children,” *Türk Pediatri Arşivi*, vol. 54, no. 3, 2019, doi: <https://doi.org/10.14744/turkpediatriars.2018.00483>.
- [26] Kamal Karkonasasi, Cheah, Y.-N., Mogana Vadiveloo, & Seyed Aliakbar Mousavi. (2023). Acceptance of a Text Messaging Vaccination Reminder and Recall System in Malaysia’s Healthcare Sector: Extending the Technology Acceptance Model. *Vaccines*, 11(8), 1331–1331. <https://doi.org/10.3390/vaccines11081331>
- [27] Pet, U. (2024, September 5). *VetVerifi*. VetVerifi. [https://www.vetverifi.com/blog/pet-vaccination-reminders-vet-clinic-notifications?utm_source](https://www.vetverifi.com/blog/pet-vaccination-reminders-vet-clinic-notifications?utm_source=feature)
- [28] *feature*. (n.d.). 11pets. <https://www.11pets.com/en/feature>
- [29] S.S., P., Menon, V., & Gopalan, S. (2024). Hybrid CNN-BiLSTM architecture with multiple attention mechanisms to enhance speech emotion recognition. *Biomedical Signal Processing and Control*, 106967. <https://doi.org/10.1016/j.bspc.2024.106967>
- [30] Karim Mohammed Rezaul, Jewel, M., Islam, S., Noor, K., & Md Abrar Hamim. (2024, July 7). *Enhancing Audio Classification Through MFCC Feature Extraction and Data Augmentation with CNN and RNN Models*. <https://doi.org/10.13140/RG.2.2.22497.93280>
- [31] Li, H., Li, X., Liu, J., Liu, Z., Zeng, X., & Wang, H. (2025). Infrasound signal classification based on attention-CNN-BiLSTM. *Journal of Low Frequency Noise, Vibration and Active Control*. <https://doi.org/10.1177/14613484251399954>

REFERENCES

- [32] Graves, A., & Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks*, 18(5-6), 602–610. <https://doi.org/10.1016/j.neunet.2005.06.042>
- [33] Yosinski, J., Clune, J., Bengio, Y., & Lipson, H. (2014). *How transferable are features in deep neural networks?* <https://arxiv.org/pdf/1411.1792>
- [34] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. (2019). *MobileNetV2: Inverted Residuals and Linear Bottlenecks*. <https://arxiv.org/pdf/1801.04381>
- [34] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. (2019). *MobileNetV2: Inverted Residuals and Linear Bottlenecks*. <https://arxiv.org/pdf/1801.04381>
- [35] Lin, M., Chen, Q., & Yan, S. (2014). *Network In Network*. <https://arxiv.org/pdf/1312.4400>
- [36] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R., & Bengio, Y. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15, 1929–1958. <https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf>
- [37] Ioffe, S. (2015). *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. <https://arxiv.org/pdf/1502.03167>
- [38] Yosinski, J., Clune, J., Bengio, Y., & Lipson, H. (2014). *How transferable are features in deep neural networks?* <https://arxiv.org/pdf/1411.1792>
- [39] Pandeya, Y. R., Kim, D., & Lee, J. (2018). Domestic Cat Sound Classification Using Learned Features from Deep Neural Nets. *Applied Sciences*, 8(10), 1949. <https://doi.org/10.3390/app8101949>
- [40] Fawzy, A., Hassan, A., Gad, A., Zaky, A. B., & El-Hussieny, H. (2023). DeepCat: A Deep Learning Approach to Understand Your Cat's Body Language. *2023 11th International Japan-Africa Conference on Electronics, Communications, and Computations (JAC-ECC)*, 207–211. <https://doi.org/10.1109/jac-ecc61002.2023.10479630>

