



Faculty of Computer Science and Information Technology

***A WEB APPLICATION CHATBOT FOR NATURE AND  
WILDLIFE SPECIES IDENTIFICATION USING PALIGEMMA  
AS THE VISION-LANGUAGE MODEL***

Audry Mayla Anak Meeta

Bachelor of Software Engineering with Honours

2025

**A WEB APPLICATION CHATBOT FOR NATURE AND WILDLIFE SPECIES  
IDENTIFICATION USING PALIGEMMA AS THE VISION-LANGUAGE MODEL**

AUDRY MAYLA ANAK MEETA

This project is submitted to in partial fulfilment of the requirements for the  
degree of Bachelor of Software Engineering with Honours

Faculty of Computer Science and Information Technology

UNIVERSITI MALAYSIA SARAWAK

2025

UNIVERSITI MALAYSIA SARAWAK

THESIS STATUS ENDORSEMENT FORM

**TITLE** A Web Application Chatbot for Nature and Wildlife Species Identification Using  
PaliGemma as the Vision-Language Model

**ACADEMIC SESSION:** 2024/2025

Audry Mayla Anak Meeta

hereby agree that this Thesis\* shall be kept at the Centre for Academic Information Services, Universiti Malaysia Sarawak, subject to the following terms and conditions:

1. The Thesis is solely owned by Universiti Malaysia Sarawak
2. The Centre for Academic Information Services is given full rights to produce copies for educational purposes only
3. The Centre for Academic Information Services is given full rights to do digitization in order to develop local content database
4. The Centre for Academic Information Services is given full rights to produce copies of this Thesis as part of its exchange item program between Higher Learning Institutions [ or for the purpose of interlibrary loan between HLI ]
5. \*\* Please tick ( ✓ )

|                                     |                     |                                                                                                            |
|-------------------------------------|---------------------|------------------------------------------------------------------------------------------------------------|
| <input type="checkbox"/>            | <b>CONFIDENTIAL</b> | (Contains classified information bounded by the OFFICIAL SECRETS ACT 1972)                                 |
| <input type="checkbox"/>            | <b>RESTRICTED</b>   | (Contains restricted information as dictated by the body or organization where the research was conducted) |
| <input checked="" type="checkbox"/> | <b>UNRESTRICTED</b> |                                                                                                            |



(AUTHOR'S SIGNATURE)

Validated by



(SUPERVISOR'S SIGNATURE)

Permanent Address

No 173, Lot 817, Lorong 8, RPR Batu  
Kawa Fasa 1, 93250, Kuching,  
Sarawak.

Date: 21 July 2025

Date: 21 July 2025

Note \* Thesis refers to PhD, Master, and Bachelor Degree

\*\* For Confidential or Restricted materials, please attach relevant documents from relevant organizations / authorities

## **Acknowledgement**

I would like to express gratitude towards those who have supported me throughout my Final Year Project.

First of all, I am sincerely thankful to my supervisor, Dr Sarah Flora Anak Samson Juan for her helpful guidance, inspiring mentorship and support. Her expertise and constructive feedback have been essential in shaping the direction of this project.

I would like to also extend my gratitude to my examiner for their evaluation and feedback which has helped me improve my work significantly.

A special thanks to my family, for I am grateful for their endless encouragement, unwavering support and understanding throughout this journey.

Finally, I would like to thank all the respondents who participated in the survey. Their time and input have been crucial in providing valuable feedback for this project.

Thank you to everyone who supported and contributed to my Final Year Project.

## Table of Contents

|                                                        |      |
|--------------------------------------------------------|------|
| Acknowledgement.....                                   | i    |
| Table of Contents .....                                | ii   |
| List of Figures .....                                  | vii  |
| List of Tables .....                                   | xiii |
| Abstract .....                                         | xiv  |
| Abstrak .....                                          | xv   |
| CHAPTER 1: INTRODUCTION .....                          | 1    |
| 1.1 Introduction .....                                 | 1    |
| 1.2 Problem Statement .....                            | 2    |
| 1.3 Objectives.....                                    | 2    |
| 1.4 Methodology .....                                  | 3    |
| 1.4.1 Phase 1: Requirement Definition.....             | 4    |
| 1.4.2 Phase 2: System and Software Design .....        | 4    |
| 1.4.3 Phase 3: Implementation and Unit Testing.....    | 4    |
| 1.4.4 Phase 4: Integration and System Testing .....    | 4    |
| 1.4.5 Phase 5: Operation and Maintenance .....         | 5    |
| 1.5 Project Scope.....                                 | 5    |
| 1.6 Significance of Project .....                      | 5    |
| 1.7 Project Schedule.....                              | 6    |
| 1.8 Expected Outcome .....                             | 7    |
| 1.9 Project Report Outline.....                        | 8    |
| 1.9.1 Chapter 1: Introduction.....                     | 8    |
| 1.9.2 Chapter 2: Literature Review .....               | 8    |
| 1.9.3 Chapter 3: Requirement, Analysis and Design..... | 9    |
| 1.9.4 Chapter 4: Implementation .....                  | 9    |

|                                                                                                             |    |
|-------------------------------------------------------------------------------------------------------------|----|
| 1.9.5 Chapter 5: Testing.....                                                                               | 9  |
| 1.9.6 Chapter 6: Conclusion and Future Work .....                                                           | 9  |
| 1.10 Summary .....                                                                                          | 10 |
| CHAPTER 2: LITERATURE REVIEW .....                                                                          | 11 |
| 2.1 Introduction .....                                                                                      | 11 |
| 2.2 Review on PaliGemma, a Vision-Language Model .....                                                      | 11 |
| 2.2.1 Introduction of PaliGemma and PaliGemma’s Architecture Review .....                                   | 11 |
| 2.2.2 PaliGemma’s Pretraining Stages .....                                                                  | 13 |
| 2.2.3 “Inference PaliGemma with Keras” – An Existing Project Utilizing PaliGemma<br>from Kaggle Review..... | 14 |
| 2.2.4 PaliGemma Demo Review - HuggingFace .....                                                             | 16 |
| 2.2.5 PaliGemma Article by Google Review .....                                                              | 24 |
| 2.2.6 Finetunability of PaliGemma.....                                                                      | 24 |
| 2.3 Review on PaliGemma 2, An Upgrade of PaliGemma .....                                                    | 25 |
| 2.3.1 PaliGemma 2 Architecture and Pretraining Review .....                                                 | 25 |
| 2.3.2 PaliGemma 2 Fine-tuning Tasks.....                                                                    | 27 |
| 2.4 Review existing systems .....                                                                           | 30 |
| 2.4.1 Google Gemini .....                                                                                   | 31 |
| 2.4.2 ChatGPT .....                                                                                         | 34 |
| 2.4.3 DALL-E 3 .....                                                                                        | 37 |
| 2.4.4 Comparison between the Existing Systems.....                                                          | 45 |
| 2.5 Review on Tools and Technology for Web Applications .....                                               | 48 |
| 2.5.1 Front-end Development.....                                                                            | 48 |
| 2.5.2 Back-end Development .....                                                                            | 48 |
| 2.5.3 API Integration Tool (PaliGemma).....                                                                 | 48 |
| 2.5.4 Database Integration .....                                                                            | 49 |

|                                                      |    |
|------------------------------------------------------|----|
| 2.6 Proposed System .....                            | 49 |
| 2.7 Summary .....                                    | 50 |
| CHAPTER 3: REQUIREMENTS ANALYSIS AND DESIGN .....    | 51 |
| 3.1 Introduction .....                               | 51 |
| 3.2 Methodology summary .....                        | 51 |
| 3.3 System requirements .....                        | 53 |
| 3.3.1 Software requirements .....                    | 53 |
| 3.3.2 Hardware requirements.....                     | 55 |
| 3.4 Requirement Elicitation & Analysis.....          | 56 |
| 3.4.1 Questionnaire Response Analysis.....           | 57 |
| 3.4.2 System Features .....                          | 66 |
| 3.5 System design.....                               | 67 |
| 3.5.1 Use Case Diagram .....                         | 67 |
| 3.5.2 Sequence Diagram.....                          | 69 |
| 3.5.3 Class Diagram.....                             | 75 |
| 3.6 Wireframe Design of Interface .....              | 77 |
| 3.6.1 Unregistered and Non-Logged In User Flow ..... | 79 |
| 3.6.2 Registered and Logged In User Flow .....       | 83 |
| 3.6.3 Additional features.....                       | 86 |
| 3.7 Database Component.....                          | 86 |
| 3.8 Summary .....                                    | 88 |
| CHAPTER 4: IMPLEMENTATION AND TESTING .....          | 89 |
| 4.1 Introduction .....                               | 89 |
| 4.2 Data Collection Implementation .....             | 89 |
| 4.2.1 Dataset Preparation.....                       | 90 |
| 4.2.2 Dataset description .....                      | 93 |

|                                                                |     |
|----------------------------------------------------------------|-----|
| 4.2.3 CSV file conversion into JSONL file .....                | 93  |
| 4.3 Development Environment Setup .....                        | 104 |
| 4.2.1 Visual Studio Code .....                                 | 104 |
| 4.2.2 Django .....                                             | 105 |
| 4.2.3 MySQL .....                                              | 108 |
| 4.2.4 Google Colab.....                                        | 111 |
| 4.2.5 HuggingFace.....                                         | 112 |
| 4.4 Deployment of Finetuned PaliGemma Model .....              | 112 |
| 4.4.1 Saving Finetuned PaliGemma Model into Google Colab ..... | 112 |
| 4.4.2 Upload Finetuned Model to HuggingFace .....              | 113 |
| 4.5 Web Application Deployment .....                           | 115 |
| 4.5.1 Render.....                                              | 115 |
| 4.5.2 MySQL .....                                              | 118 |
| 4.5.3 HuggingFace Inference Endpoint.....                      | 119 |
| 4.5.4 Web Application Interface .....                          | 120 |
| 4.7 Chapter Summary.....                                       | 131 |
| CHAPTER 5: TESTING.....                                        | 133 |
| 5.1 Introduction .....                                         | 133 |
| 5.2 Functional Testing .....                                   | 133 |
| 5.2.1 Unit Testing .....                                       | 133 |
| 5.2.2 Integration Testing .....                                | 143 |
| 5.3 Non-Functional Testing.....                                | 147 |
| 5.3.1 Usability Testing.....                                   | 147 |
| CHAPTER 6: CONCLUSION AND FUTURE WORK .....                    | 154 |
| 6.1 Introduction .....                                         | 154 |
| 6.2 Achievement of Objectives .....                            | 154 |

|                       |     |
|-----------------------|-----|
| 6.3 Limitations ..... | 155 |
| 6.4 Future Work.....  | 156 |
| 6.5 Conclusion.....   | 157 |
| References .....      | 158 |
| Appendices.....       | 161 |

## List of Figures

|                                                                                                               |    |
|---------------------------------------------------------------------------------------------------------------|----|
| Figure 1.1: Modified Waterfall Model with Iteration between phases. ....                                      | 3  |
| Figure 1.2: Project Schedule Gantt Chart. ....                                                                | 6  |
| Figure 2.1: PaliGemma’s architecture visual overview. ....                                                    | 12 |
| Figure 2.2: PaliGemma demo with prompt 1a) and answer 1a). ....                                               | 17 |
| Figure 2.3: PaliGemma demo with prompt 1b) and answer 1b). ....                                               | 17 |
| Figure 2.4: PaliGemma demo with prompt 1c) and answer 1c). ....                                               | 17 |
| Figure 2.5: PaliGemma demo with prompt 2a) and answer 2a). ....                                               | 19 |
| Figure 2.6: PaliGemma demo with prompt 2b) and answer 2b). ....                                               | 19 |
| Figure 2.7: PaliGemma demo with prompt 2c) and answer 2c). ....                                               | 19 |
| Figure 2.8: PaliGemma demo with prompt 3a) and answer 3a). ....                                               | 21 |
| Figure 2.9: PaliGemma demo with prompt 3b) and answer 3b). ....                                               | 21 |
| Figure 2.10: PaliGemma demo with prompt 3c) and answer 3c). ....                                              | 21 |
| Figure 2.11: PaliGemma demo with prompt 1a) and answer 1a). ....                                              | 23 |
| Figure 2.12: PaliGemma demo with prompt 1b) and answer 1b). ....                                              | 23 |
| Figure 2.13: PaliGemma demo with prompt 1c) and answer 1c). ....                                              | 23 |
| Figure 2.14: Preview of Google Gemini’s user interface. ....                                                  | 32 |
| Figure 2.15: Pop-up box of sharing module when clicking share icon. ....                                      | 33 |
| Figure 2.16: ChatGPT’s user interface. ....                                                                   | 35 |
| Figure 2.17: OpenArt website home page with limited access to DALL-E 3 ....                                   | 38 |
| Figure 2.18: Image to Prompt UI. ....                                                                         | 39 |
| Figure 2.19: Image to Prompt (text generation). ....                                                          | 40 |
| Figure 2.20: Image generation from DALL-E 3 model. ....                                                       | 41 |
| Figure 2.21: Model switch selection for DALL-E 3. ....                                                        | 42 |
| Figure 2.22: Models fine-tune section for DALL-E 3. ....                                                      | 42 |
| Figure 2.23: Creation history for DALL-E 3. ....                                                              | 43 |
| Figure 2.24: Sharing module for DALL-E 3. ....                                                                | 44 |
| Figure 2.25: User feedback module for DALL-E 3. ....                                                          | 44 |
| Figure 3.1: Modified Waterfall Model with Iteration between phases ....                                       | 52 |
| Figure 3.2: Are you familiar with AI chatbot such as ChatGPT, Google Gemini and other<br>similar system?..... | 57 |
| Figure 3.3: Have you used the services of AI chatbot?.....                                                    | 57 |

|                                                                                                               |    |
|---------------------------------------------------------------------------------------------------------------|----|
| Figure 3.4: How often do you use AI chatbot? From 1 (seldom) to 5 (very often).                               | 58 |
| Figure 3.5: What do you usually use the AI chatbot for?                                                       | 59 |
| Figure 3.6: Would you consider using AI chat for the species identification of nature and wildlife?           | 59 |
| Figure 3.7: Are you familiar with the AI model used within these chatbots? (E.g. GPT4o-mini, Gemma)           | 60 |
| Figure 3.8: Do you know of PaliGemma as a vision-language model?                                              | 61 |
| Figure 3.9: Do you know of any system that is able to describe image after prompting? (image captioning)      | 61 |
| Figure 3.10: If your answer for the question above is Yes, state the name of the app/website                  | 62 |
| Figure 3.11: Would you be interested to know how PaliGemma works to describe an image?                        | 63 |
| Figure 3.12: If there is a chance to build such system, what features would you expect/prefer?                | 63 |
| Figure 3.13: What other features would you suggest for this web application chatbot? (including answers)      | 64 |
| Figure 3.14: Additional answers for ‘What other features would you suggest for this web application chatbot?’ | 65 |
| Figure 3.15: Additional answers for ‘What other features would you suggest for this web application chatbot?’ | 65 |
| Figure 3.16: Use case diagram.                                                                                | 67 |
| Figure 3.17: PaliGemma processes when user is not logged in.                                                  | 69 |
| Figure 3.18: PaliGemma processes when user is logged in.                                                      | 71 |
| Figure 3.19: Web Application processes when user signs up.                                                    | 73 |
| Figure 3.20: Class diagram.                                                                                   | 75 |
| Figure 3.21: Sign-Up Page.                                                                                    | 78 |
| Figure 3.22: Login Page.                                                                                      | 79 |
| Figure 3.23: Main page for Non-Logged In User.                                                                | 80 |
| Figure 3.24: Image Upload and Prompt Sample for Non-Logged In User.                                           | 80 |
| Figure 3.25: Chatbot Response Sample for Non-Logged In User.                                                  | 81 |
| Figure 3.26: A Pop-Up of Overlay Prompting User to Save Current Chat.                                         | 82 |
| Figure 3.27: User Choose ‘Yes’ in Figure 3.26.                                                                | 82 |

|                                                                                                                      |     |
|----------------------------------------------------------------------------------------------------------------------|-----|
| Figure 3.28: User Choose ‘No’ from Figure 3.26. ....                                                                 | 83  |
| Figure 3.29: Main Page for Logged In User. ....                                                                      | 84  |
| Figure 3.30: Image Upload and Prompt Sample for Logged In User. ....                                                 | 84  |
| Figure 3.31: Chatbot Response Sample for Logged In User. ....                                                        | 85  |
| Figure 3.32: Sidebar menu. ....                                                                                      | 86  |
| Figure 3.33: Entity Relationship Diagram. ....                                                                       | 87  |
| Figure 4.1: iNaturalist website interface. ....                                                                      | 90  |
| Figure 4.2: Step 1 create a query. ....                                                                              | 91  |
| Figure 4.3: Step 2 preview. ....                                                                                     | 92  |
| Figure 4.4: Step 3 choose column. ....                                                                               | 92  |
| Figure 4.5: Step 4 create export. ....                                                                               | 93  |
| Figure 4.6: Convert CSV to JSON Python script. ....                                                                  | 94  |
| Figure 4.7: The converted CSV file to JSON file output. ....                                                         | 95  |
| Figure 4.8: Convert JSON to JSONL Python script. ....                                                                | 95  |
| Figure 4.9: The converted JSON to JSONL file output. ....                                                            | 96  |
| Figure 4.10: A Python script to add prefix into existing JSONL file. ....                                            | 97  |
| Figure 4.11: Added prefix into JSONL file. ....                                                                      | 98  |
| Figure 4.12: A Python script to combine suffix. ....                                                                 | 99  |
| Figure 4.13: Combined suffix (scientific_name & common_name). ....                                                   | 99  |
| Figure 4.14: A Python script to reorder fields in JSONL file according to PaliGemma JSONL<br>annotation format. .... | 100 |
| Figure 4.15: The output file of the reordered JSONL fields. ....                                                     | 101 |
| Figure 4.16: A Python script to convert image URLs into ascending numbers. ....                                      | 102 |
| Figure 4.17: The output file of converted image URLs. ....                                                           | 102 |
| Figure 4.18: A Python script to add .jpg file extension into the image field. ....                                   | 103 |
| Figure 4.19: The output file of the added .jpg file extension in the image field. ....                               | 104 |
| Figure 4.20: Django project folder. ....                                                                             | 105 |
| Figure 4.21: Django app folder. ....                                                                                 | 106 |
| Figure 4.22: Application definition. ....                                                                            | 106 |
| Figure 4.23: Template directory configuration. ....                                                                  | 107 |
| Figure 4.24: Database configuration for Django. ....                                                                 | 109 |
| Figure 4.25: Model creation for paligemma database. ....                                                             | 110 |

|                                                                                |     |
|--------------------------------------------------------------------------------|-----|
| Figure 4.26: Fine-tuning process. ....                                         | 111 |
| Figure 4.27: Creating a directory for model download.....                      | 112 |
| Figure 4.28: Saving the fine-tuned parameters of the model. ....               | 113 |
| Figure 4.29: Install huggingface_hub library.....                              | 113 |
| Figure 4.30: Initialize login into HuggingFace.....                            | 113 |
| Figure 4.31: Upload downloaded fine-tuned model to HuggingFace Model hub. .... | 114 |
| Figure 4.32: GitHub repository structure.....                                  | 116 |
| Figure 4.33: Procfile content.....                                             | 116 |
| Figure 4.34: Contents of ‘requirements.txt’ .....                              | 117 |
| Figure 4.35: Build commands for Render.....                                    | 118 |
| Figure 4.36: HuggingFace Inference Endpoint dashboard. ....                    | 119 |
| Figure 4.37: Main page. ....                                                   | 120 |
| Figure 4.38: Image preview after upload. ....                                  | 121 |
| Figure 4.39: Response page. ....                                               | 121 |
| Figure 4.40: First pop-up overlay.....                                         | 122 |
| Figure 4.41: “Yes” option pop-up overlay. ....                                 | 123 |
| Figure 4.42: “No” option pop-up overlay. ....                                  | 123 |
| Figure 4.43: Login page.....                                                   | 124 |
| Figure 4.44: Register page. ....                                               | 125 |
| Figure 4.45: Redirect to login page.....                                       | 125 |
| Figure 4.46: Home page.....                                                    | 126 |
| Figure 4.47: Image preview. ....                                               | 127 |
| Figure 4.48: Response page. ....                                               | 127 |
| Figure 4.49: Feedback validation.....                                          | 128 |
| Figure 4.50: Chat history page.....                                            | 129 |
| Figure 4.51: Thumbs up feedback filter. ....                                   | 130 |
| Figure 4.52: Thumbs down feedback filter. ....                                 | 130 |
| Figure 4.53: Viewing prompt and response history. ....                         | 131 |
| Figure 5.1: What did you like most about the system? .....                     | 148 |
| Figure 5.2: What problems did you encounter, if any? .....                     | 149 |
| Figure 5.3: What suggestions do you have for improvement? .....                | 150 |
| Figure 5.4: Images for testing. ....                                           | 151 |

|                                                                                                      |     |
|------------------------------------------------------------------------------------------------------|-----|
| Figure 5.5: Formula to calculate PaliGemma’s accuracy.....                                           | 152 |
| Figure 7.1: Question 1 of Section 1. ....                                                            | 161 |
| Figure 7.2: Question 2 of Section 1. ....                                                            | 161 |
| Figure 7.3: Question 3 of Section 1. ....                                                            | 161 |
| Figure 7.4: Question 4 of Section 1. ....                                                            | 162 |
| Figure 7.5: Question 1 of Section 2. ....                                                            | 162 |
| Figure 7.6: Question 2 of Section 2. ....                                                            | 162 |
| Figure 7.7: Question 3 of Section 2. ....                                                            | 163 |
| Figure 7.8: Question 4 of Section 2. ....                                                            | 163 |
| Figure 7.9: Question 5 of Section 2. ....                                                            | 163 |
| Figure 7.10: Question 1 of Section 3. ....                                                           | 163 |
| Figure 7.11: Question 2 of Section 3. ....                                                           | 164 |
| Figure 7.12: Question 3 of Section 3. ....                                                           | 164 |
| Figure 7.13: Question 4 of Section 3. ....                                                           | 164 |
| Figure 7.14: Question 5 of Section 3. ....                                                           | 164 |
| Figure 7.15: Question 6 of Section 3. ....                                                           | 165 |
| Figure 7.16: Question 7 of Section 3. ....                                                           | 165 |
| Figure 7.17: Were you able to sign up/register as a user successfully? .....                         | 166 |
| Figure 7.18: Were you able to log in after signing up? .....                                         | 166 |
| Figure 7.19: Did you receive an email after trying to reset password? .....                          | 167 |
| Figure 7.20: Were you able to reset your password successfully?.....                                 | 167 |
| Figure 7.21: Was there an error message when resetting password format was not followed?<br>.....    | 168 |
| Figure 7.22: Were you able to logout? .....                                                          | 168 |
| Figure 7.23: Were you able to upload an image? .....                                                 | 169 |
| Figure 7.24: Was the image previewed after uploading? .....                                          | 169 |
| Figure 7.25: Was a success message displayed after you uploaded an image? .....                      | 170 |
| Figure 7.26: Did the system allow you to upload an image without issues? .....                       | 170 |
| Figure 7.27: Were you able to type a question? .....                                                 | 171 |
| Figure 7.28: Were you able to click on the submit button? .....                                      | 171 |
| Figure 7.29: Did the web application redirect you to a response page after clicking submit?<br>..... | 172 |

|                                                                                                       |     |
|-------------------------------------------------------------------------------------------------------|-----|
| Figure 7.30: Did the generated result match what you expected? .....                                  | 172 |
| Figure 7.31: Were you able to start a new chat? .....                                                 | 173 |
| Figure 7.32: Were you redirected to the homepage?.....                                                | 173 |
| Figure 7.33: Were you able to click and open the sidebar menu? .....                                  | 174 |
| Figure 7.34: Were you able to view your account's chat history? .....                                 | 174 |
| Figure 7.35: Was your account's chat history correct? .....                                           | 175 |
| Figure 7.36: Was the interface easy to use? .....                                                     | 175 |
| Figure 7.37: Did you find it easy to understand what to do on each page? .....                        | 176 |
| Figure 7.38: Was it easy to navigate the web application? .....                                       | 176 |
| Figure 7.39: Did the layout and design feel clear and intuitive? .....                                | 177 |
| Figure 7.40: Were instructions easy to understand? .....                                              | 177 |
| Figure 7.41: Was the response time acceptable?.....                                                   | 178 |
| Figure 7.42: Did you experience any crashes, freezes or bugs? .....                                   | 178 |
| Figure 7.43: If you provided invalid input, did the system give you a helpful error message?<br>..... | 179 |
| Figure 7.44: If you submitted a blank prompt, did the system handle it gracefully? .....              | 179 |
| Figure 7.45: Did image a1.jpg gave the response 'centipede'? .....                                    | 180 |
| Figure 7.46: Did image a2.jpg gave the response 'deer'? .....                                         | 180 |
| Figure 7.47: Did image a3.jpg gave the response 'hawk'?.....                                          | 181 |
| Figure 7.48: Did image a4.jpg gave the response 'goat'?.....                                          | 181 |
| Figure 7.49: Did image a5.jpg gave the response 'turtle'? .....                                       | 182 |
| Figure 7.50: Did image p1.jpg gave the response 'fig'?.....                                           | 182 |
| Figure 7.51: Did image p2.jpg gave the response 'lichen'? .....                                       | 183 |
| Figure 7.52: Did image p3.jpg gave the response 'mushroom'? .....                                     | 183 |
| Figure 7.53: Did image p4.jpg gave the response 'lily'?.....                                          | 184 |
| Figure 7.54: Did image p5.jpg gave the response 'red ginger'?.....                                    | 184 |

## List of Tables

|                                                                                                  |     |
|--------------------------------------------------------------------------------------------------|-----|
| Table 2.1: Prompt format and syntax. ....                                                        | 15  |
| Table 2.2: Prompt and Answer for Rafflesia flower image.....                                     | 16  |
| Table 2.3: Prompt and Answer for Tulip flower image. ....                                        | 18  |
| Table 2.4: Prompt and Answer for Cheetah image. ....                                             | 20  |
| Table 2.5: Prompt and Answer for Rafflesia flower image for model paligemma-3b-mix-448.<br>..... | 22  |
| Table 2.6: Comparative analysis review of three (3) existing system. ....                        | 46  |
| Table 3.1: Software requirements. ....                                                           | 55  |
| Table 3.2: Hardware requirements. ....                                                           | 56  |
| Table 5.1: User registration page. ....                                                          | 133 |
| Table 5.2: User login page. ....                                                                 | 134 |
| Table 5.3: Forgot password page. ....                                                            | 136 |
| Table 5.4: Password reset page. ....                                                             | 136 |
| Table 5.5: Image Upload.....                                                                     | 138 |
| Table 5.6: Response Generation .....                                                             | 140 |
| Table 5.7: Chat History .....                                                                    | 141 |
| Table 5.8: Help Menu .....                                                                       | 142 |
| Table 5.9: Language Menu .....                                                                   | 143 |
| Table 5.10: Image and Prompt Submission .....                                                    | 143 |
| Table 5.11: AI Model Generates Response .....                                                    | 144 |
| Table 5.12: AI Model Failure.....                                                                | 145 |
| Table 5.13: Model Performance Testing Results .....                                              | 151 |

## Abstract

*Biodiversity is essential for maintaining ecological balance, yet identifying and understanding different species can be challenging, especially for non-experts. With artificial intelligence (AI) transforming the way we tackle complex problems, this project explores how AI can make nature identification more accessible. It introduces a web application powered by PaliGemma, a state-of-the-art vision-language model developed by Google, designed to help users identify plants and animals simply by uploading images and prompting questions. The motivation behind this project is the need for an easy-to-use and efficient tool that supports researchers, educators, and the general public in learning about nature and wildlife. Traditional species identification methods often require specialized knowledge, making them difficult for many people to use. This project aims to bridge that gap by providing an AI-driven solution that delivers accurate species information, including names, classifications, and habitats. The main goals of this project are to develop a user-friendly web application, integrate PaliGemma for precise species recognition, and assess the system's real-world effectiveness. While the project prioritizes accuracy, usability, and accessibility, it also acknowledges limitations such as integrating AI into a web platform, the model's difficulty in recognizing rare species, and potential issues with training data and image quality. Following a modified waterfall approach with iteration between phases, the development process includes requirement analysis, system design, implementation, testing, and maintenance. The expected outcome is a reliable and interactive tool that not only simplifies species identification but also fosters curiosity and appreciation for biodiversity. By leveraging AI, this project makes learning about nature more engaging and accessible, contributing to the broader field of AI-driven conservation technology.*

## Abstrak

Biodiversiti amat penting dalam mengekalkan keseimbangan ekologi, namun mengenal pasti dan memahami pelbagai spesies boleh menjadi cabaran, terutamanya bagi mereka yang bukan pakar. Dengan kecerdasan buatan (AI) yang mengubah cara kita menangani masalah kompleks, projek ini meneroka bagaimana AI boleh menjadikan proses mengenal pasti alam semula jadi lebih mudah diakses. Projek ini memperkenalkan aplikasi web yang dikuasakan oleh PaliGemma, model visi-bahasa canggih yang dibangunkan oleh Google, yang membolehkan pengguna mengenal pasti tumbuhan dan haiwan dengan hanya memuat naik gambar dan bertanya soalan. Motivasi di sebalik projek ini adalah keperluan untuk alat yang mudah digunakan dan cekap bagi membantu penyelidik, pendidik, dan masyarakat umum dalam mempelajari tentang alam semula jadi dan hidupan liar. Kaedah tradisional dalam mengenal pasti spesies sering memerlukan pengetahuan khusus, menjadikannya sukar diakses oleh kebanyakan orang. Projek ini bertujuan untuk merapatkan jurang tersebut dengan menyediakan penyelesaian berasaskan AI yang memberikan maklumat spesies dengan tepat, termasuk nama, klasifikasi, dan habitatnya. Matlamat utama projek ini adalah untuk membangunkan aplikasi web yang mesra pengguna, mengintegrasikan PaliGemma bagi pengecaman spesies yang lebih tepat, serta menilai keberkesanan sistem dalam penggunaan dunia sebenar. Walaupun projek ini memberi keutamaan kepada ketepatan, kebolehgunaan, dan kebolehcapaian, ia juga mengambil kira beberapa batasan seperti cabaran dalam mengintegrasikan AI ke dalam platform web, kesukaran model dalam mengenal pasti spesies yang jarang ditemui, serta isu berkaitan data latihan dan kualiti imej. Menggunakan pendekatan model air terjun yang diubah suai dengan iterasi antara fasa, proses pembangunan meliputi analisis keperluan, reka bentuk sistem, pelaksanaan, ujian, dan penyelenggaraan. Hasil yang diharapkan adalah sebuah alat yang boleh dipercayai dan interaktif yang bukan sahaja memudahkan pengecaman spesies tetapi juga memupuk rasa ingin tahu dan penghargaan terhadap biodiversiti. Dengan memanfaatkan AI, projek ini menjadikan pembelajaran tentang alam semula jadi lebih menarik dan mudah diakses, sekaligus menyumbang kepada bidang teknologi pemuliharaan berasaskan AI.

## CHAPTER 1: INTRODUCTION

### 1.1 Introduction

In the era of Artificial Intelligence (AI) technology becoming an uprising trend, there are a wide range of available AI chatbot tools which optimizes Machine Learning (ML) tasks such as Question Answering (QA) with the capability of understanding human questions and giving personalised answers. The most common AI chatbot like ChatGPT is a prime example of how an AI chatbot has helped humans with their daily lives, helping them answering from the simplest questions such as summarizing a children's story book to answering difficult questions from a university's computer science students such as writing a program's code.

This project aims to develop a web application for nature and wildlife identification using PaliGemma as the Visual-Language Model. PaliGemma, is a fairly new and recently developed Visual-Language Model developed by Google with the purpose of improving multimodal models that can work across different type of data such as text and images. In essence, PaliGemma is a vector language modeler (VLM) that accepts one or more images as input together with a textual task description, often known as the prompt or inquiry (prefix) while the response, which we frequently refer to as the suffix, is the text string that PaliGemma then autoregressively generates as a prediction (Beyer et al., 2024).

The web application will feature an interface that resembles a messaging application interface where users can upload images of plants, animals and other wildlife. Furthermore, the PaliGemma model will then process these images and return responses about the identified species which includes common name, scientific classifications and habitat details. Moreover, this proposed project's main focus is to test PaliGemma's capability in understanding images related to the nature and wildlife domain. Additionally, the application will also provide an

opportunity for users to contribute to a growing database, promoting citizen science and community involvement in nature and wildlife conservation efforts.

## **1.2 Problem Statement**

Generally, ChatGPT is a general chatbot which answers every type of question or statement, ranging from categories such as art, music, fashion to educational questions. However, it is not specified to a category in which it almost always gives a generalised answer. Specific categories for example, nature, will help nature enthusiast or anyone interested in expanding their knowledge of the nature. Although ChatGPT can also generate descriptions based on both image and text input, it is not specified to a category such as nature and wildlife. Identifying species in the wild can be challenging which often requires an in-depth knowledge and experience. This is a challenge for the general population because such information is frequently not readily available through conventional methods of species identification, such as field guide or expert consultations.

The exploration and development of the system, powered by PaliGemma will provide accurate species identification but also detailed description and recommendation based on the visual input submitted by users. PaliGemma plays a role in this project as a visual-language model that is designed to understand and process visual data with text inputs enhancing the experience for users seeking to explore and learn about biodiversity.

## **1.3 Objectives**

1. To review existing systems of Large Language Models and compare their capabilities with PaliGemma as the Visual-Language Model.
2. To develop a web application chatbot which integrates PaliGemma for both visual and text-based input for educational purposes.

3. To evaluate the performance of the PaliGemma in describing images related to nature and wildlife based on user prompts.

## 1.4 Methodology

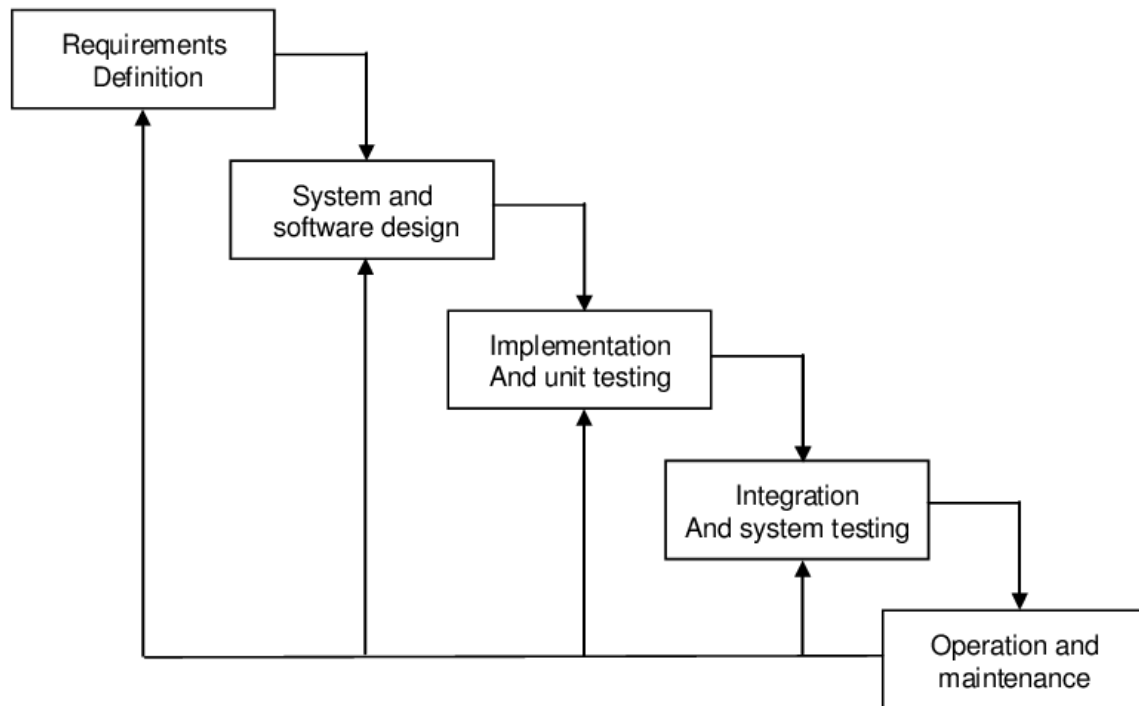


*Figure 1.1: Modified Waterfall Model with Iteration between phases.*

The methodology that this project will use is the Modified Waterfall Model with Iteration between phases which can be seen in Figure 1.1. There are 5 life cycles in this model which are requirements definition, system and software design, implementation and unit testing, integration and system testing, and operation and maintenance. Each phase will go through an iteration should that there needs to be any changes. The five phases will be explained below:

#### **1.4.1 Phase 1: Requirement Definition**

Gathering and defining all system requirements will start in the first phase by conducting a detailed analysis to understand user needs, such as species identification, educational content, and PaliGemma integration for visual recognition.

#### **1.4.2 Phase 2: System and Software Design**

In the design phase, the actions done will be to develop the architectural and technical design for the web application by including design documents that consists of architecture diagrams, UI mockups, and a detailed integration plan for PaliGemma. UML class diagram, activity diagram and sequence diagram will be designed using LucidChart based on the requirement definition including the design of PaliGemma integration plan. The UI mockups will be first designed on Figma as a prototype before moving forward to implementation of the frontend design.

#### **1.4.3 Phase 3: Implementation and Unit Testing**

The development of the web application's unit features will be done in this phase, initializing the database and backend setups of the web application including user database then building the frontend according to the Figma prototype design in Phase 2 and lastly, perform PaliGemma integration into the web application for species recognition. And finally, performing unit testing for each of the features, emphasizing especially on PaliGemma's performance.

#### **1.4.4 Phase 4: Integration and System Testing**

In this phase, the integration of all components will be done to test the system as a single, functioning web application which consists of the web application's frontend, backend,

PaliGemma and database. The system testing is to validate the initial system requirements in terms of functionality, usability, performance and user experience.

#### **1.4.5 Phase 5: Operation and Maintenance**

The final phase will oversee the deployment and launch of the web application into a hosting server for user to easily access it online while also performing maintenance works on the web application based on user feedback and experience.

### **1.5 Project Scope**

The case study of this project will cover on the theme of nature and wildlife, specifically to identify any images involving nature and wildlife species. The focus of this project is on the system development of a web application chatbot which will be integrated with a Vision-Language Model, PaliGemma for image captioning. Furthermore, the system will enable users to upload an image and input text to identify a plant or wildlife species. Hence, the targeted user of this system will be the public user to achieve the educational goal of this system.

### **1.6 Significance of Project**

The significance of this project is to provide an AI-based web application chatbot on a specific case study such as nature and wildlife species identification for the public users, primarily focusing on its educational approach. PaliGemma is a newly developed vision-language model by Google focusing on image captioning by receiving user prompts mainly through an image upload and text-based input so exploring a new vision-language model such as the PaliGemma is the next step to gaining new knowledge especially in the Artificial Intelligence field. Thus, grasping the core of the vision-language model in understanding how the model recognizes images and comprehending the output of the model after a user prompts a command.

## 1.7 Project Schedule

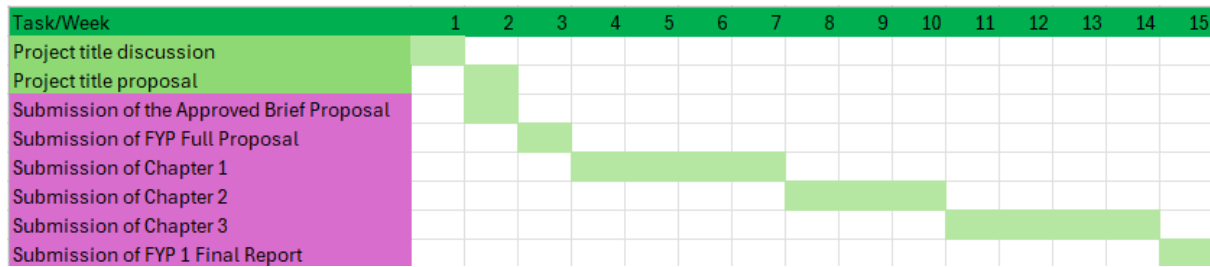


*Figure 1.2: Project Schedule Gantt Chart.*

The initial discussion of the project title with final year project’s supervisor began in Week 1 (7/10/2024 – 13/10/2024) and took place over two sessions. These sessions mainly covered the key aspects of the project including the title, type, cluster, and selection of a case study for the project. After thorough consideration, we decided on a systems development and within the AI cluster, focusing on a case study of nature and wildlife identification

The project title, “A Web Application Chatbot for Nature and Wildlife Identification using PaliGemma as the Vision-Language Model” was proposed to the supervisor and completion of the brief proposal is done waiting for approval from project’s supervisor. After the brief proposal was approved by project’s supervisor, the submission of the approved brief proposal was done on 20<sup>th</sup> October 2024 in Week 2 (14/10/2024 – 20/10/2024).

The feedback and comment from examiner on brief proposal were pending in Week 3 however, the examiner provided approval and feedback in Week 4 (21/10/2024 – 3/11/2024).

Once the examiner approved and commented on the brief proposal, work began on preparing the full proposal for submission on 14<sup>th</sup> November 2024 and two weeks were allocated to complete the full proposal from Week 5 until Week 6 (4/11/2024 – 17/11/2024). The full proposal will primarily include the project title, introduction, problem statement,

objective, methodology, scope, significance of project, project schedule and expected outcomes.

Once the full proposal is submitted, work on Chapter 1 of FYP 1 will resume, with the submission scheduled for 21<sup>st</sup> November 2024 in Week 7 of the semester (18/11/2024 – 24/11/2024). The contents of Chapter 1 will be similar to the full proposal but with the addition of a project report outline and a summary of Chapter 1 added at the end of the chapter.

The submission of Chapter 2 will be on 13<sup>th</sup> December 2024 and three weeks will be allocated to complete Chapter 2 of FYP 1 from Week 8 until Week 10 (25/11/2024 – 15/12/2024). Chapter 2 will focus on the literature review.

Chapter 3 will be submitted on 5<sup>th</sup> January 2025 and will be given three weeks to complete Chapter 3 of FYP 1 from Week 11 until Week 13 (16/12/2024 – 5/1/2025). Chapter 3 will cover the requirements, analysis and design of the proposed system.

The final report of FYP 1 will be submitted on 17<sup>th</sup> January 2025 and two weeks will be given to complete the final report from Week 14 until Week 15 (6/1/2025 – 19/1/2025). The final report contents will consist of Chapter 1, 2 and 3.

Presentation dates will be scheduled for 24<sup>th</sup> and 25<sup>th</sup> January 2025, should they be required.

## **1.8 Expected Outcome**

1. A functional web application which satisfies the functions of uploading image and textual input.
2. An educational tool for the public.
3. Integration of a visual-language model, PaliGemma into the web application chatbot.

4. Performance testing and evaluation of the web application's accuracy in identifying species in the wildlife and an overall user experience of the web application's use.

## **1.9 Project Report Outline**

### **1.9.1 Chapter 1: Introduction**

Chapter 1 highlights the introduction and description of the project that proposes the development of a web application chatbot for nature and wildlife identification using PaliGemma as the Vision-Language model. This chapter includes the problem statement, project objectives, methodology, project scope, significance of project, project schedule and project outcomes. The problem statement discusses the difficulties of species identification in the wild. The project objectives discuss the goals to be achieved in this proposed project. The chosen methodology to be used in this project will be Modified Waterfall Model with iteration between phases. The project scope highlights the targeted users and the boundaries of the nature and wildlife species identification. This project's significance highlights the educational purpose of the proposed project towards the targeted users. The project schedule will visualise the timing needed to complete the first part of this final year project using a Gantt chart. And finally, the project's outcome is to develop a functional web application chatbot integrated with a vision-language model as an educational tool for the targeted users.

### **1.9.2 Chapter 2: Literature Review**

Chapter 2 discusses on the comparison and review of existing Large Language Models and framework, thus highlighting why PaliGemma was chosen for this project. This chapter will also be reviewing other LLM applications to understand their features and compare it with PaliGemma.

### **1.9.3 Chapter 3: Requirement, Analysis and Design**

Chapter 3 describes the methodology selected for this project which is Modified Waterfall Model with Iteration between phases. This part of the chapter will utilize the UML activity diagram, use case and sequence diagram to visualize the flow of the web application.

### **1.9.4 Chapter 4: Implementation**

Chapter 4 oversees the implementation phase of the proposed project, covering the development of core features and integration of the PaliGemma model. This chapter will cover the technologies and tools used for the development, such as open-source code editor Visual Studio Code, Figma for UI design and other software development tools. Documenting the database setup, frontend and backend development, and the integration of visual-language capabilities for species identification.

### **1.9.5 Chapter 5: Testing**

Chapter 5 presents the testing phase, documenting the functionality, usability, and performance testing of the system. This chapter will describe the test cases created to validate each feature, including image upload, species identification, and response generation. The methods used to assess the system's accuracy and usability will also be included to make adjustments to the system based on testing results.

### **1.9.6 Chapter 6: Conclusion and Future Work**

Chapter 6 concludes the project by summarizing the key achievements and findings, reflecting on the achieved objectives, the impact of the system on user, and the overall performance in identifying nature and wildlife species. This chapter will also discuss the potential areas for future improvements such as fine-tuning the PaliGemma model for higher accuracy rate on nature and wildlife species identification.

### **1.10 Summary**

To summarize, this chapter describe the proposed project, the web application chatbot for nature and wildlife identification using PaliGemma as the vision-language model. This project's main objective is to evaluate the PaliGemma model's performance in image identification, especially in identifying image of nature and wildlife species. To serve its educational purposes, this system will be made available for the public for use.

## **CHAPTER 2: LITERATURE REVIEW**

### **2.1 Introduction**

This chapter will be thoroughly reviewing the three existing system's large language model (LLM) that shares similarities with PaliGemma, the proposed vision-language model (VLM) as a Question Answering (QA) system for this project. Through a comparative analysis of these systems, the features to incorporate into the proposed system will be determined based on the analysis results.

In addition to the review of the existing systems, this chapter will also outline the front-end development tools and back-end development tools to be used in this project.

### **2.2 Review on PaliGemma, a Vision-Language Model**

#### **2.2.1 Introduction of PaliGemma and PaliGemma's Architecture Review**

Developed by Google AI, PaliGemma is an open-source VLM which offers a unique method for combining textual and visual comprehension (Goh et al., 2024). PaliGemma is a multimodal model based on two other models, SigLIP model – a vision model and Gemma model – a large language model (LLM). It is a composition of a Transformer decoder and a Vision Transformer image encoder. PaliGemma accepts both image and text as input then generates text as the output. PaliGemma is an open model that combines the Gemma family of language models with the PaLI vision-language models (Beyer et al., 2024).

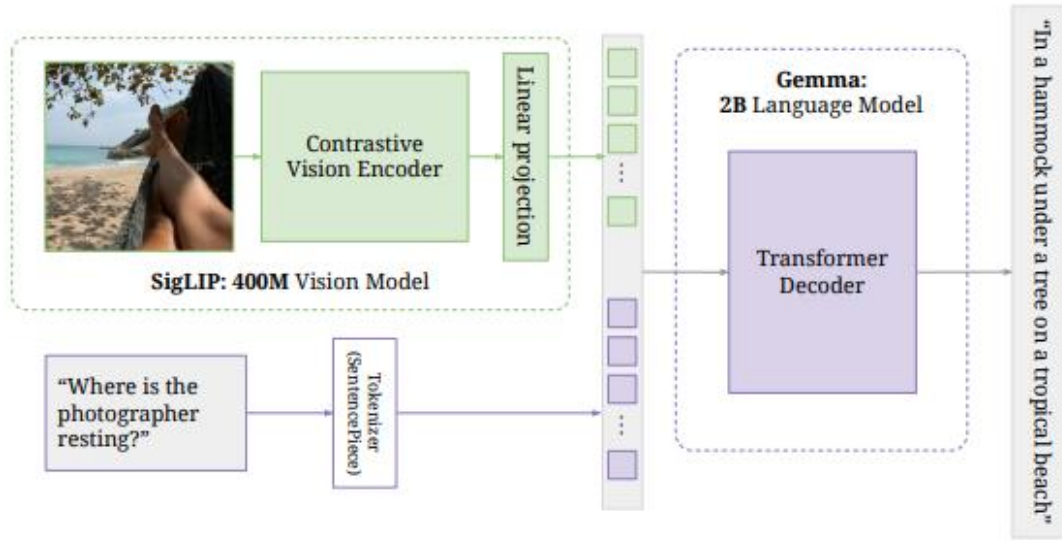


*Figure 2.1: PaliGemma's architecture visual overview.*

As shown in Figure 2.1, the architecture of PaliGemma consists of three components which are the SigLIP 400M Vision Model, Gemma 2B Language Model and Linear projection. The vision model output feeds into the language model through a linear layer projection by projecting the SigLIP's output tokens into the same dimensions as Gemma 2B's vocab tokens for concatenation.

PaliGemma's architecture consists of three unimodal components which are an image encoder, a decoder-only language model and a linear layer. The architecture's visual representation can be seen in Figure 2.1. As described by Beyer et al. (2024), the image encoder used is a publicly accessible SigLIP checkpoint, more precisely, the "shape optimized" ViTSo400m image encoder where the model has demonstrated state-of-the-art performance particularly considering its modest size and was contrastively pretrained at large scale using the sigmoid loss while the decoder-only language model used is an openly accessible Gemma-2B v1.0 raw pretrained checkpoint, which achieves an excellent trade-off between size and

speed and finally, the linear layer projects the output tokens of SigLIP into the same dimensions as the vocab tokens of Gemma-2B to enable concatenation. Gemma, a lightweight decoder-only language model made for effective text production, powers PaliGemma's text decoding capabilities (Goh et al., 2024).

### **2.2.2 PaliGemma's Pretraining Stages**

PaliGemma comes in two states, pretrained and fine-tuned. As a pretrained model, it can process both images and text as input and generate text as output. The tasks it can perform as a pretrained model are image captioning, visual question answering, image segmentation and video input. This model is pretrained on a mixture of datasets such as Web Language Image (WebLI), CC3M-35L, VQA-CC3M-35L, VQG-CC3M-35L, OpenImages, and WIT. The pretraining stages include 4 stages which are:

- 1) Stage0: Unimodal pretraining
- 2) Stage1: Multimodal pretraining
- 3) Stage2: Resolution increase
- 4) Stage3: Transfer

Starting off from Stage0, Beyer et al. (2024) mentioned PaliGemma relies on pre-existing, publicly accessible checkpoints rather than performing any new unimodal pretraining. In Stage1 is where the unimodal components mentioned in 2.2.1 are combined and deviation from standard procedure of freezing the image encoder in the interest of gaining additional abilities during pretraining (Beyer et al., 2024). Successing from Stage1 is Stage2 in which the main focus is expanding the model's capacity to comprehend higher-resolution photos by increasing the resolution to both 448 x 448 pixel and 896 x 896 pixel according to Beyer et al.

(2024). And lastly in Stage3, Beyer et al. (2024) remarked that it will be the transfer of the base model to a variety of distinct academic benchmarks using a straightforward, unified transfer formula with a small number of hyper-parameters to demonstrate their efficacy.

In HuggingFace, a machine learning (ML) and data science platform and community, under Google's collections there is a section for PaliGemma's release containing the pretrained models with differing resolutions namely paligemma-3b-pt-224, paligemma-3b-pt-448 and paligemma-3b-pt-896. This section also contains other pretrained models of PaliGemma mainly trained for a mixture of tasks.

While the pretrained models are accessible, the fine-tuned models are accessible in HuggingFace as well with 39 models' variations. The models are fine-tuned based on the tasks mentioned Beyer et al. (2024).

### **2.2.3 “Inference PaliGemma with Keras” – An Existing Project Utilizing PaliGemma from Kaggle Review**

Due to limited research papers available on PaliGemma, the review of PaliGemma will include projects utilizing PaliGemma, such as from Kaggle, a data-science community platform. The project to be reviewed is from the author Nilay Chauhan under notebook name “Inference PaliGemma with Keras”<sup>1</sup>. This project experimented with PaliGemma's inference using Keras framework. It follows up with the environment setups and then the testing of the model in local machine. The prompts to be used in PaliGemma has a format for the model to process the prompt request which can be referred to in Table 1.

---

<sup>1</sup> <https://www.kaggle.com/code/nilaychauhan/inference-paligemma-with-keras>

| Prompt format                                                                        | Prompt syntax format                                                                                   | Examples                                                                               |
|--------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"> <li>answer en<br/>[question/command]\n</li> </ul> | <ul style="list-style-type: none"> <li>the prompt has to end with “\n”</li> </ul>                      | <ul style="list-style-type: none"> <li>“answer en what color is the cow?\n”</li> </ul> |
| <ul style="list-style-type: none"> <li>describe en\n</li> </ul>                      | <ul style="list-style-type: none"> <li>‘en’ means English for the language for the response</li> </ul> | <ul style="list-style-type: none"> <li>“detect cow\n”</li> </ul>                       |
| <ul style="list-style-type: none"> <li>detect [image focus point]\n</li> </ul>       |                                                                                                        | <ul style="list-style-type: none"> <li>“describe en\n”</li> </ul>                      |
| <ul style="list-style-type: none"> <li>segment [image focus point]\n</li> </ul>      |                                                                                                        |                                                                                        |

*Table 2.1: Prompt format and syntax.*

In the Table 2.1, the prompt format would be the format users use to prompt the model and the prompt syntax format is the syntax of the prompt that users should include in their prompt. Hence, the format would generally be as the examples.

The sample data Nilay Chauhan used was a picture of ‘a cow standing on a beach’. However, when the picture is tested on the PaliGemma model, it can describe the image comprehensively. An example is when the image is uploaded then user prompts “describe en”, the model responds with a text response:

“A brown cow stands on the wet sand at the beach, its head resting on its legs. The sky is clear and blue with a few white clouds. The water is calm and blue, reflecting the sky. The cow's horns are sharp and its eyes are bright. The sand is wet and the waves crash gently against the shore. The cow's shadow stretches long across the sand. The photographer's logo is visible in the bottom right corner of the image.”.

### 2.2.4 PaliGemma Demo Review - HuggingFace

In HuggingFace under PaliGemma release section, there is a PaliGemma demo<sup>2</sup> part where users can test the PaliGemma's capability before applying manual testing in their local environment. Users can upload an image, type a prompt and choose the model to test with. The models available for selection are paligemma-3b-mix-224 and paligemma-3b-mix-448. Below the model's demo are the example data for the demo testing. For the purpose of literature review, the PaliGemma demo will be tested with two images of two different flowers and an image of a rooster. Below is the result of the PaliGemma demo testing.

Model: **paligemma-3b-mix-224**

| Image 1: Rafflesia flower                  |            |
|--------------------------------------------|------------|
| Prompt 1:                                  | Answer 1:  |
| 1a) What is this flora species?            | 1a) Red    |
| 1b) What is this species?                  | 1b) Flower |
| 1c) What is this flower's scientific name? | 1c) Rose   |

*Table 2.2: Prompt and Answer for Rafflesia flower image.*

As described in Table 2.2, the first image used to test on PaliGemma demo which is the rafflesia flower and lists three different prompts used to test the image captioning capability of PaliGemma with three differing answers generated by PaliGemma to the three different prompts. This shows the model's capability in understanding the questions thus generating answers based on prompts although the answers provided are not accurate. Theoretically, the differing answers are due to the flower's rarity and locality whereby the flower can only be found in the rainforest of Borneo Island which means PaliGemma's flora species recognition

---

<sup>2</sup> <https://huggingface.co/spaces/big-vision/paligemma>

capability is limited to an extent.

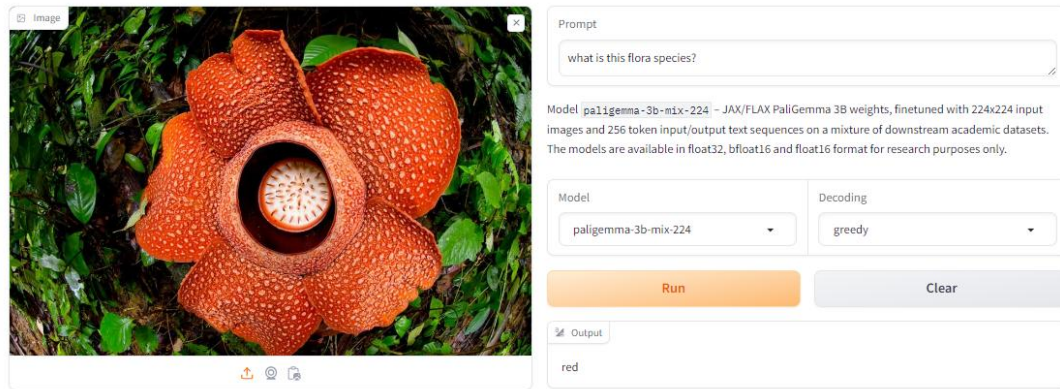


*Figure 2.2: PaliGemma demo with prompt 1a) and answer 1a).*

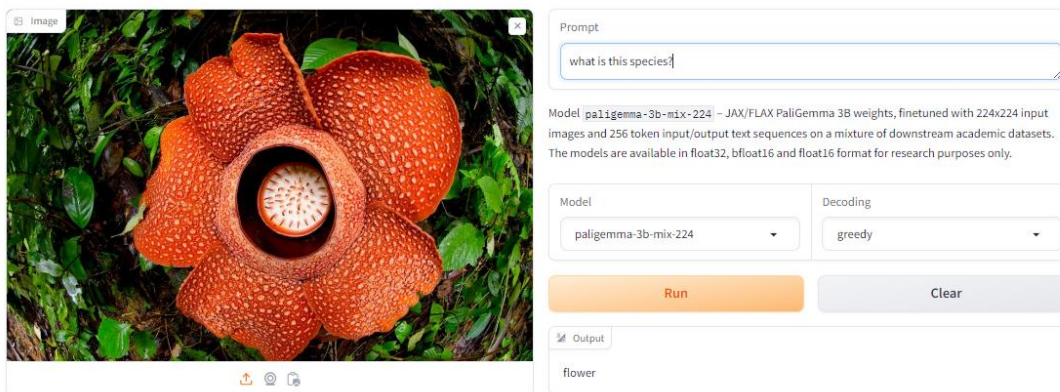


*Figure 2.3: PaliGemma demo with prompt 1b) and answer 1b).*

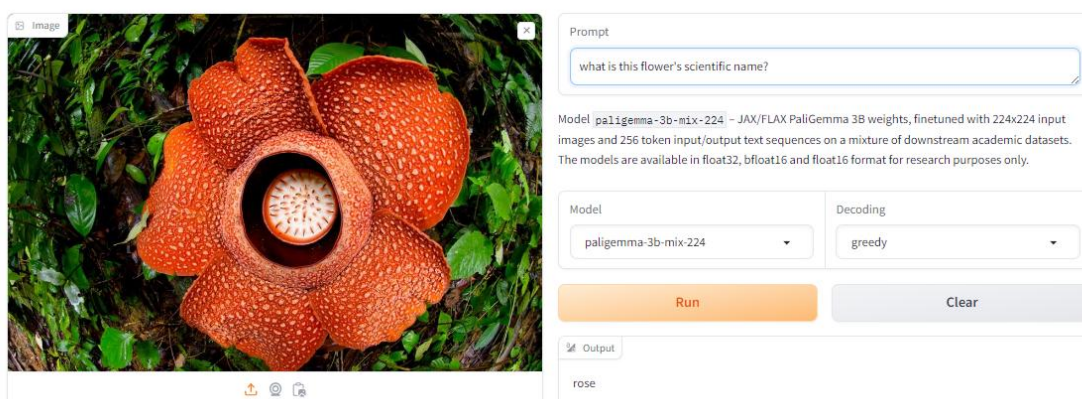


*Figure 2.4: PaliGemma demo with prompt 1c) and answer 1c).*

Figure 2.2, Figure 2.3 and Figure 2.4 shows the results of PaliGemma demo tested using an image of the rafflesia flower sequenced by the order of prompts and answers based on Table 2.2.

| Image 2: Tulip flower                      |           |
|--------------------------------------------|-----------|
| Prompt 2:                                  | Answer 2: |
| 2a) What is this flora species?            | 2a) tulip |
| 2b) What is this species?                  | 2b) tulip |
| 2c) What is this flower's scientific name? | 2c) tulip |

*Table 2.3: Prompt and Answer for Tulip flower image.*

Based on Table 2.3, the second image used to test on PaliGemma demo is the tulip flower as well as listing the prompts used to test this image on PaliGemma demo and answers generated by the model based on the given image. The answers generated by PaliGemma demo for the tulip flower in Table 2.3 differs compared to the answers generated for the image of rafflesia flower in Table 2.2. The answers are the same instigating that PaliGemma recognizes the tulip flower better than the rafflesia flower in Figure 2.2. Although the prompts in Table 2.3 are the same as the prompts in Table 2.2, the results differ which further theorize PaliGemma's flora species recognition capability to be limited in scope.

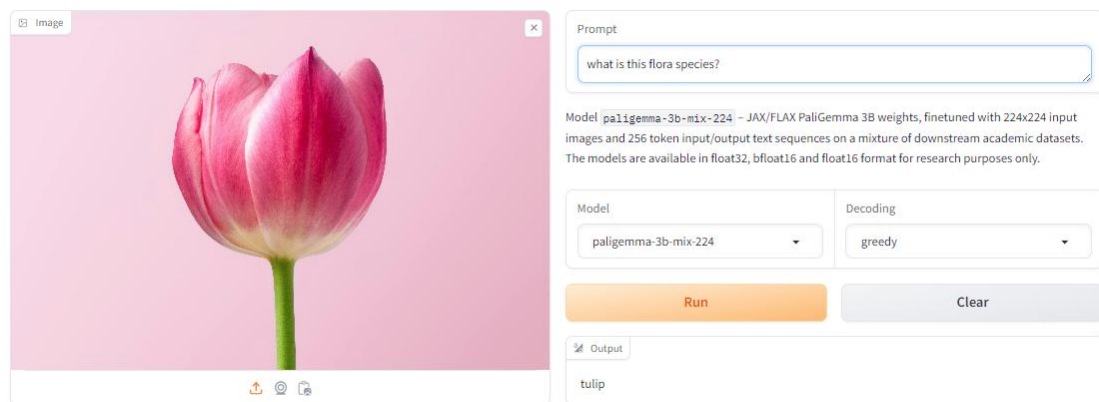


Figure 2.5: PaliGemma demo with prompt 2a) and answer 2a).

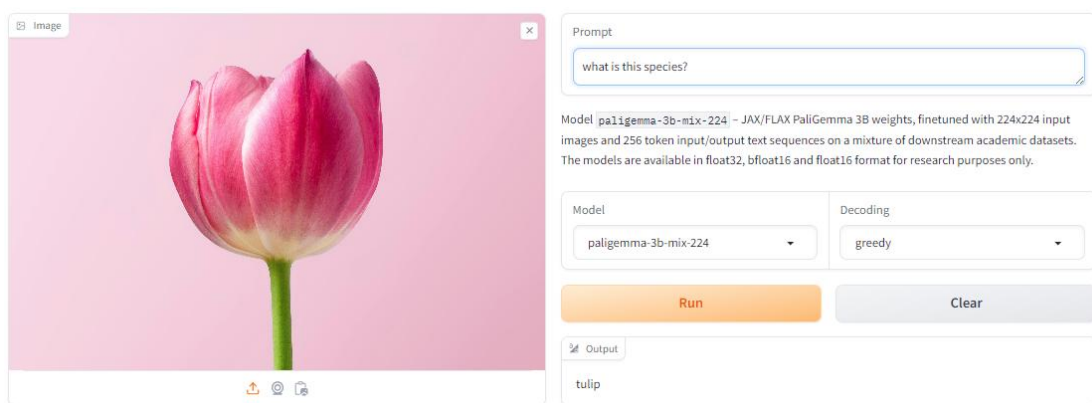


Figure 2.6: PaliGemma demo with prompt 2b) and answer 2b).

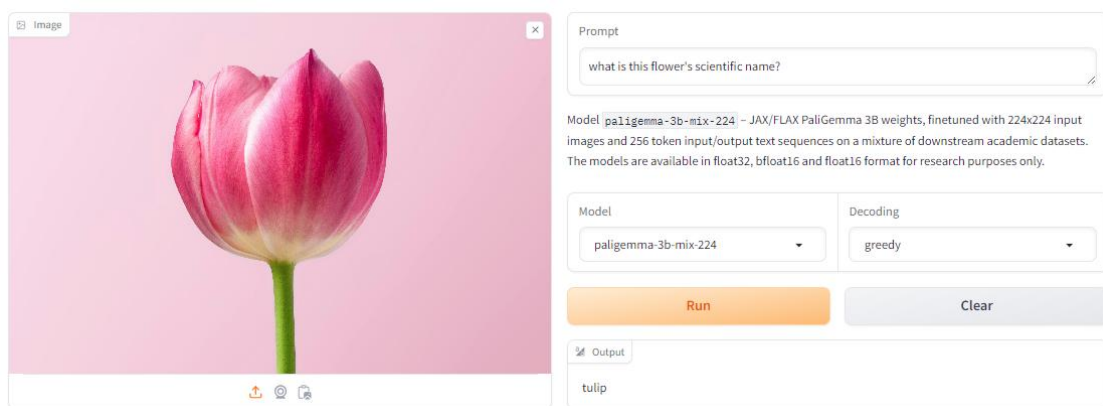


Figure 2.7: PaliGemma demo with prompt 2c) and answer 2c).

The prompts and answers for tulip flower are shown in Figure 2.5, Figure 2.6, and Figure 2.7.

|                                            |             |
|--------------------------------------------|-------------|
| Image 3: Cheetah                           |             |
| Prompt 3:                                  | Answer 3:   |
| 3a) What is this animal?                   | 3a) cheetah |
| 3b) What is this species?                  | 3b) cheetah |
| 3c) What is this animal's scientific name? | 3c) cheetah |

*Table 2.4: Prompt and Answer for Cheetah image.*

In table 2.4, the test image is of a fauna species, cheetah. The prompts' structure is similar to the prompts structure of rafflesia flower in Table 2.2 and tulip flower in Table 2.3 however, only the word "flower" is replaced with the word "animal" specifying the main focus of the image. The answers for all three prompts are the same which is cheetah. The results produced are similar though not entirely identical, to tulip flower's answers in table 2.3. This shows that the model recognizes fauna species well, such as the cheetah, well known for being the fastest fauna species on land.

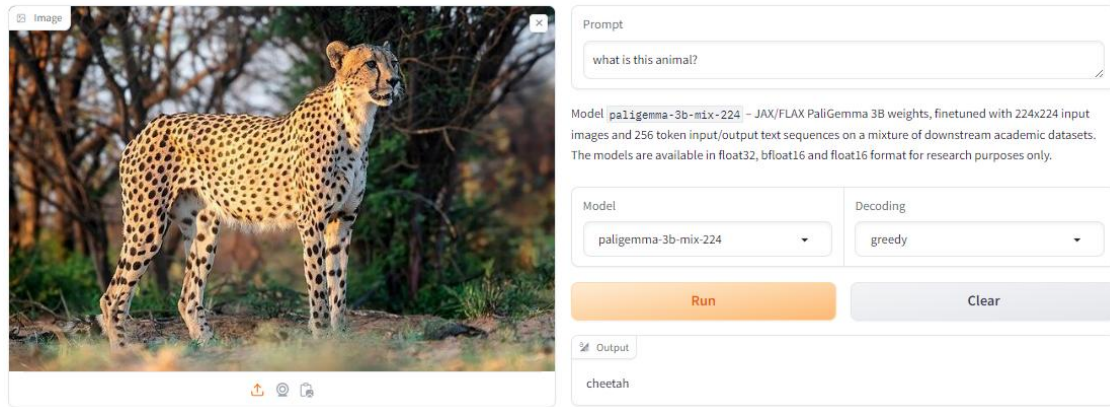


Figure 2.8: PaliGemma demo with prompt 3a) and answer 3a).

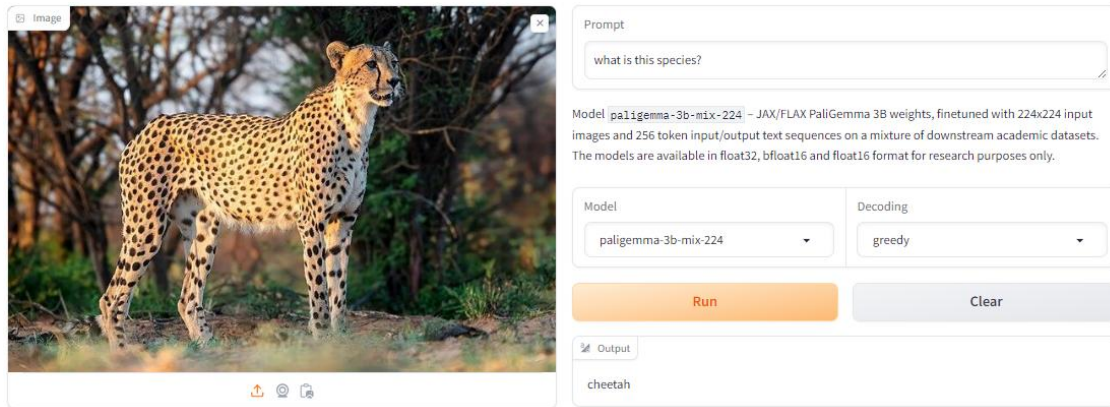


Figure 2.9: PaliGemma demo with prompt 3b) and answer 3b).

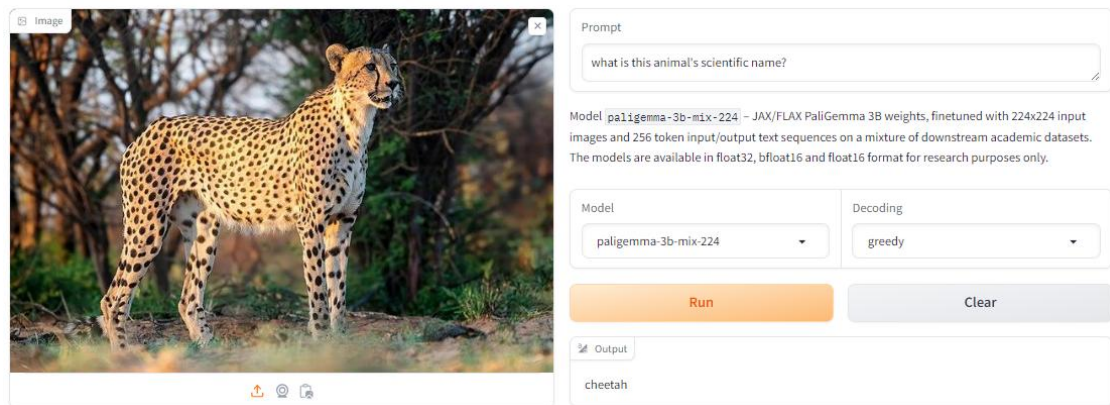


Figure 2.10: PaliGemma demo with prompt 3c) and answer 3c).

The results of prompts for the image cheetah shown in Figure 2.8, Figure 2.9 and Figure 2.10.

The PaliGemma demo model test is continued with another variant of the model, paligemma-3b-mix-448 resolution. The test subjects are the same as Table 2.2, Table 2.3 and Table 2.4 with the same prompts. However, the results are comparable with results obtained using the paligemma-3b-mix-224 resolution variant.

Model: **paligemma-3b-mix-448**

| Image 1: Rafflesia flower                  |                  |
|--------------------------------------------|------------------|
| Prompt 1:                                  | Answer 1:        |
| 1a) What is this flora species?            | 1a) unanswerable |
| 1b) What is this species?                  | 1b) unanswerable |
| 1c) What is this flower's scientific name? | 1c) unanswerable |

*Table 2.5: Prompt and Answer for Rafflesia flower image for model paligemma-3b-mix-448.*

In table 2.5, the test image is a rafflesia flower, similar to Table 2.2. The prompts used to test this variant is the same as in Table 2.2. Although the results produced for all three prompts in table 2.5 are the same, it is a result that invites discussion on the different capabilities of the different model's variants. Compared to the answer in table 2.2, the answers in table 2.5 are "unanswerable". This shows the differing capabilities between the varying PaliGemma variants.

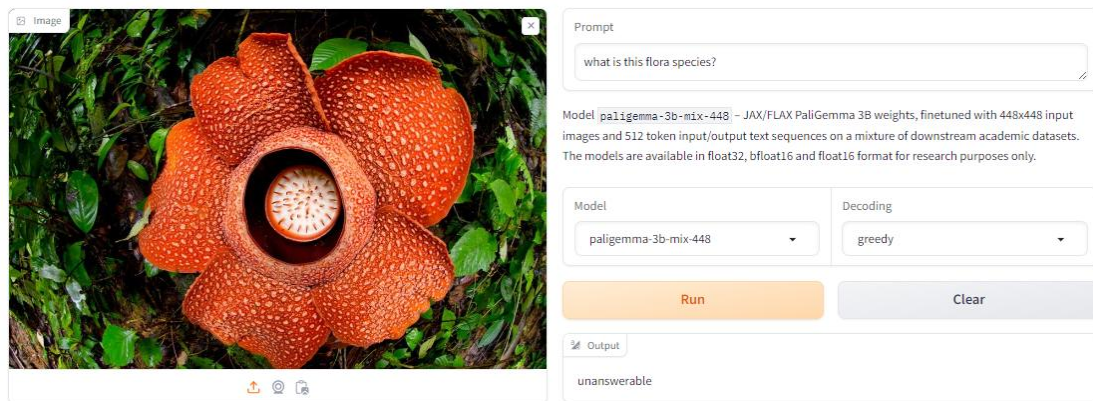


Figure 2.11: PaliGemma demo with prompt 1a) and answer 1a).

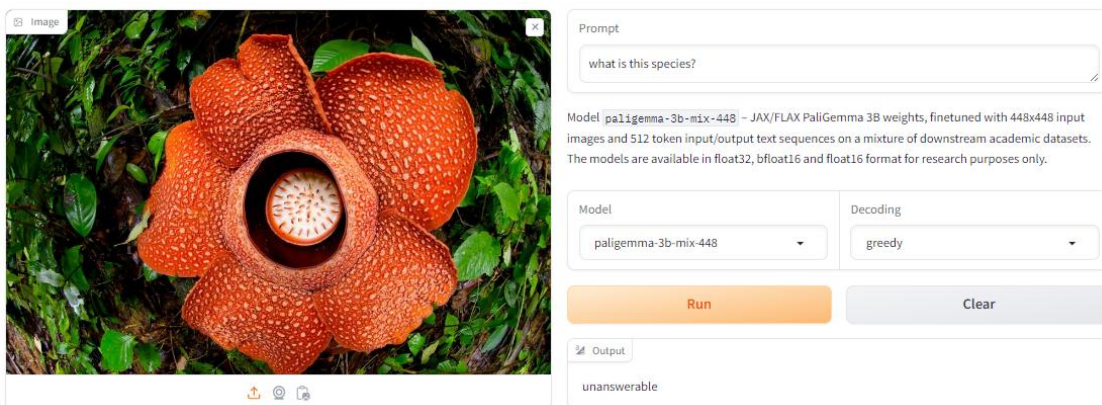


Figure 2.12: PaliGemma demo with prompt 1b) and answer 1b).

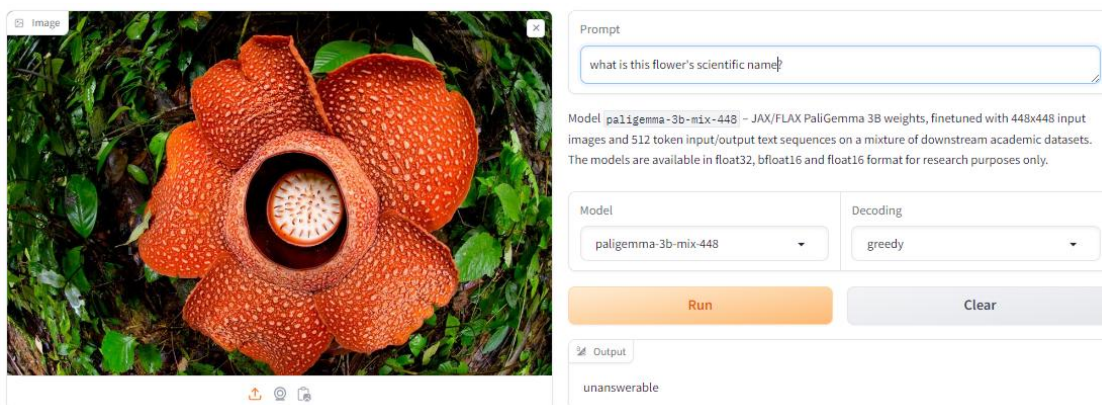


Figure 2.13: PaliGemma demo with prompt 1c) and answer 1c).

In the model paligemma-3b-mix-448, the result for tulip flower and cheetah are the same however for rafflesia flower, the result differs significantly. The answers in Figure 2.11, Figure 2.12, and Figure 2.13 are the same which is “unanswerable” compared to the answers obtained in model paligemma-3b-mix-224 where the answers for all three prompts in Figure 2.2, Figure 2.3, and Figure 2.4 are different – red, flower, rose. The result of the answers for Rafflesia flower for both paligemma-3b-mix-224 and paligemma-3b-mix-448 models are likely due to the locality of the flower, being that it is a flower that could only be found in the rainforest of Borneo Island.

### **2.2.5 PaliGemma Article by Google Review**

PaliGemma is an open vision-language model developed by Google with Lucas Beyer and XiaoHua Zhai leading the development. According to Beyer et al. (2024), the primary objective of their work is to offer a flexible base VLM. Other than that, Beyer et al. (2024) also stated that their main objective is not just to match the modalities but also to train a basic model that adapts effectively to a variety of tasks. The model is designed to be a foundational tool for researchers and developers, enabling them to build upon its capabilities and explore innovative applications in computer vision and natural language processing (NLP). Beyer et al. (2024) stated that by offering the base model without instruction tuning available, they intend to promote a more distinct division between base models and fine-tunes in VLM research and provide a helpful foundation for future studies in instruction tuning and specific applications.

### **2.2.6 Finetunability of PaliGemma**

In conclusion, the model works as both a pretrained model and fine-tuned model however the PaliGemma’s ability to be fine-tuned on custom data is the highlight of this model, contributing to the advancement of multimodal AI. Google also mentioned that PaliGemma is

not for direct use but rather serves as an experimental material for researchers to experiment with a vision-language model. As stated by both Leo Ueno (2024) and Trevor Lynn (2024), although Google claims that PaliGemma is "not designed to be used directly, but to be transferred (by fine-tuning) to specific tasks using a similar prompt structure," this means that any baseline we can observe with the model weights is merely the beginning of the model's potential utility in a particular situation. Nonetheless, PaliGemma can be further fine-tuned and trained to a domain specific task depending on research purposes.

## **2.3 Review on PaliGemma 2, An Upgrade of PaliGemma**

### **2.3.1 PaliGemma 2 Architecture and Pretraining Review**

PaliGemma 2 is an upgraded version of PaliGemma released by Google in HuggingFace<sup>3</sup> platform on December 5<sup>th</sup>, 2024. Though upgraded, PaliGemma 2 still incorporates the same architecture and utilizes the same training recipe as PaliGemma (Steiner et al., 2024). However, a change in its language model, from Gemma 2B to Gemma 2, creates new PaliGemma 2 base VLMs at three distinct sizes (3B, 10, 28B) and three distinct resolutions (224px<sup>2</sup>, 448px<sup>2</sup>, 896px<sup>2</sup>) as stated by Steiner et al. (2024). PaliGemma 2 has three different sizes – 3B, 10B and 28B, in comparison to PaliGemma with only one size – 3B, suggesting potential improvements of the upgraded model at different sizes. According to Steiner et al. (2024), the upgraded version is designed to be fine-tuned. Steiner et al. (2024) mentioned that at the same resolution and model size, PaliGemma 2 performs somewhat better than PaliGemma, but achieves notable gains at bigger model sizes.

In terms of architecture, PaliGemma 2 consists of a pretrained SigLIP-So400m vision encoder and a whole range of Gemma 2 language models from 2B up until 27B model

---

<sup>3</sup> <https://huggingface.co/collections/google/paligemma-2-release-67500e1e1dbfdd4dee27ba48>

according to Steiner et al. (2024). Steiner et al. (2024) stated that they employed the same pretrained SigLIP-So400m vision encoder and apply a linear projection to translate its series of embeddings to the Gemma 2 input space. The Gemma 2 language model receives the visual embeddings along with a prefilled text prompt (Steiner et al., 2024). After that, predictions are retrieved from the language model using autoregressive sampling (Steiner et al., 2024).

Steiner et al. (2024) mentioned that they employed the same three-stage training process as PaliGemma to give these VLMs a wide range of capabilities. The pretraining stages includes Stage 0 where unimodal pretraining of sigmoid loss for language image pre-training and Gemma model are done (Steiner et al., 2024).

Stage 1 is where the multimodal pretraining commence by concurrently training the vision encoder and Gemma 2 checkpoints on a multimodal task mixture of 1 billion examples which allows for fine-tuning to facilitate transferability to a variety of tasks in  $224\text{px}^2$  resolution.

Stage 2 usually increases the resolution from  $224\text{px}^2$  resolution to  $448\text{px}^2$  resolution and  $896\text{px}^2$  resolution. According to Steiner et al. (2024), Stage 2 trains for 50 million instances at  $448\text{px}^2$  resolution initially, followed by 10 million examples at  $896\text{px}^2$  resolution; during this stage, the components of the task mixture are the same, but the output sequence length is increased and tasks that benefit from high resolution are upweighted.

Stage 3 is normally where the transfer is done. As described by Steiner et al. (2024), Stage 3 fine-tunes the checkpoints from Stage 1 or Stage 2 to the intended task, in accordance with the resolution, by analyzing the same collection of hyperparameters from PaliGemma (COCO Captions, Remote Sensing VQA, Video Captioning, or InfographicQA) and taking into account an identical set of benchmarks.

### **2.3.2 PaliGemma 2 Fine-tuning Tasks**

Beyer et al. (2024) stated that the first version of PaliGemma were transferred to more than 30 academic benchmarks via fine-tuning however, Steiner et al. (2024) also mentions that PaliGemma 2 will incorporate the same transfer tasks and also explore new tasks which were not explored in depth. These tasks include:

- 1) Text detection and recognition
- 2) Table structure recognition
- 3) Molecular structure recognition
- 4) Optical music score recognition
- 5) Long caption generation
- 6) Spatial reasoning
- 7) Radiography report generation

Will further describe each tasks...

Steiner et al. (2024) mentioned that they used PaliGemma 2 for sophisticated OCR that includes word recognition and localization from photos and in addition, they tested PaliGemma 2 using the most widely used OCR benchmarks, ICDAR'15, Total-Text, MLT17 and MLT19, HierText, TextOCR, and IntelOCR, and they refined it using a combination of these train splits. It was emphasized that, unlike the OCR literature, this result may be achieved by simply optimizing a general-purpose VLM that does not rely on task-specific design components (Steiner et al., 2024). According to Steiner et al. (2024), this demonstrates the flexibility of PaliGemma 2's interface and the advantages of OCR-related pretraining in Stages 2 and 3. Additionally, Steiner et al. (2024) attempted to lower the resolution, which resulted in

significantly worse prediction quality; nevertheless, increasing the model size did not contribute to improvements.

According to Steiner et al. (2024), extracting table text content, matching bounding box coordinates, and the HTML table structure from document images is the aim of table structure recognition. Steiner et al. (2024) refined (the train splits of) two well-known data sets, FinTabNet, which contains 113k financial report tables from the annual reports of S&P 500 companies, and PubTabNet, which contains 516k images of tabular data from the PubMed Central Open Access Subset (commercial use collection), in order to transfer PaliGemma 2 to this task. The Tree Edit Distance Similarity (TEDS) and the Grid Table Similarity (GriTS), two families of metrics that gauge cell text content, cell topology/structure, and bounding box quality, are used to evaluate the quality of the model (Steiner et al., 2024). Steiner et al. (2024) also attempted expanding the model size, but this did not yield any further advantages, and employing a lower image resolution resulted in a slight decline in quality.

In order to deduce the molecule graph structure (expressed as a SMILES string) from molecular drawings, we investigate PaliGemma 2 for molecular structure recognition (Steiner et al., 2024). Following MolScribe, Steiner et al. (2024) employed one million molecules from the PubChem dataset as training data. Steiner et al. (2024) mentioned that these molecules are rendered using the Indigo toolkit and supplemented with a range of drawing styles and random perturbations. When employing 448px<sup>2</sup> resolution, PaliGemma 2 performs better than the state-of-the-art MolScribe; however, a higher precise match percentage was not achieved by further increasing the resolution (Steiner et al., 2024).

In order to convert photographs of single-line pianoform scores into their digital score representation in the `**kern` format<sup>1</sup>, we use PaliGemma 2 for optical music score identification (Steiner et al., 2024). Pitch and duration are encoded in the `**kern` representation, along with other typical score-related data like barlines and articulation (Steiner et al., 2024). Steiner et al. (2024) utilized the official train, validation, and test splits using the GrandStaff dataset [79], which has 53.7k images. Steiner et al. (2024) mentioned that both the original and artificially enhanced versions of the photographs are used during training. According to Steiner et al. (2024), the Line Error Rate (LER) is based on complete lines in the `**kern` encoding, the Character Error Rate (CER) counts errors at the character level, and the Symbol Error Rate (SER) measures errors at the symbol level (combining several characters).

According to Steiner et al. (2024), there are numerous applications for producing lengthy image captions with fine-grained information in multimodal learning, such as training text-to-image generation models with strong controllability. Steiner et al. (2024) refined the DOCCI (Descriptions of Connected and Contrasting Images) data set, which comprises 15,000 images with an average of 7.1 sentences (639 characters, 136 words) of comprehensive human-annotated English descriptions. The descriptions offer world knowledge, text rendering, object spatial relations, object counting, and more (Steiner et al., 2024). As a result, compared to many popular VLMs, which are frequently instruction-tuned on 10–100× bigger high-quality captioning sets than PaliGemma 2, the refined PaliGemma 2 model generates more factually aligned phrases as stated by Steiner et al. (2024). As expected, we find that improving both model size and resolution enhances factual alignment (Steiner et al., 2024).

These activities and the benchmarks that go along with them frequently rely on annotations that are generated by machines and are unaware of sophisticated failure modes, including negations (Steiner et al., 2024). Steiner et al. (2024) evaluated PaliGemma 2's spatial

reasoning capabilities using the Visual Spatial Reasoning (VSR) benchmark, which is intended to address these problems. Steiner et al. (2024) stated that it is designed as a classification job in which a model must ascertain the accuracy of a statement regarding the spatial relationships between items in the image. Steiner et al. (2024) framed this benchmark as a QA task with True/False responses in order to take advantage of PaliGemma 2's versatile text interface. PaliGemma 2 performs better than earlier fine-tuned models, and InstructBlip is also noticeably improved by fine-tuning (Steiner et al., 2024). While expanding beyond resolution 224 did not result in improvements, we do see notable advantages from a bigger model size, suggesting advantages from better language comprehension (Steiner et al., 2024).

Steiner et al. (2024) used PaliGemma 2 models to generate automatic chest X-ray reports, which may be viewed as a (long) captioning task on X-ray images, in order to investigate their potential in the medical field. Steiner et al. (2024) refined PaliGemma 2 using the MIMICCXR dataset, which includes free-text radiology reports and 377k images from 228k radiography studies at the Beth Israel Deaconess Medical Center in Boston, MA. Steiner et al. (2024) mentioned that the F1 score between the entities taken from the reference report and the one generated using RadGraph is what they call the RadGraph F1-score. RadGraph considers whether or not the findings are included in the report, along with how they relate to the attributes of the images (Steiner et al., 2024). A state-of-the-art RadGraph score is achieved using PaliGemma 2 (Steiner et al., 2024). Both resolution and model size increase result in slight gains (Steiner et al., 2024).

## **2.4 Review existing systems**

The existing systems to be reviewed will be Google Gemini, ChatGPT and DALL-E. The following subchapters will be reviewing the AI-powered chatbots' LLM architecture, capabilities and features.

### **2.4.1 Google Gemini**

Google Gemini, previously known as Google Bard, is a multimodal AI model developed by Google AI. Google Bard AI is founded on the Pathways Language Model 2 (PaLM 2), a large language model designed to be particularly effective at grasping facts, logical thinking, and arithmetic (Carlà et al., 2024). It is a generative AI chatbot with question answering capabilities designed to provide responses to prompts and queries. After being trained on a sizable corpus of text data, it can simulate human-like interactions to respond to a wide range of user queries and prompts, offering comprehensive and instructive responses to user inputs (Carlà et al., 2024).

Google Gemini is intriguing because of its multimodal capability which means it is capable of processing different types of input including audio, video, text and images. According to Rane et al. (2024), a more recent addition, Google's Gemini, is a multimodal AI model with even more capabilities, including image processing, language processing, and possibly other modalities. One of the more interesting features are the vision-language question answering that includes image uploading and text prompts where user can upload an image then type a prompt in the textbox, the chatbot will then generate a text response in natural human language based on the image uploaded. Additionally, Google Gemini works similarly to ChatGPT in handling traditional text-based question answering where users can prompt using text and the chatbot will generate a text response.

Developed by Gemini Team in Google, the Gemini 1.5 family succeeds from its predecessor, Gemini 1.0, adopting similar architecture with major tweaks in its multimodal capability. The free version of Google Gemini – the AI chatbot, utilizes the Gemini 1.5 Flash model. According to Team et al. (2024), Gemini 1.5 Flash's architecture is designed for effective use of tensor processing units (TPUs) with reduced latency for model serving, it is a

transformer decoder model that shares the same 2M+ context and multimodal features as Gemini 1.5 Pro. This model is online distilled from the considerably bigger Gemini 1.5 Pro model and performs concurrent computation of attention and feedforward components (Team et al., 2024). As stated by Team et al. (2024), Gemini 1.5 Flash belongs to the Gemini 1.5 family, a new family of extremely powerful multimodal models that pushes the limits of efficiency, reasoning, planning, multilinguality, function calling, and long-context performance by incorporating our most recent developments in sparse and dense scaling along with significant advancements in training, distillation, and serving infrastructure.

Due to the public accessibility of Google Gemini, the system review will examine its features and capabilities, following the format of PaliGemma's review. As a generative AI chatbot, Google Gemini incorporates common features such as login module, media uploads, textbox prompt, text-to-speech (TTS) function, text generation, conversation history, model switch module, sharing module and user feedback module. A screenshot of the system's user interface (UI) is shown in Figure 2.14.

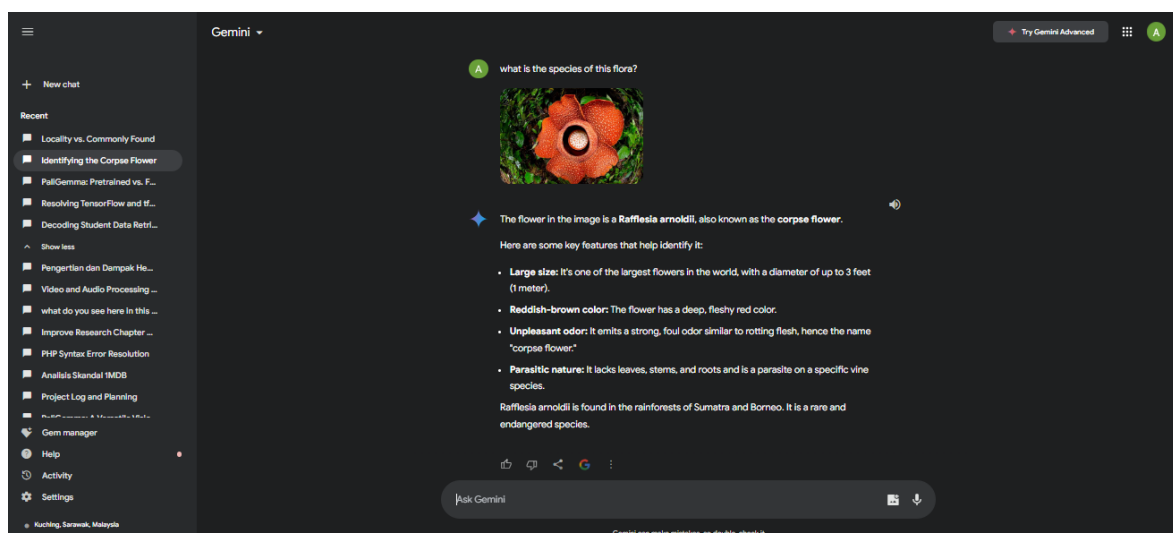


Figure 2.14: Preview of Google Gemini's user interface.

As shown in Figure 2.14, Google Gemini's textbox prompt is located on the bottom of the page which includes the placeholder "Ask Gemini", a media upload icon and a TTS button on the rightmost of the textbox prompt. A conversation with the system is the highlight of the page. The conversation simulation is conducted similarly to PaliGemma's system review, with the same image and prompt. However, Google Gemini generates a more extensive response in identifying the species, describing its characteristics and specifying its geographical location compared to Figure 2.2. Below the generated response, a user feedback module allows users to express satisfaction or dissatisfaction (thumbs-up/down icon) towards the response, a share icon, a Google icon and a three-dots menu to explore further options.

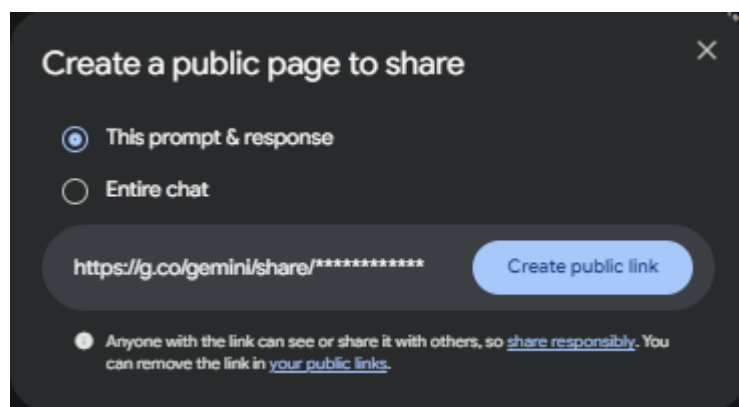


*Figure 2.15: Pop-up box of sharing module when clicking share icon.*

When the share icon is clicked, a pop-up box appears as shown in Figure 2.15. Users can choose to either share a specific prompt and response or the entire conversation. A public link is then generated for sharing.

Other than the response generated by Google Gemini, the UI displays the common features mentioned as shown in Figure 2.14. On the left side of the page, a conversation history panel displays all the previous conversations that a user had with Google Gemini including

providing access to “Help”, “Activity” and “Settings” module. A model switch is located at the top of the page right next to the conversation history panel, allowing users to switch between different models of Gemini. The user profile in the top-right corner indicates that a login module is included.

These features will be compared to ChatGPT and DALL-E, which will be reviewed in the following subchapters.

### **2.4.2 ChatGPT**

ChatGPT is developed by OpenAI and according to Wu et al. (2023), the public release of ChatGPT by OpenAI at the end of 2022 further garnered international notice for its flawless response to any human request expressed in natural language. ChatGPT is widely known and used by users from diverse backgrounds and fields. It is a generative AI chatbot that uses natural language processing to respond to user prompts in human-like language much like Google Gemini. Natural language processing (NLP) is a branch in AI that focuses on enabling computers to understand and generate human language. In order to communicate with computers in both written and spoken environments using natural human languages rather than computer languages, natural language processing (NLP) is a subfield of artificial intelligence that studies, comprehends, and produces the languages that humans naturally use (Kounte et al., 2020). According to the official announcement, ChatGPT has demonstrated powerful capabilities on a range of language understanding and generating activities as an AIGC member, including multilingual machine translation, code debugging, story authoring, acknowledging errors, and even turning down inappropriate requests (Wu et al., 2023).

ChatGPT’s present architecture utilizes GPT-4 from the GPT-3.5 series. According to Wu et al. (2023), the original GPT (Generative pre-trained Transformer) model, which was

iteratively upgraded from GPT-1 to GPT-4, serves as ChatGPT's foundation. Although GPT-4's technical specifications (hardware, architecture, dataset construction, training process, etc.) have not been made public, the primary methods of GPT-4 remain comparable to those of GPT-3/GPT-3.5 (Wu et al., 2023). With the limitation of ChatGPT's architecture information, Wu et al. (2023) mentioned that the official claims that its implementation is similar to InstructGPT, a sibling model to ChatGPT that differs from it in terms of data gathering and the pre-trained backbone, but the technical details of ChatGPT and GPT-4 have not been disclosed. According to Ouyang et al. (2022), the sole difference between both models, which have the same architecture, is that InstructGPT is optimized using our human data. To conclude, ChatGPT is a language model that is based on Transformer architecture.

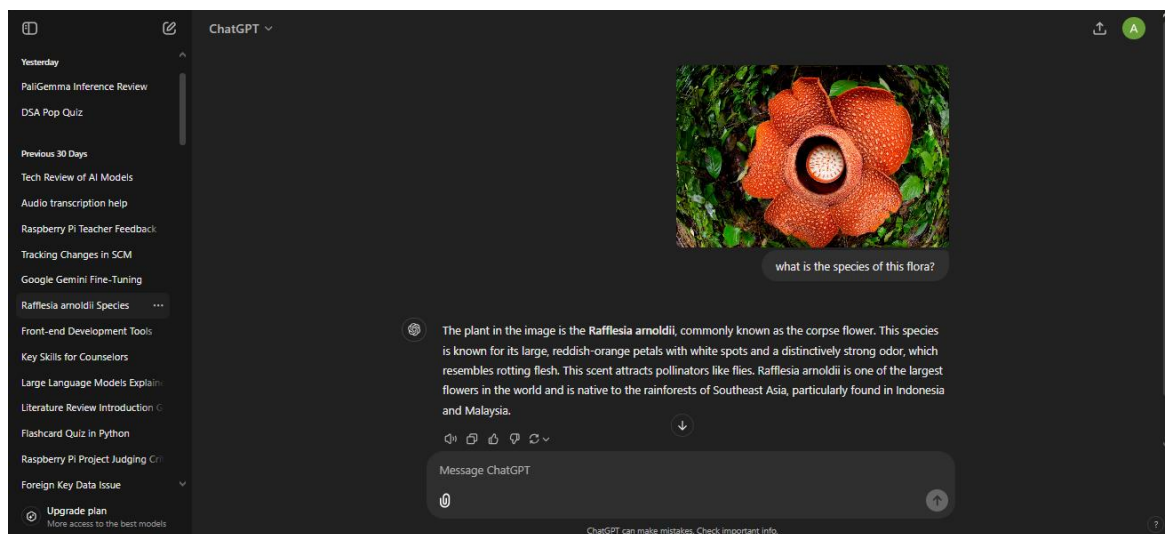


*Figure 2.16: ChatGPT's user interface.*

As illustrated in Figure 2.16, the UI is similar to Google Gemini in Figure 2.14 with standard features such as a login module, media upload, textbox prompt, text generation, conversation history panel, model switch module, sharing module and user feedback module.

As shown in Figure 2.16, ChatGPT's textbox prompt is positioned at the bottom of the page with a placeholder text labelled "Message ChatGPT". To the bottom-left corner of the textbox is a media uploads icon, allowing users to attach files. On the left of the page, the conversation history panel displays a list of prior conversations with ChatGPT.

At the top-right corner, the user profile indicates a login module and next to it is the "Share" icon, enabling users to share the conversation. The central section of the UI highlights the conversation with ChatGPT, displaying the user's prompts and the response generated by ChatGPT. Below each response by ChatGPT, there are five icons namely the "Read Aloud" icon, "Copy" icon, user feedback module represented by "Thumbs Up/Down" icon and a model switch icon, each of these icons have different functionality.

1. "Read Aloud" icon when clicked will read the generated response aloud to users, helping users with vision problem to instead listen to the response.
2. "Copy" icon when clicked will copy the generated response into user's clipboard.
3. The user feedback module's "Thumbs Up/Down" icon represents satisfaction or dissatisfaction towards the generated response.
4. A model switch icon when clicked allows users to switch from different models depending on whether the users paid for an extra or premium model of ChatGPT.

Additionally, another model switch module is located at the top of the page, to the right of the conversation history panel as shown in Figure 2.16.

These features will be collected for a comparative analysis with Google Gemini and Dall-E.

### **2.4.3 DALL-E 3**

DALL-E is an image generation AI model developed by OpenAI, the developers of ChatGPT. It has several iterations, starting with DALL-E, then DALL-E 2, and now DALL-E 3. DALL-E 3 is an artificial intelligence system that creates a new image as an output after receiving a text prompt (OpenAI, 2023b). DALL-E 3 improves upon its predecessors by providing better alignment with user prompts and generating more realistic and creative images as outputs. The latest DALL-E version, DALL-E 3, employs techniques such as improved synthetic captioning, where 95% of the training data comprises of captions generated by advanced AI captioners and 5% includes ground-truth captions (Betker et al., n.d.). This combination enhances the model's ability to follow complex prompts effectively.

Architecturally, DALL-E relies on a transformer-based decoder-only model that combines text and image data in a single stream (Ramesh et al., 2021). Like earlier versions, it uses technologies such as CLIP for improved text-image alignment. According to Ramesh et al. (n.d.), DALL-E 2 is trained on a combination of two components, a prior and a decoder, and given their CLIP image embeddings, the decoder enables us to invert images, and the prior enables us to learn a generative model of the image embeddings. Despite its advancements, limitations include occasional errors in spatial reasoning and generating text within images. According to Betker et al. (n.d.), even though DALL-E 3 represents a major advancement in prompt following, it still has issues with spatial awareness and item placement.

Currently, DALL-E 3 is accessible through OpenAI's upgraded ChatGPT Plus subscription which cost \$20 a month, but OpenAI also provides free limited-use credits for users accessing the model independently. OpenArt is a website which allows users to independently access DALL-E 3.

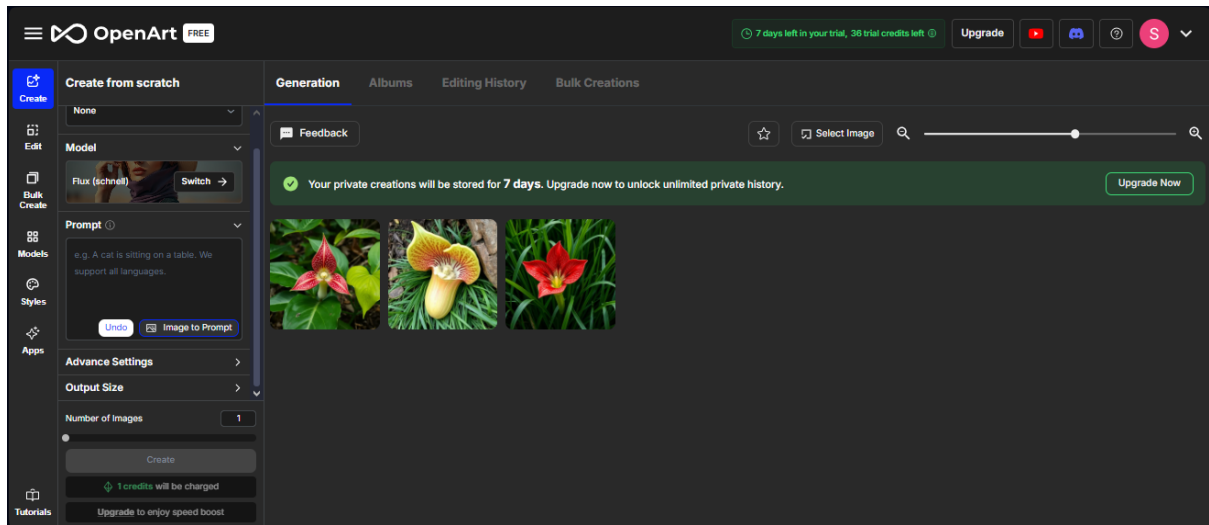


*Figure 2.17: OpenArt website home page with limited access to DALL-E 3*

Based on Figure 2.17, the home page displays standard features which includes a login module, media uploads functionality, a textbox for prompts, text generation capabilities, image generation capabilities, conversation history panel, a sharing module, a model switch feature and a user feedback module. A user profile icon is located in the top-right corner of the page, indicating the presence of a login module for user authentication. While DALL-E 3 is primarily an image generation AI, the UI also includes media upload functionality which allows users to upload an image then DALL-E 3 will generate a text output based on the image uploaded. A textbox prompt is positioned on the left side of the page, enabling users to input a text prompt. Despite these features, DALL-E 3 does not include a text-to-speech (TTS) function or read aloud functionality unlike Google Gemini and ChatGPT.

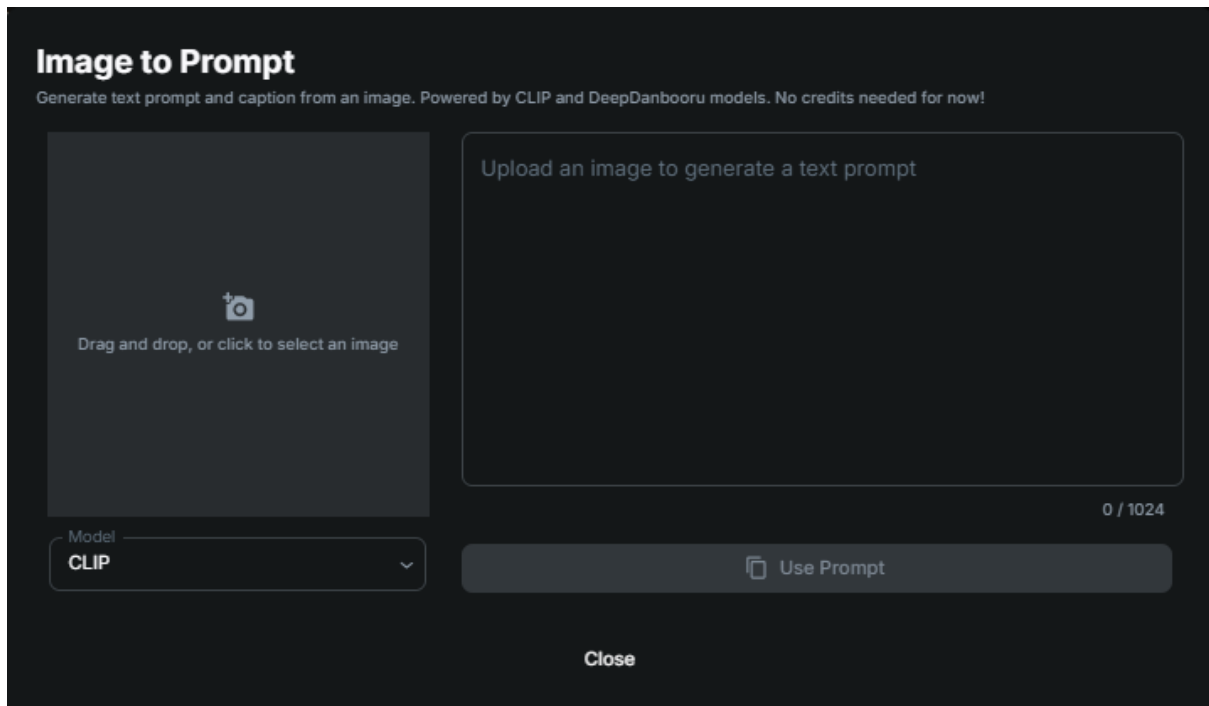


*Figure 2.18: Image to Prompt UI.*

The media upload functionality in DALL-E 3 generates texts as output based on the uploaded image as input. The UI of this functionality is shown in Figure 2.18. The UI shows a section for users to upload image by either dragging and dropping the image into the upload image section or clicking the section to select an image from their file explorer. Once an image is uploaded, the text will be generated based on the image uploaded.

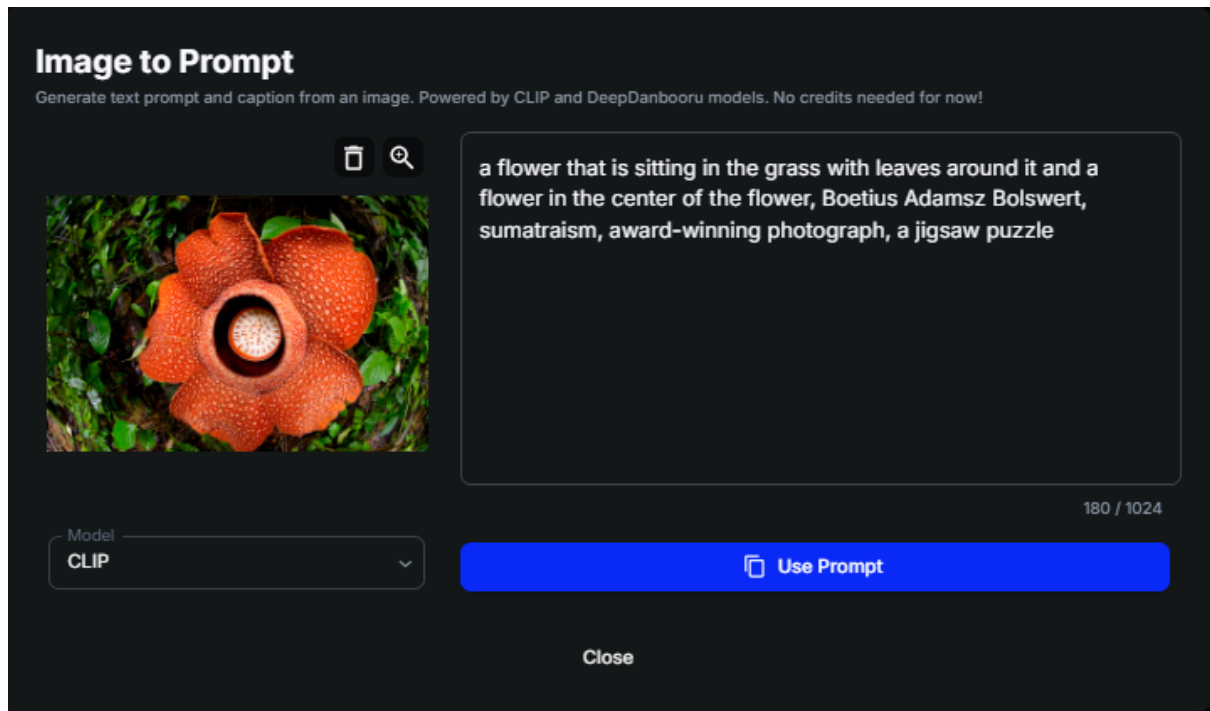


*Figure 2.19: Image to Prompt (text generation).*

The media upload functionality in DALL-E 3 generates texts as output based on the uploaded image as input. The UI of this functionality is shown in Figure 2.19. The text generation result is not as expected since DALL-E 3 is an image generation AI, it does not have the capability to identify objects in images. Hence, the result of the text generation which does not match the image uploaded. This shows limitations of DALL-E compared to PaliGemma, Google Gemini and ChatGPT in terms of image captioning capability.

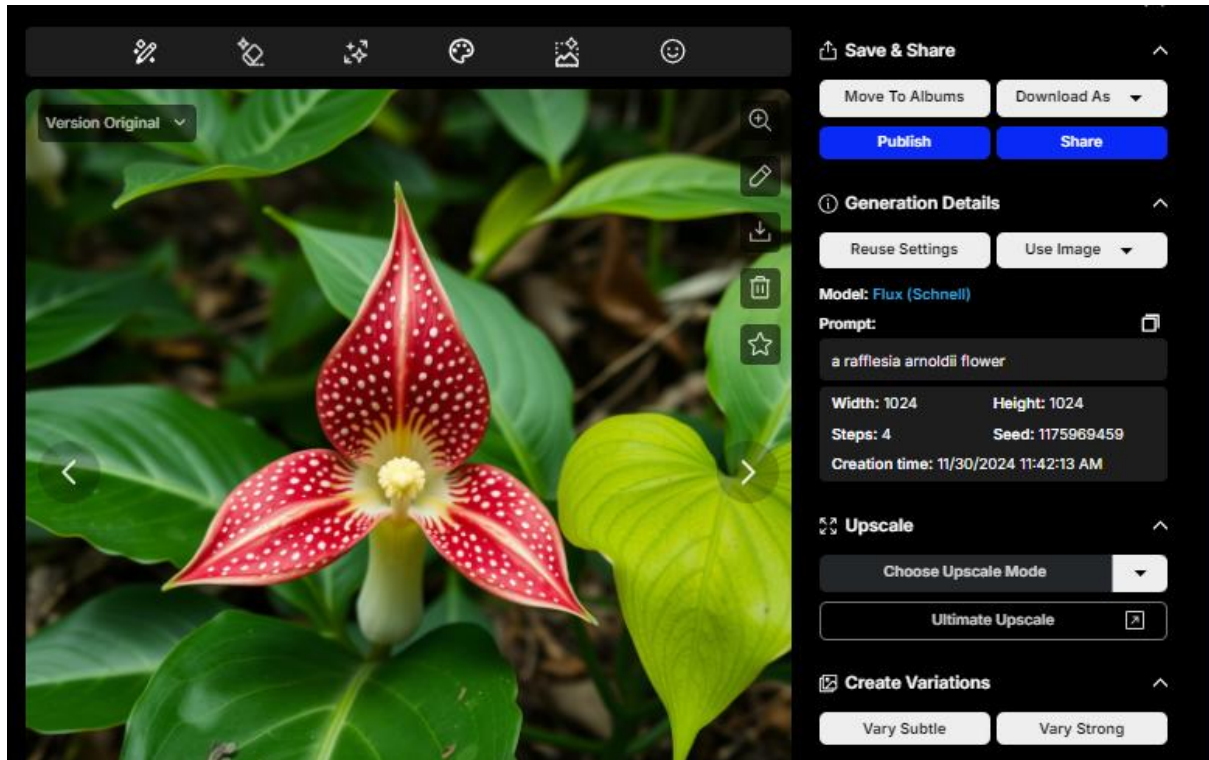


*Figure 2.20: Image generation from DALL-E 3 model.*

Differing from the text generation capability, a demonstration of the intended use of DALL-E 3 is shown in Figure 2.20, where user inputs a text prompt and the system outputs an image generation as a result from the input. Based on Figure 2.20, the user prompted “a rafflesia arnoldii flower” and the generated image does not match the prompt where it shows a flower which is far different from what is described in the textbox prompt. As mentioned previously, DALL-E 3 does not have the capability to identify flora species as it is an image generation AI.

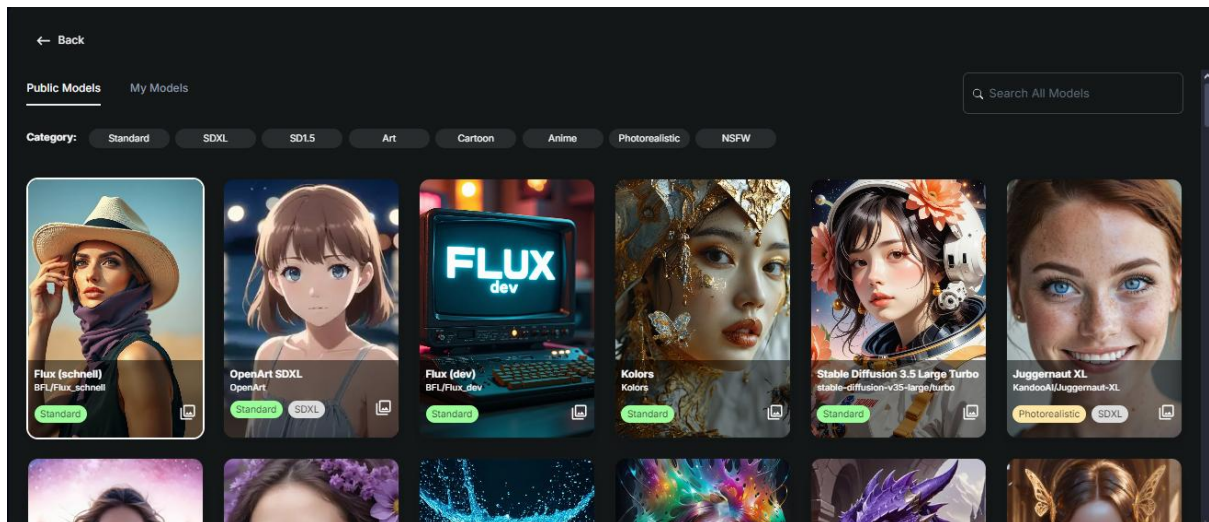


Figure 2.21: Model switch selection for DALL-E 3.

The model switch functionality in DALL-E 3 differs from systems like Google Gemini and ChatGPT. Instead of the traditional toggling between predefined models, users can select from models created and shared by other users within the platform. A wide range of models' selection is shown in Figure 2.21.

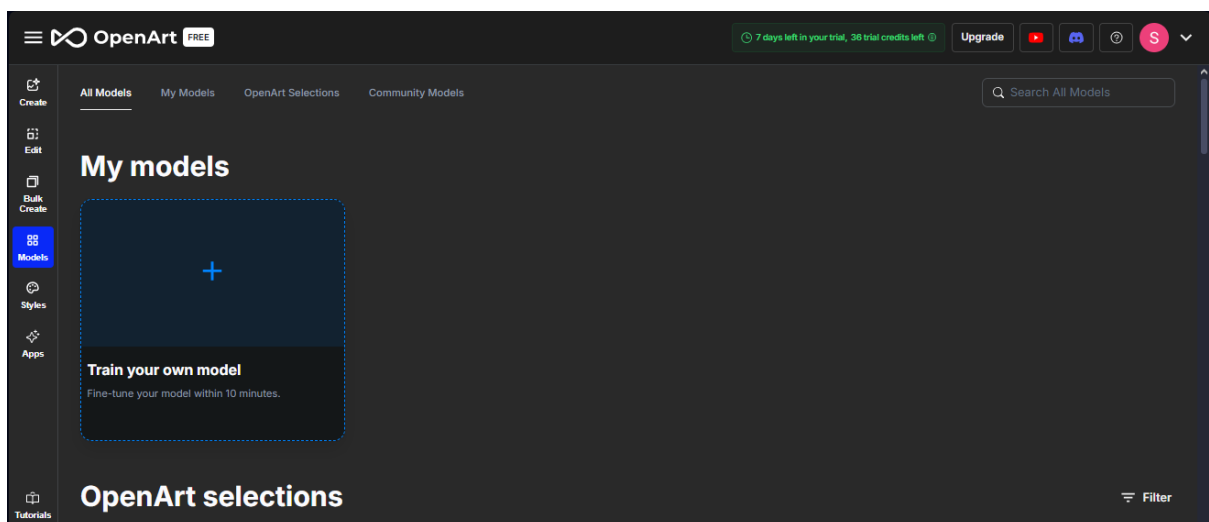


Figure 2.22: Models fine-tune section for DALL-E 3.

Additionally, users have the option to train and fine-tune their own model within 10 minutes. To access this feature, users can navigate through the UI via the three-bar menu on the top-left corner of the page then to the “Models” section as shown in Figure 2.22.

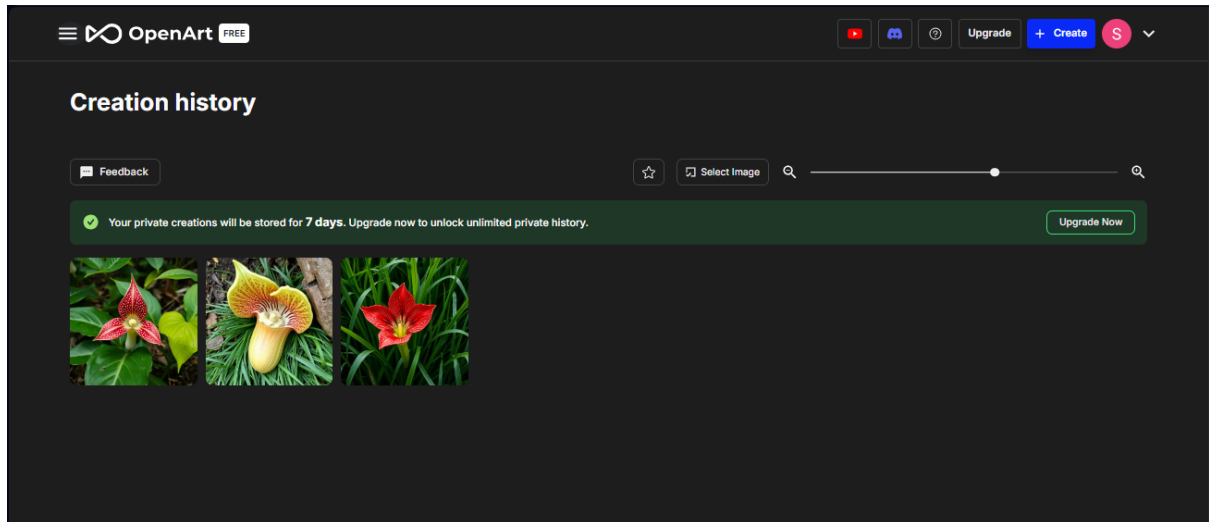


*Figure 2.23: Creation history for DALL-E 3.*

In DALL-E 3, instead of a conversation history panel, it is described as “Creation History” where users can view their previous image generation activities as shown in Figure 2.23. This section allows users to revisit and manage their creations efficiently.

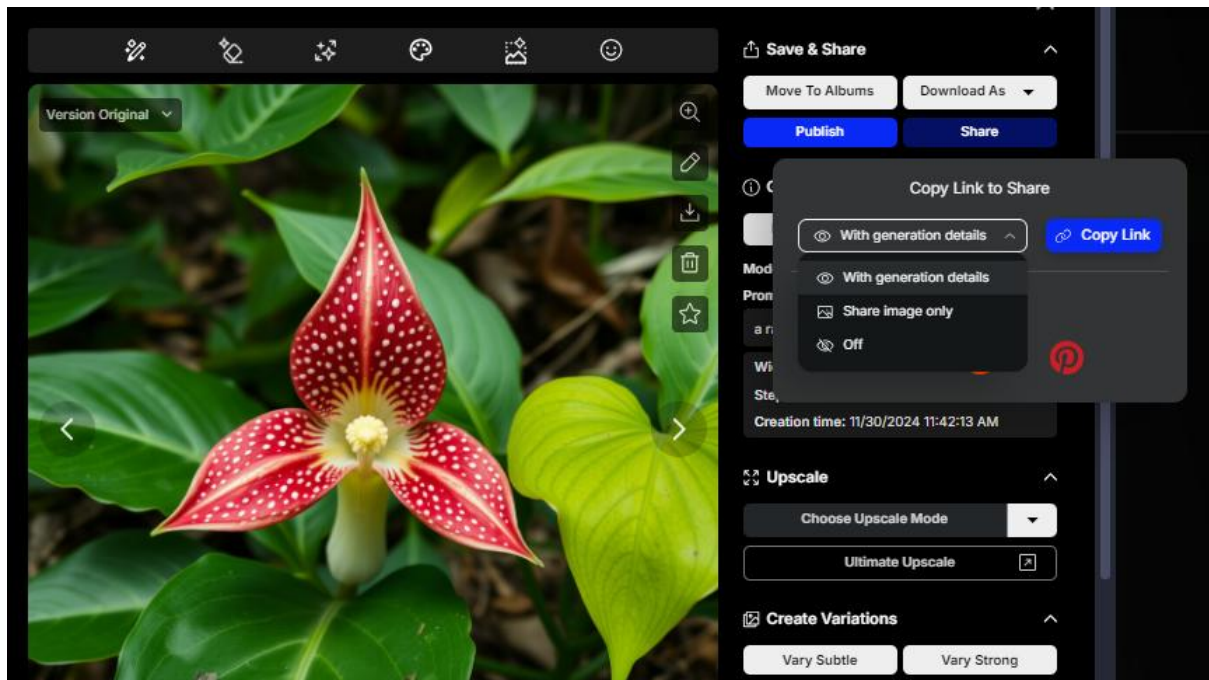


*Figure 2.24: Sharing module for DALL-E 3.*

Based on Figure 2.24, the sharing module in DALL-E 3 provides a selection of sharing options. Users can choose to either share an image along with the generation details or to share only the image. Additionally, the system generates a shareable link that can be copied and shared with other users.

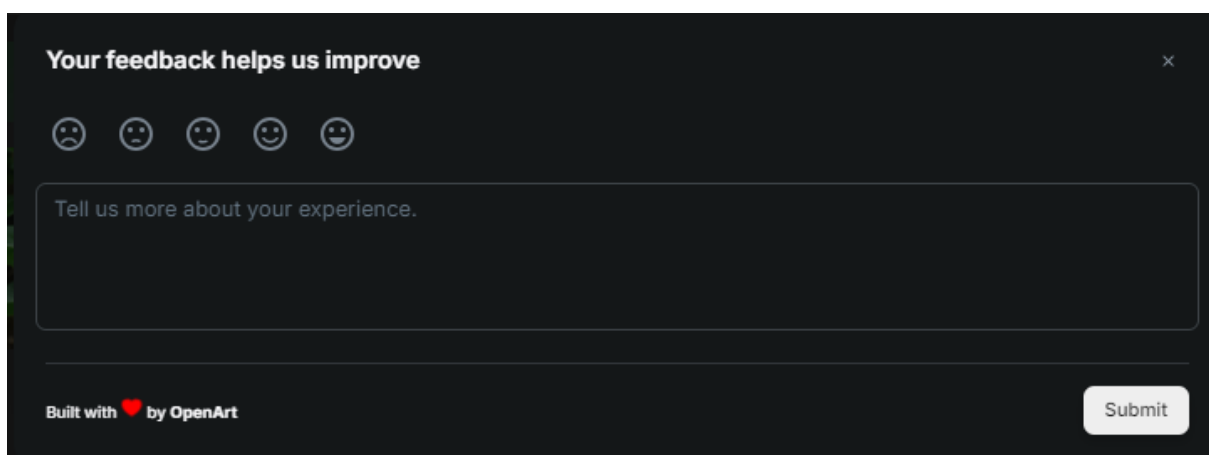


*Figure 2.25: User feedback module for DALL-E 3.*

Based on Figure 2.25, the user feedback module in DALL-E 3 differs from the user feedback module in Google Gemini and ChatGPT. It is simply a feedback form with a set of emoji icons. These icons allow users to express their satisfaction or dissatisfaction with the system's performance and the feedback form allows users to comment on the system's performance as well as providing comments on improvements suggestions.

#### 2.4.4 Comparison between the Existing Systems

| Features \ Systems            | Google Gemini | ChatGPT | DALL-E 3 |
|-------------------------------|---------------|---------|----------|
| Textbox prompt                | ✓             | ✓       | ✓        |
| Image upload                  | ✓             | ✓       | ✓        |
| Login                         | ✓             | ✓       | ✓        |
| Conversation history          | ✓             | ✓       | ✓        |
| Read aloud/listen to response | ✓             | ✓       | X        |
| User rating (thumbs up/down)  | ✓             | ✓       | X        |
| Feedback form                 | X             | X       | ✓        |
| Text generation capabilities  | ✓             | ✓       | ✓        |
| Image generation capabilities | X             | ✓       | ✓        |
| Model switch                  | ✓             | ✓       | ✓        |

|                       |   |   |   |
|-----------------------|---|---|---|
| New conversation/chat | ✓ | ✓ | X |
| Chat sharing/export   | ✓ | ✓ | ✓ |
| Image sharing         | X | X | ✓ |
| Settings              | ✓ | ✓ | ✓ |
| Chat customization    | ✓ | ✓ | X |
| Help section          | ✓ | ✓ | ✓ |
| Report problem        | ✓ | ✓ | X |

*Table 2.6: Comparative analysis review of three (3) existing system.*

All of the existing systems are collectively question answering system with both Google Gemini and ChatGPT being similar while DALL-E generates images from user prompt, question answering with image as the response instead of text as response.

Table 2.6 shows the features checklist across the three existing systems. While some of the features are similar, the methods are different. For example, in both Gemini and ChatGPT, the user feedback is collected through the option of a thumbs up or thumbs down for each response generated by the chatbot, however, in DALL-E, instead of thumbs up or thumbs down, it has a feedback form for users to rate their experience with DALL-E generally.

Other than that, the conversation history of DALL-E, described as “Creation history”, is different compared to Google Gemini and ChatGPT. Since it is an image generation AI, it does not have “conversations” with the users, instead it has users’ creation history where it stores users’ previous image generation. Due to DALL-E 3’s inaccessibility to the public, the

website OpenArt provides users with a preview to DALL-E 3 although the access is limited to free-use credits and a 7-day free trial.

As described in table 2.6, all three systems have the capability to generate text as output however, in terms of the capability to generate image as output, only Google Gemini are incapable of doing so. Notably, ChatGPT integrates DALL-E 3 in its premium subscription – ChatGPT Plus, for image generation capabilities but it requires paid subscription which means not every member of the public can access it for free.

Despite all these differences, there is one more feature that all three systems have but the execution methods are different. For instance, the model switch feature. In both Google Gemini and ChatGPT, the model switch feature allows users to toggle between predefined models and subscription models. However, in DALL-E 3, the model switch feature is not the traditional models switch. Its model switch feature allows users to switch between models that were trained and fine-tuned by other users of the platform, with different themes and styles for image generation.

The chat customization feature between all three systems varies even though the features are the same. In Google Gemini, the chat customization allows users to switch from light theme to dark theme. However, in ChatGPT, the chat customization allows users to customize the response generated by the system. DALL-E 3 does not have the chat customization feature.

Each system has unique features tailored to specific use case. Google Gemini has the advantage of being a straightforward text-based assistance, ChatGPT has balanced multimodal capabilities with the integration of DALL-E 3, and DALL-E 3 is advantageous for advanced image generation. However, Google Gemini has no image generation capabilities, ChatGPT

and DALL-E 3 have accessibility constraints which are disadvantageous for the general public. Thus, these advantages and disadvantages must be weighed to determine the suitability of specific applications.

Although the three reviewed systems are QA based system, only Google Gemini and ChatGPT are identical in model architecture and system design whereas DALL-E generates images based on user prompts and differ mainly on the system design as well as the system structure.

## **2.5 Review on Tools and Technology for Web Applications**

### **2.5.1 Front-end Development**

The front-end development for this system will utilize Figma for designing prototypes. For the development itself, a full-stack Python framework like Django will be employed to build and manage the front-end components, facilitating smooth integration with backend functionalities and enabling dynamic content delivery.

### **2.5.2 Back-end Development**

The backend development for this system will be implementing Python language and utilizing Django, a full-stack Python framework that provides a set of tools for building web applications. With Django's built-in ORM (Object-Relational Mapping), the system will efficiently manage and query the database, and an authentication system for secure user management. The database will be implemented using MySQL and XAMPP.

### **2.5.3 API Integration Tool (PaliGemma)**

PaliGemma is an open-source model which will be integrated into the web application through an API interface. Since the selected system development language will be Python, the

development process will utilize Django, a full-stack Python framework. By utilizing Django's requests library to handle the API requests, the web application will be able to send data to the PaliGemma API, receive the model's processed responses and integrate the responses directly into the web application's functionalities.

#### **2.5.4 Database Integration**

The system will integrate MySQL as the primary database and handles all aspects of user management. MySQL will be responsible for storing, retrieving and managing user account within the web application. This database will be managing key functionalities of the web application like user authentication which includes new user registration and existing user login.

### **2.6 Proposed System**

The proposed system's features will be discussed further in Chapter 3, focusing on the system's core functionalities and user interface design. The features to be determined will be inspired by the three existing systems reviewed in this chapter.

The proposed system's model architecture is significantly differs from the reviewed systems. However, they all belong to the same field - question answering (QA), incorporating natural language processing (NLP). Due to PaliGemma's nature as a visual language model(VLM), it lacks the capability for back-to-back conversational interactions with users. Each prompt can only be processed once and the system does not support continuous dialogue.

The review of PaliGemma 2 is also included in this literature review, with considerations for incorporating it into the proposed system. By including a model switch feature into the web application, users will be able to toggle between PaliGemma and

PaliGemma 2. However, the execution of this feature will be further assessed until there is a feasible way to integrate both models into the web application. It is worth noting that the integration of both models is currently a conceptual idea under exploration.

## **2.7 Summary**

Chapter 2 discussed the comparison of three existing systems by comparing the features and system architectures of Google Gemini, ChatGPT and DALL-E. A comparative analysis of the three systems were also discussed and concluded. Since PaliGemma will be integrated with the proposed system, it is crucial to review PaliGemma as well by mainly covering the use cases, architectures, existing project review, and research paper review. By understanding the system's core and structure, it will be easier to implement it into the proposed system.

## **CHAPTER 3: REQUIREMENTS ANALYSIS AND DESIGN**

### **3.1 Introduction**

In this chapter, the chosen methodology will be summarized and each phase of the proposed methodology will be explained in detail including the tools to be utilized. The software requirements will describe the software development tools necessary for developing the proposed system, along with additional tools which were not mentioned in Chapter 2. The hardware requirements will specify the hardware architecture needed to support the development of the proposed system. Lastly, the system design will illustrate the structure of the proposed system through the implementation of UML diagrams/ Data Flow Diagrams (DFD)/Entity Relationship Diagram (ERD).

### **3.2 Methodology summary**

This project will be implementing the Modified Waterfall Model with Iteration which consists of five phases namely, Requirements Definition, System and software design, Implementation and unit testing, Integration and system testing and Operation and maintenance.

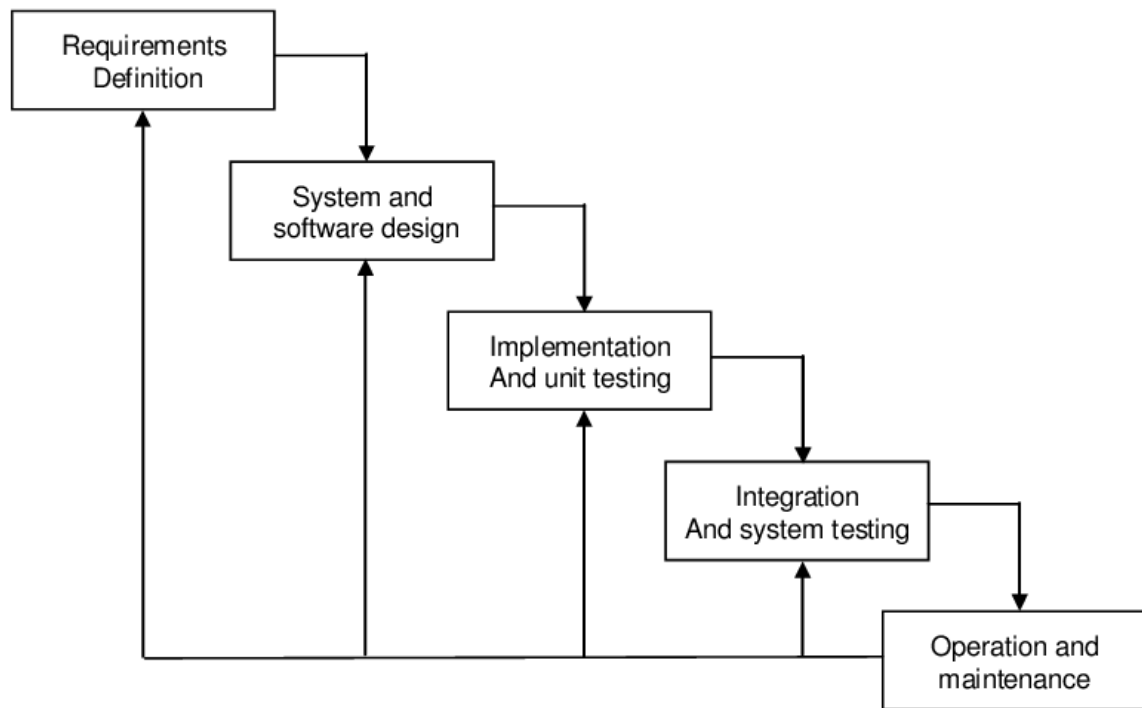


*Figure 3.1: Modified Waterfall Model with Iteration between phases*

The requirement definition phase will be involving a detailed analysis to gather system requirements such as species identification, educational content and PaliGemma integration for visual recognition and image captioning tasks. This phase will analyse what the proposed system needs to proceed into the development phase. The elicitation technique used to gather requirements is by distributing questionnaire. This will help in identifying requirements needed for the system.

The system and software design phase will be designing the proposed system's interface using a software design tool, Figma, and also illustrating the flow of the web application using UML diagrams by utilizing a web-based diagramming application, LucidChart.

The proposed system development's process will be done in the implementation and unit testing phase. The development will include building the frontend, backend and

implementing PaliGemma into the web application. The frontend and backend will be developed using Django, a Python framework and setting up the database for user authentication in the web application while PaliGemma will be implemented into the web application using a remote API. After completing all the implementations, a unit testing will be done to test the key features of the web application.

Integration of the frontend, backend and PaliGemma as one system will be done in the integration and system testing phase. The system testing will validate the functionality, usability and performance of the system.

Finally, the deployment of the web application will be done in the operation and maintenance phase. By deploying the developed web application on a hosting server to ease online accessibility for a wide range of users. Maintenance of the system will also be done in this phase based on user feedback and experience with the web application.

Additionally, the reason modified waterfall model with iteration was chosen is to allow iteration between phases to ease the process of any changes.

### **3.3 System requirements**

#### **3.3.1 Software requirements**

The software required for the development of this project will be Visual Studio Code, XAMPP, MySQL, Google Colab, Microsoft Azure, Python, Django framework and HuggingFaceAPI.

| Software Requirements | Description |
|-----------------------|-------------|
|-----------------------|-------------|

|                    |                                                                                                                                                                                                                                                                                                                                          |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Visual Studio Code | Visual Studio Code (VSCode) is a free, open-source code editor that will be used for code development in this project.                                                                                                                                                                                                                   |
| XAMPP              | XAMPP is a free, open-source cross-platform web server solution stack package developed by Apache Friends, consisting mainly of the Apache HTTP Server, MariaDB database and interpreters for PHP and Perl programming languages. This solution allows for local testing of the developed web application before hosting it on a server. |
| MySQL              | MySQL will be the relational database management system used to handle the backend of the proposed system.                                                                                                                                                                                                                               |
| Google Colab       | Google Colab is a free, browser-based tool that allows users to write and execute Python code. In this project's case, Google Colab will be used to test the output of the model based on a given image and compare it with the model applied inside the web application.                                                                |
| Microsoft Azure    | Microsoft Azure will be used to host the AI model through an inference endpoint.                                                                                                                                                                                                                                                         |

|                                         |                                                                                                               |
|-----------------------------------------|---------------------------------------------------------------------------------------------------------------|
| Python environment                      | Python will be the main programming language used to develop the proposed project.                            |
| Django                                  | A Python framework that will be used to develop the frontend and backend of the proposed project.             |
| Application Programming Interface (API) | A HuggingFace API will be used to access the PaliGemma without having to install it locally into the machine. |

*Table 3.1: Software requirements.*

### 3.3.2 Hardware requirements

First of all, the attempt to apply PaliGemma, a vision language model into a web application has its challenges mainly because it is a free model provided by Google. Even though it is a free model, the trial and error phase of trying to execute it on a local machine poses a huge risks due to its large size. In this subsection, the minimum hardware required to execute PaliGemma locally will be thoroughly explained in Table 3.2.

| Hardware Requirements         | Description                                                    |
|-------------------------------|----------------------------------------------------------------|
| Operating System              | Windows 10/11<br><br>macOS<br><br>Linux (Ubuntu 20.4 or newer) |
| Central Processing Unit (CPU) | Intel Core i5 13 <sup>th</sup> Gen or similar                  |
| Memory (RAM)                  | Minimum 16GB RAM                                               |

|                                |                                        |
|--------------------------------|----------------------------------------|
| Graphics Processing Unit (GPU) | Minimum 16GB VRAM (NVIDIA RTX 4080)    |
| Storage                        | Minimum 512 GB Solid State Drive (SSD) |

*Table 3.2: Hardware requirements.*

### **3.4 Requirement Elicitation & Analysis**

The requirement elicitation technique selected for the proposed system is a questionnaire. The questionnaire was distributed to 45 respondents, consisting of individuals from Universiti Malaysia Sarawak (UNIMAS) as well as those from non-UNIMAS backgrounds. The questionnaire contains 3 sections and 16 questions, with the first section being the demographics collection of respondents. The second and third section of this questionnaire are questions inquiring respondents of their familiarity with AI chatbot and inquiring respondents' opinion and suggestion of a web application chatbot features, respectively. In this requirement elicitation and analysis, each questions' responses will be analyzed in the forms of a pie chart, bar chart and horizontal bar chart.

The second section of the questionnaire is a data collection of respondents' familiarity with AI chatbot. This section consists of 5 questions and each will be analyzed to understand respondents' familiarity of AI chatbot.

### 3.4.1 Questionnaire Response Analysis

Are you familiar with AI chatbot such as ChatGPT, Google Gemini and other similar system?

 [Copy chart](#)

45 responses

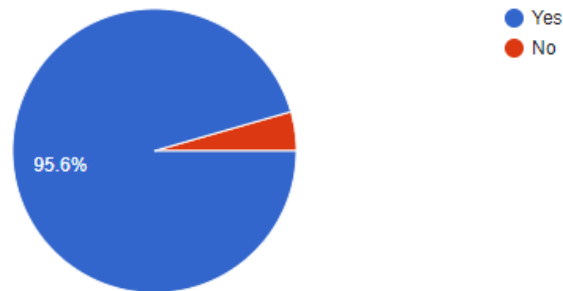


*Figure 3.2: Are you familiar with AI chatbot such as ChatGPT, Google Gemini and other similar system?*

According to Figure 3.2, 95.6 percent of respondents (43 respondents) responded 'Yes' indicating that they are familiar with AI chatbots. Meanwhile, 4.4 percent of (2 respondents) responded 'No', indicating that they are not familiar with AI chatbots.

Have you used the services of AI chatbot?

 [Copy chart](#)

45 responses

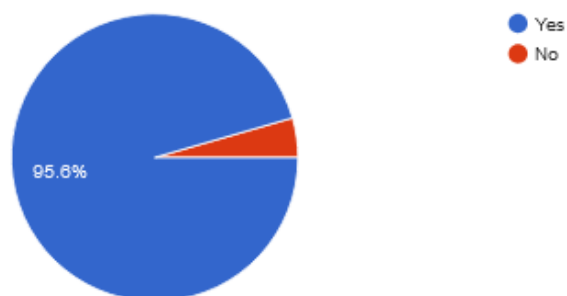


*Figure 3.3: Have you used the services of AI chatbot?*

According to Figure 3.3, 95.6 percent of respondents (43 respondents) responded ‘Yes’, showing that they have ever used AI chatbot service. Meanwhile, 4.4 percent of (2 respondents) responded ‘No’, showing that they have never used any AI chatbot service.

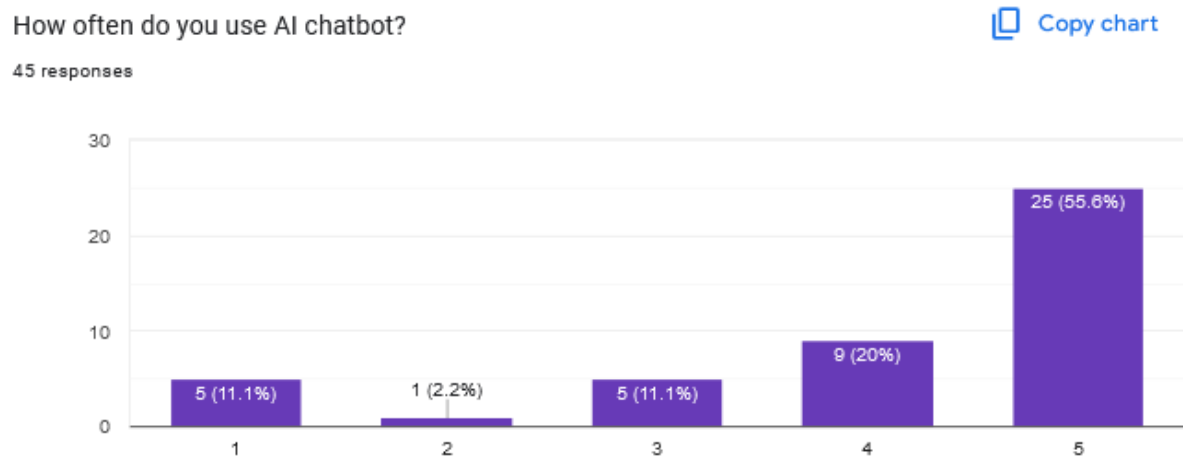


*Figure 3.4: How often do you use AI chatbot? From 1 (seldom) to 5 (very often).*

According to Figure 3.4, 25 respondents chose the score 5, which indicates that respondents very often use AI chatbot., 9 respondents chose the score 4, which indicates that respondents often use AI chatbot, 5 respondents chose the score 3, which indicates that they use AI chatbot in average, 1 respondent chose the score 2, which indicates that they less often use AI chatbot, and 5 respondents chose the score 1, which indicates that they seldom use AI chatbot.

What do you usually use the AI chatbot for?

 [Copy chart](#)

45 responses

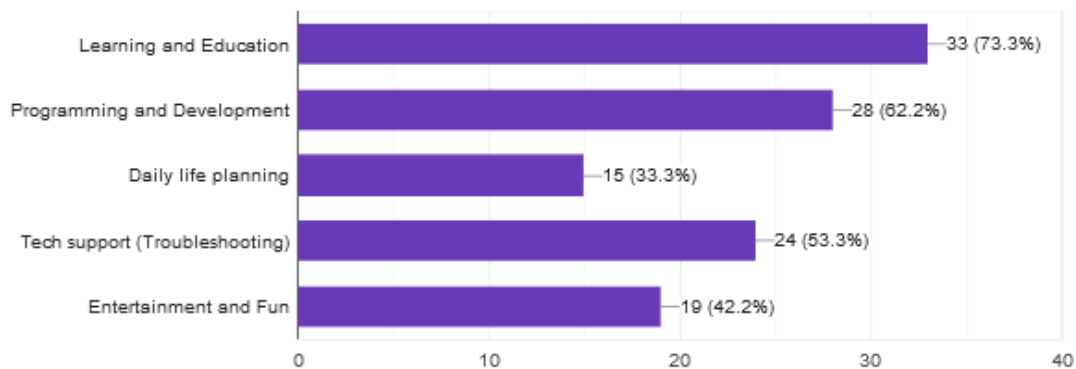


*Figure 3.5: What do you usually use the AI chatbot for?*

According to Figure 3.5, 33 respondents use AI chatbot for learning and education, 28 respondents use AI chatbot for programming and development, 15 respondents use AI chatbot for daily life planning, 24 respondents use AI chatbot for tech support or troubleshooting, and 19 respondents use AI chatbot for entertainment and fun.

Would you consider using AI chat for the species identification of nature and wildlife?

 [Copy chart](#)

45 responses

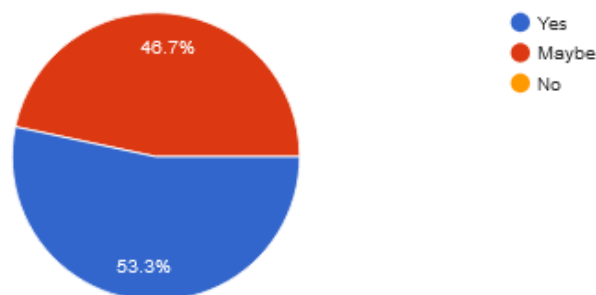


*Figure 3.6: Would you consider using AI chat for the species identification of nature and wildlife?*

According to Figure 3.6, 53.3% of the respondents (24 respondents) responded ‘Yes’, indicating that they consider using AI chatbot for the species identification of nature and wildlife, while the remaining 46.7% of the respondents (21 respondents) responded ‘Maybe’, indicating that they are unsure about it.

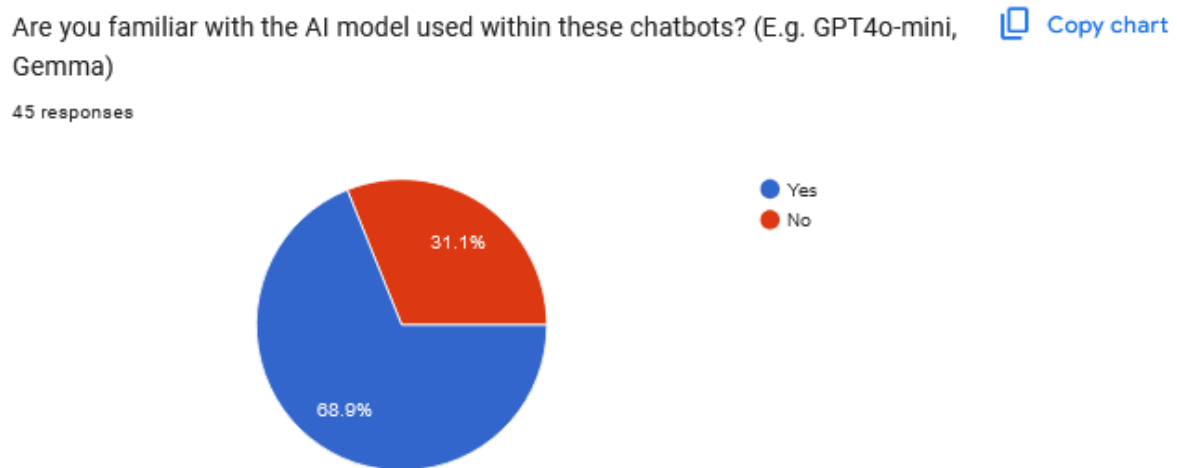


*Figure 3.7: Are you familiar with the AI model used within these chatbots? (E.g. GPT4o-mini, Gemma)*

According to Figure 3.7, 68.9% of the respondents (31 respondents) responded ‘Yes’, indicating that they are familiar with the AI model used in the chatbot. 31.1% of the respondents (14 respondents) responded ‘No’, indicating that they are not familiar with AI model used in the chatbot.

Do you know of PaliGemma as a vision-language model?

 [Copy chart](#)

45 responses

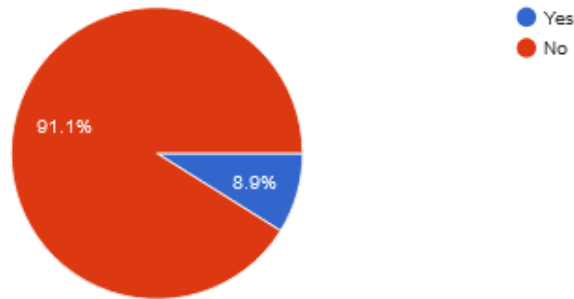


*Figure 3.8: Do you know of PaliGemma as a vision-language model?*

According to Figure 3.8, 91.1% of the respondents (41 respondents) responded ‘No’, indicating that they do not know that PaliGemma is a vision-language model. 8.9% of the respondents (5 respondents) responded ‘Yes’, indicating that they know PaliGemma is a vision-language model.

Do you know of any system that is able to describe image after prompting?  
(image captioning)

 [Copy chart](#)

45 responses

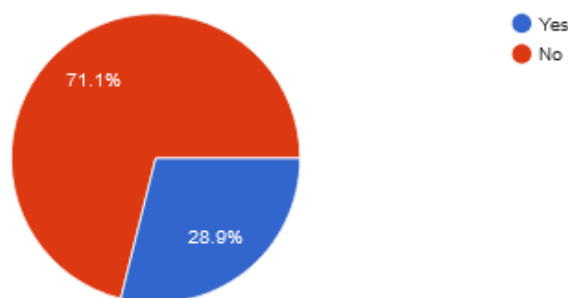


*Figure 3.9: Do you know of any system that is able to describe image after prompting?*

*(image captioning)*

According to Figure 3.9, 71.1% of respondents (32 respondents) answered ‘No’, indicating that they do not know any system that is able to describe image after prompting. 28.9% of respondents (13 respondents) responded ‘Yes’, indicating that they do know at least a system that is able to describe image after prompting.

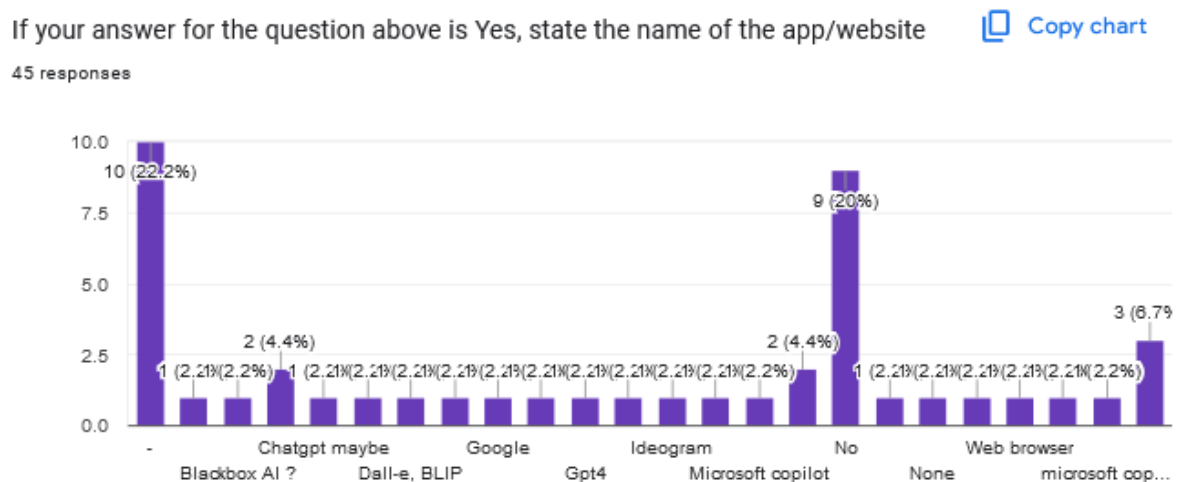


Figure 3.10: If your answer for the question above is Yes, state the name of the app/website

According to Figure 3.10, the application or websites that some of the respondents know can describe image after prompting are ChatGPT, Microsoft Copilot, Astica.ai, Blackbox.ai, DALL-E, BLIP, Google Camera, Google, Ideogram, Imagecaptiongenerator.com, and web browser.

Would you be interested to know how PaliGemma works to describe an image?

 [Copy chart](#)

45 responses

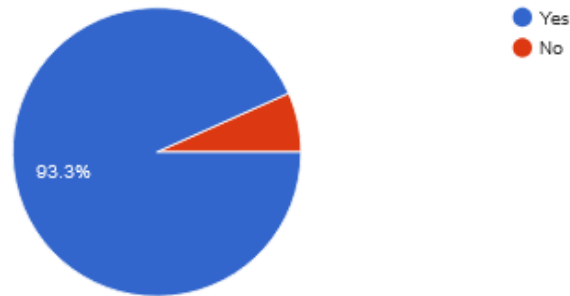


*Figure 3.11: Would you be interested to know how PaliGemma works to describe an image?*

According to Figure 3.11, 93.3% of the respondents (42 respondents) responded ‘Yes’, indicating that they are interested in knowing how PaliGemma works to describe an image, meanwhile the remaining 6.7% of the respondents (3 respondents) responded ‘No’.

If there is a chance to build such system, what features would you expect/prefer?

 [Copy chart](#)

45 responses

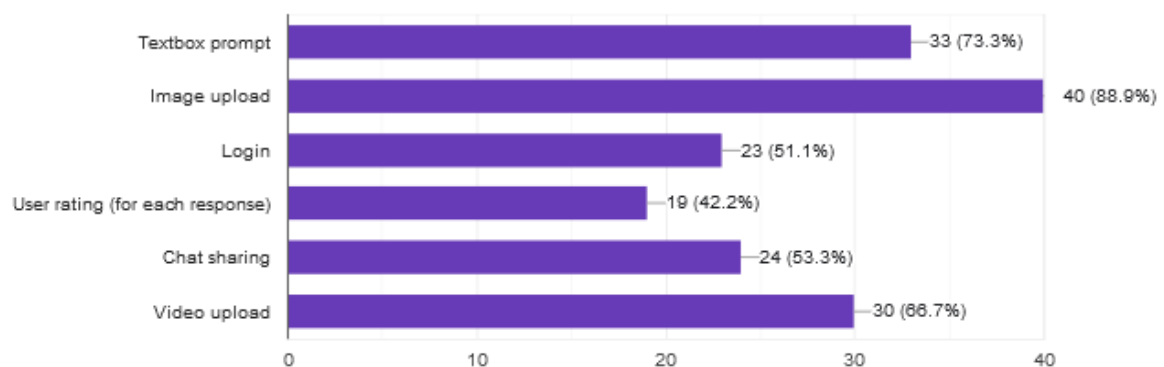


*Figure 3.12: If there is a chance to build such system, what features would you expect/prefer?*

According to Figure 3.12, 33 respondents want ‘Textbox prompt’ feature in the system, 40 respondents want ‘Image upload’ feature in the system, 23 respondents want ‘Login’ feature

in the system, 19 respondents want ‘User rating for each response’ feature in the system, 24 respondents want ‘Chat sharing’ feature in the system, and 30 respondents want ‘Video upload’ feature in the system.

What other features would you suggest for this web application chatbot?

23 responses

|                                                              |
|--------------------------------------------------------------|
| -                                                            |
| Without limit of usage                                       |
| User friendly                                                |
| Easy button to sent voice note                               |
| No idea.                                                     |
| voice prompter (?)                                           |
| What about mobile app? Consider it                           |
| Voice recognition                                            |
| If possible lah, ability to store and retrieve chat history. |

*Figure 3.13: What other features would you suggest for this web application chatbot?  
(including answers)*

|                                                                                                                                                                                                                                                                      |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Audio sharing                                                                                                                                                                                                                                                        |
| Making videos                                                                                                                                                                                                                                                        |
| Predictive actions and emotional intelligence                                                                                                                                                                                                                        |
| User can upload image or video with the text description. Not just can only upload image/video/text.<br>Able to select what kind of description to be generate. eg, from biological view, from historical view<br>Able to select the description in various language |
| Upload file                                                                                                                                                                                                                                                          |
| Voice prompt                                                                                                                                                                                                                                                         |
| NA                                                                                                                                                                                                                                                                   |
| feedback                                                                                                                                                                                                                                                             |

*Figure 3.14: Additional answers for ‘What other features would you suggest for this web application chatbot?’*

|                                                                                                                                                                                |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| saving                                                                                                                                                                         |
| Convert picture                                                                                                                                                                |
| 1.The bot having multilingual support to make it accessible worldwide.<br>2. Utilize GPS data to narrow down species based on user's location and known regional biodiversity. |

*Figure 3.15: Additional answers for ‘What other features would you suggest for this web application chatbot?’*

In Figure 3.13, 3.14 and 3.15, the responses vary, indicating that respondents are interested in having more unique features in the app other than features mentioned in previous question. One of the most requested features based on the responses is Voice Prompt.

### 3.4.2 System Features

Based on the analysed questionnaire, there are features suggested by respondents which are considered for the web application such as report section and multilingual options. Other features were considered based on comparison of existing systems before distributing the questionnaire. Among the features considered for the web application are as follows:

- |                                   |                                              |
|-----------------------------------|----------------------------------------------|
| 1. Textbox prompt                 | 9. Chat sharing                              |
| 2. Image upload                   | 10. Help section                             |
| 3. Login                          | 11. Report issues (user experience feedback) |
| 4. Chat history                   | 12. Languages option                         |
| 5. User rating (thumbs up & down) | 13. Save chat option (save to chat history)  |
| 6. Text generation capabilities   | 14. Copy response                            |
| 7. Model switch                   | 15. Menu                                     |
| 8. New chat                       |                                              |

## 3.5 System design

### 3.5.1 Use Case Diagram

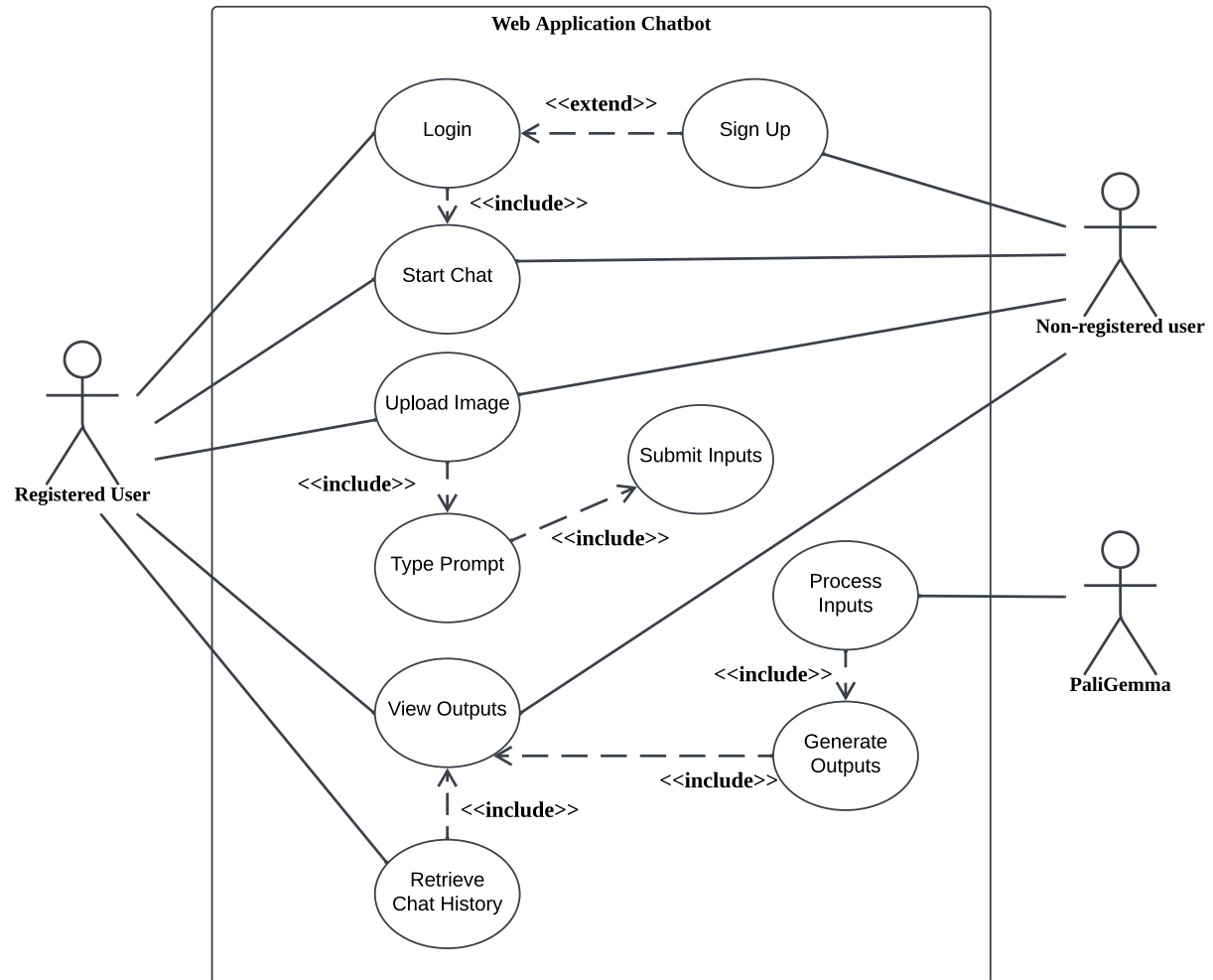


Figure 3.16: Use case diagram.

Figure 3.16 illustrates the flow of a web application chatbot that is integrated with PaliGemma. There are two users within this system which are registered user and non-registered user, and PaliGemma, the AI model embedded within the web application chatbot.

Beginning with a registered user interaction, where users log into the web application. Once logged in, they can start a chat with the chatbot. In the chat, a registered user can upload

an image and type a prompt. After preparing their inputs, users submit both the inputs to the system.

Following the submission, the system processes the inputs and forwards them to PaliGemma for further analysis and generation. PaliGemma then produces outputs based on the provided inputs, which are subsequently returned to the web application.

Additionally, a registered user has access to an additional feature in which they are able to retrieve their chat history with the chatbot when they are logged in and can view the inputs and outputs from a past conversation with the chatbot.

Besides illustrating a registered user's interaction with the system, Figure X also shows a non-registered user interaction in the system. A non-registered user is a user that could access and use the chatbot without logging in, such as submitting inputs and viewing the outputs generated by PaliGemma. Unless a non-registered user decides to start a new chat and want to save their chat history, then they will not be able to retrieve a chat history from past conversations with the chatbot. When a non-registered user decides to start a new chat, the system will give options for the user to choose between logging in or signing up. A non-registered user should choose the option to sign up. Afterwards, they will be navigated to the login page to log into the system and starts a new chat. The previous conversation will be automatically saved into the chat history once a user is logged in.

### 3.5.2 Sequence Diagram

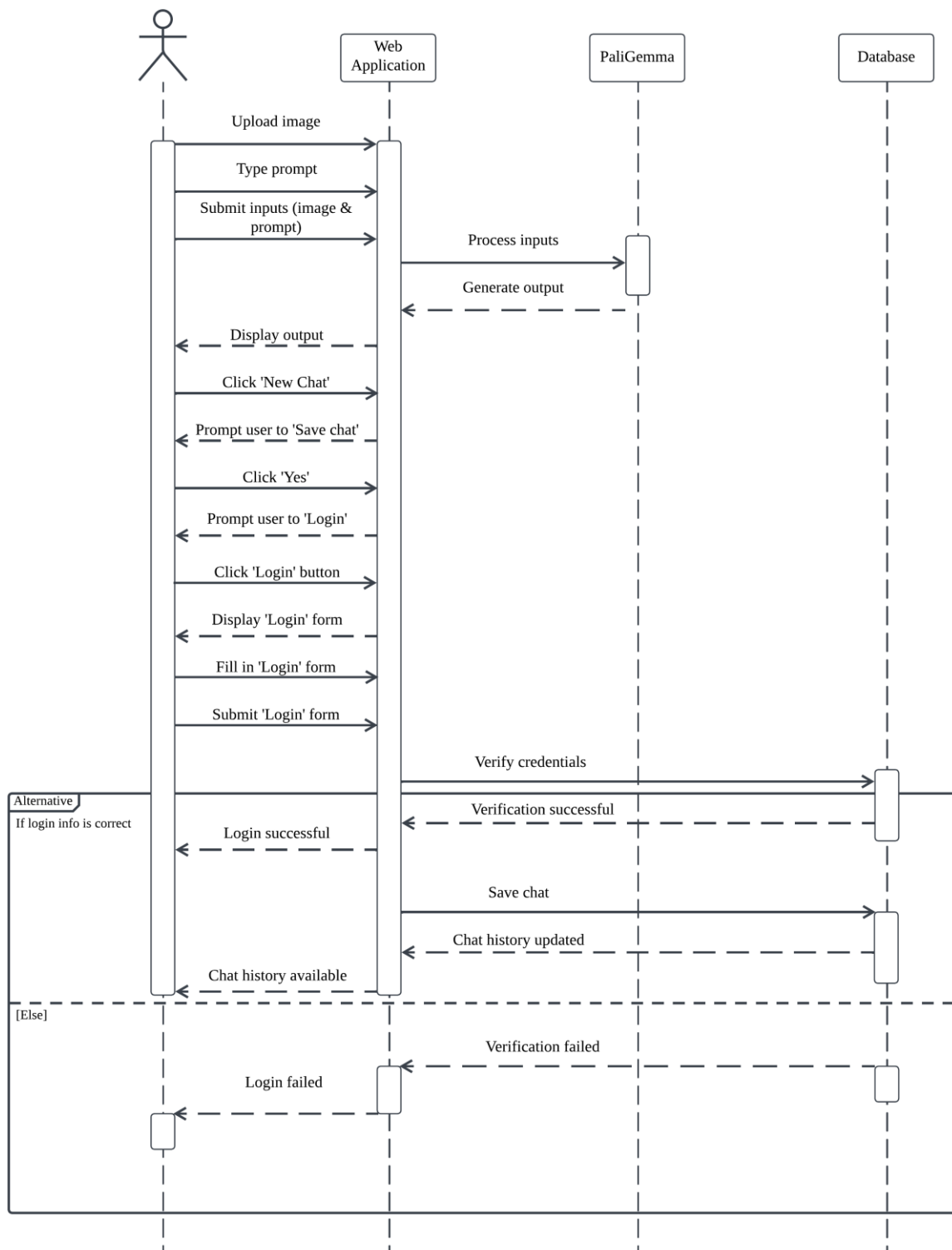


Figure 3.17: PaliGemma processes when user is not logged in.

Figure 3.17 is a sequence diagram for the activity of ‘non-logged in’ session which consists of user as the actor, web application, database and PaliGemma as the object. When user accesses the web application, they will see an interface that urges users to upload an image and type a prompt based on the image uploaded. User will upload an image and types a prompt. These inputs consisting of the image and the prompt will then be submitted through the web application and forwarded to PaliGemma for processing. PaliGemma will analyze the inputs (image and prompt), process it, then generates an output based on the given inputs. Once the output is ready, it is sent back to the web application. The web application will then display the generated output to the user.

Due to the nature of PaliGemma, the proposed project (chatbot) is not a multi-turn chatbot. Therefore, once a prompt is submitted, user will need to opt for ‘New Chat’ to start a new conversation with new prompts. If user choose to start a new chat session, they will be prompted to save the current chat session. If the user decides to save the current chat session, the system will initiate a login process.

The user is prompted to log in by filling in a login form and submitting it. The submitted login details will be sent to the database for verification. Once the verification is successful, the login will be successful and the chat will be saved into a chat history. However, if the login fails, the system will notify the user with a ‘Login failed’ message.

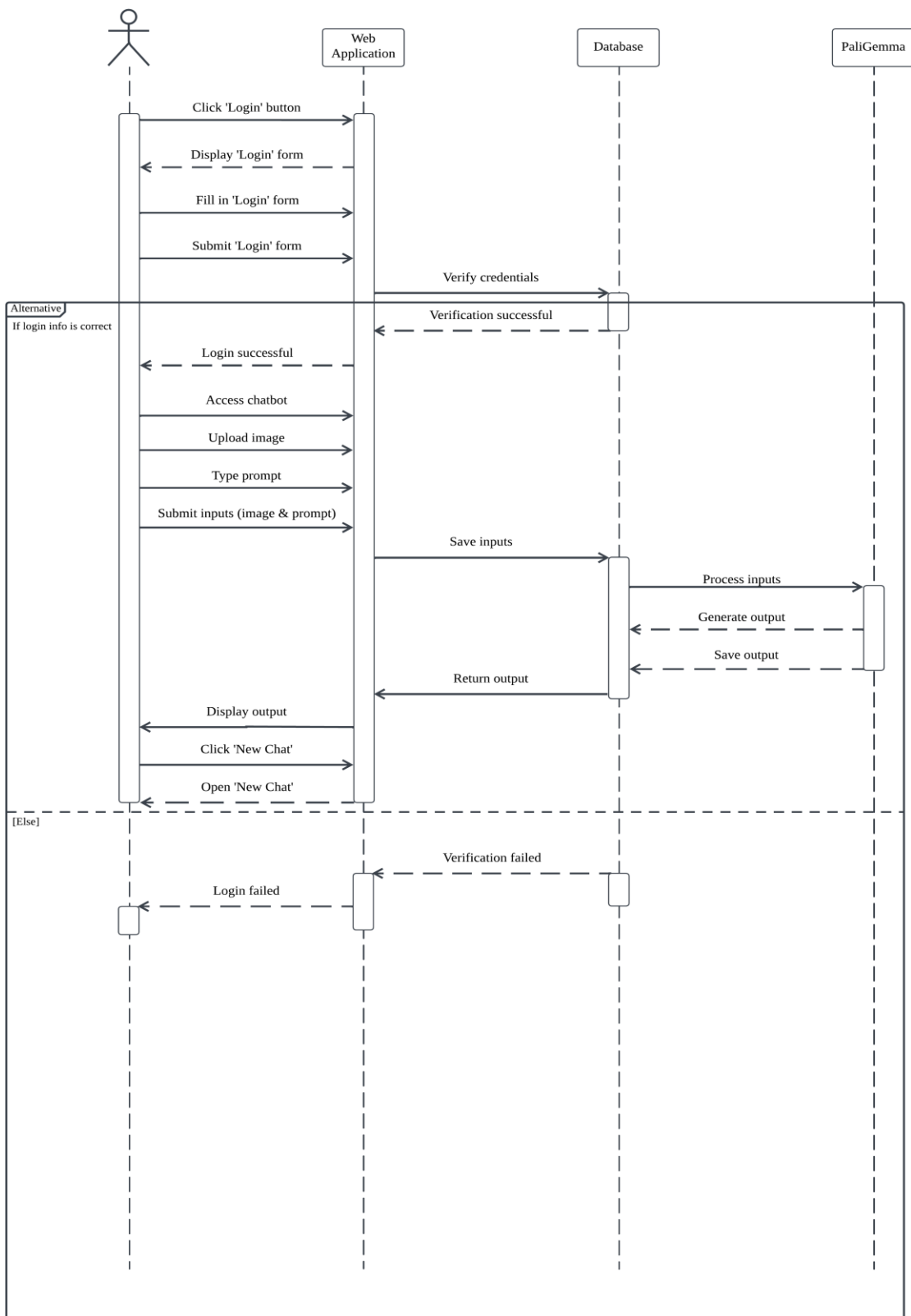


Figure 3.18: PaliGemma processes when user is logged in.

Based on Figure 3.18, this is a sequence diagram for the activity of a 'logged in' session which consists of a user acting as an actor and objects which are web application, database and PaliGemma. The login process begins when the user clicks the 'Login' button. The web application responds by displaying a 'Login' form for the user to fill in. Once the form is completed and submitted, the web application sends the credentials to the database for verification. If the credentials are valid, the database send a 'Login successful' message back to the user through the web application.

After logging in successfully, the user can proceed to interact with the chatbot. The user uploads an image and types a prompt then submits through the web application to the database to save the inputs. The database will then forward the saved inputs to PaliGemma for input processing. Next, PaliGemma will generate an output based on the input and sends the output to the database for storing. Lastly, the database will display the output to the user through the web application. The user can opt to start a 'New Chat' which opens a fresh chat interface for further interaction.

If the login verification fails, the database will send a 'Verification failed' message to the web application which will then notify the user with a 'Login failed' message.

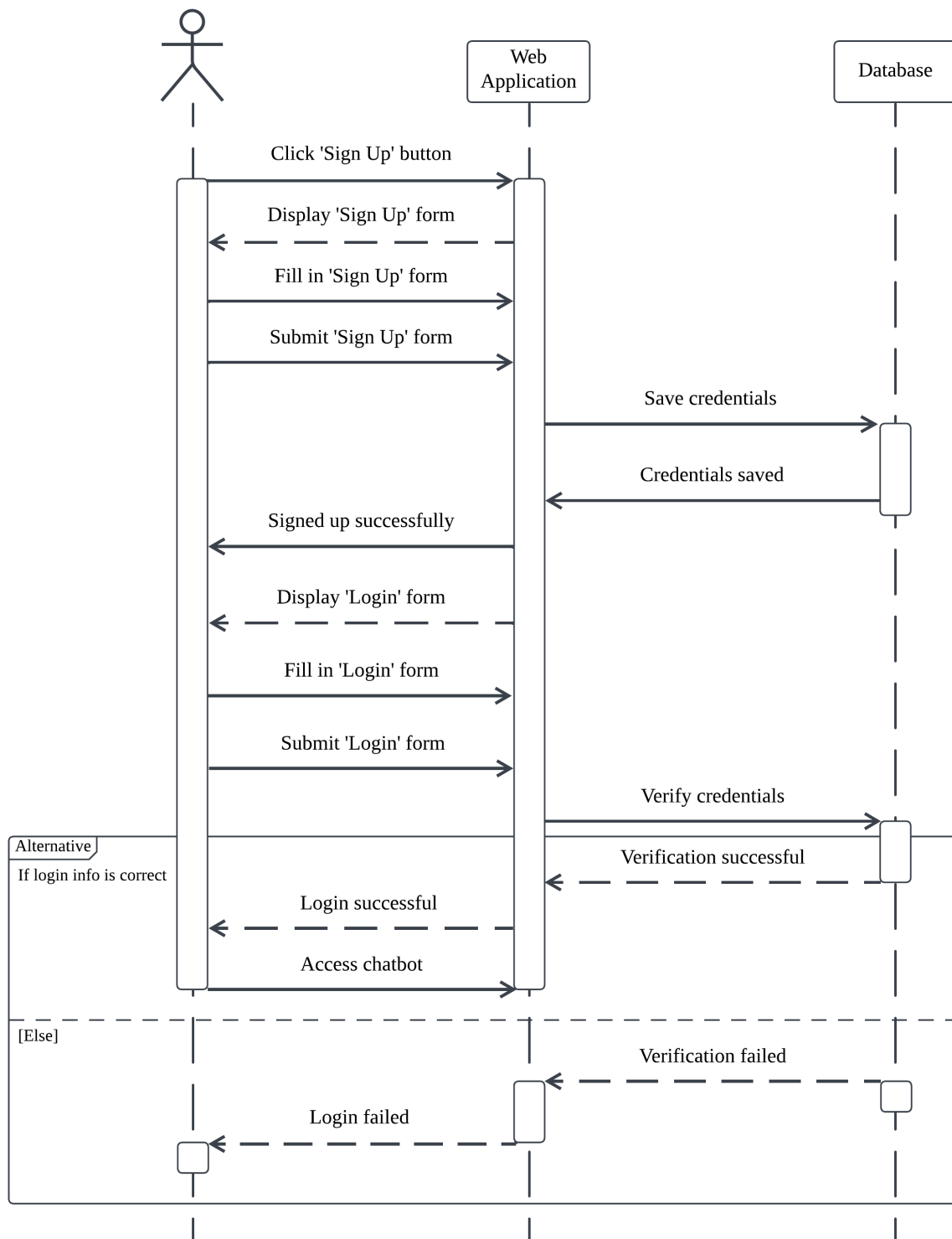


Figure 3.19: Web Application processes when user signs up.

Figure 3.19 is a sequence diagram for the activity of a 'Sign up' session which consists of only an actor and objects which are web application and database, compared to the 'Login' and 'Non-logged in' activities. The user registration process begins when the user clicks on the 'Sign Up' button. Then, the web application will display the 'Sign up' form for the user to fill in and submit. After the 'Sign up' form is completed and submitted, the web application will then send the provided credentials to the database for storing. Once the database confirms that the credentials have been saved, the web application will inform the user that the registration process was successful.

Following a successful registration, the web application will display a login form for the user to fill in and submit. Once the login form is completed and submitted, the web application will send the credentials to the database for verification.

If the verification is successful, the web application will send a 'Login successful' message to the user. The user will then be able to access the chatbot. If the verification fails, the web application will send a 'Login failed' message to the user. The user will need to re-fill in the login form and do so until the login is successful.

### 3.5.3 Class Diagram

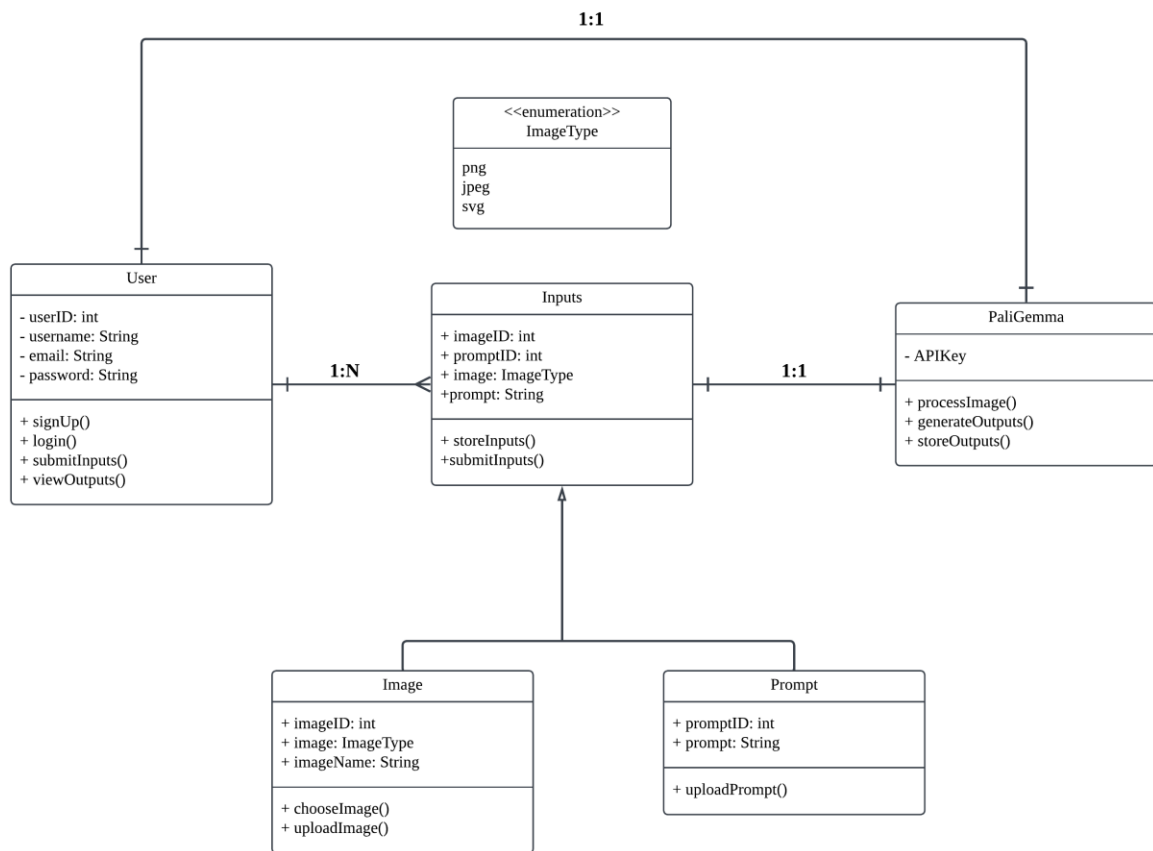


Figure 3.20: Class diagram.

The class diagram in Figure 3.20 represents a system designed to identify nature and wildlife using PaliGemma, a Visual-Language Model (VLM). It encapsulates the main components and their relationships to facilitate the system's functionality, including user interaction, data handling, and AI model integration.

At the core of the system is the **User** class, which represents the end-users interacting with the application. Each user is uniquely identified by attributes such as `userID`, `username`, `email`, and `password`. Users can perform essential actions like signing up (`signUp()`), logging in (`login()`), submitting inputs (`submitInputs()`), and viewing outputs (`viewOutputs()`). The

**Inputs** class serves as the intermediary between users and the system, consolidating both image and textual inputs required for processing. A single user can submit multiple inputs, forming a one-to-many relationship between the User and Inputs classes.

The **Inputs** class has attributes such as imageID, promptID, image, and prompt, which collectively define the data submitted by the user. This class provides methods like storeInputs() to save the inputs and submitInputs() to send them for further processing. Inputs are composed of two key components: images and prompts. The **Image** class handles image-related data, with attributes like imageID, imageName, and imageType (an enumeration supporting formats such as PNG, JPEG, and SVG). Users can choose (chooseImage()) and upload (uploadImage()) images through this class. Similarly, the **Prompt** class manages textual inputs, with attributes like promptID and prompt and a method to upload text queries (uploadPrompt()). Both Image and Prompt classes form a composition relationship with the Inputs class, ensuring that inputs are complete and well-structured.

The **PaliGemma** class represents the AI model that processes user inputs. It includes an APIKey attribute for authenticating and interacting with the PaliGemma through HuggingFace's access token. The class includes methods such as processImage() to analyze uploaded images, generateOutputs() to produce species identification results, and storeOutputs() to save the generated outputs. There is a one-to-one relationship between the Inputs and PaliGemma classes, signifying that each input is processed individually by the AI model.

The system design emphasizes modularity, with distinct classes for users, inputs, images, prompts, and the AI model. This modular approach allows for flexibility, scalability, and ease of maintenance. By centralizing the Inputs class and its relationship with PaliGemma,

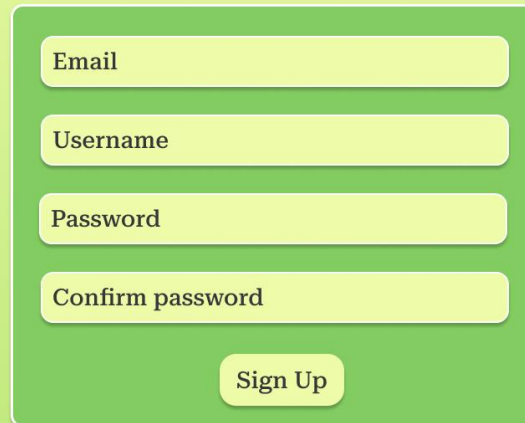
the system ensures efficient data handling and seamless interaction between the user and the AI model. Overall, the class diagram captures the essential components and workflows of a robust application for nature and wildlife identification, leveraging the capabilities of PaliGemma while promoting an intuitive user experience.

### **3.6 Wireframe Design of Interface**

The main component of web application's interface prototype such as the login page, sign up page, main page, chat history page, chatbot response page is designed using Figma. The flow of the web application follows two main flows, registered and logged in user flow and unregistered and unlogged in user flow, which will be showcased in the subsection 3.6.1 and 3.6.2, respectively. The sign-up page and login page are shown in Figure 3.21 and Figure 3.22.

# PaliGemma

Sign Up to access PaliGemma



The sign-up form is a light green rounded rectangle containing four text input fields and a submit button. The fields are labeled 'Email', 'Username', 'Password', and 'Confirm password' from top to bottom. The 'Sign Up' button is located at the bottom right of the form.

|                  |
|------------------|
| Email            |
| Username         |
| Password         |
| Confirm password |
| Sign Up          |

*Figure 3.21: Sign-Up Page.*

In Figure 3.21, user is required to fill in their email, username and password. Password must be re-entered again to confirm that both passwords match to proceed with the sign-up process. Once all fields have been filled, user needs to click the ‘Sign Up’ button to conclude the sign-up process and the button will navigate user to the login page in Figure 3.22.

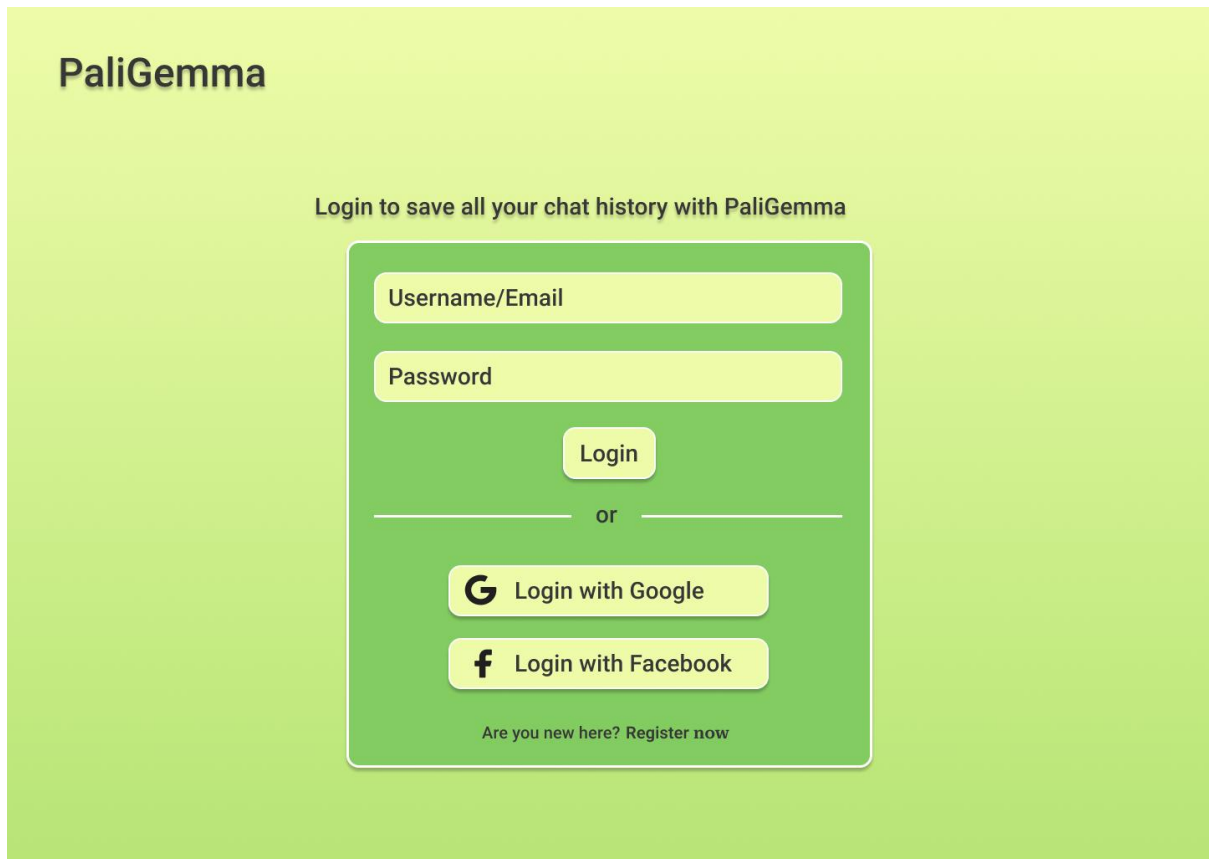
The image shows a login page for 'PaliGemma' with a light green background. At the top left is the 'PaliGemma' logo. Centered below it is the text 'Login to save all your chat history with PaliGemma'. Below this is a white rounded rectangle containing the login form. The form has two input fields: 'Username/Email' and 'Password'. Below these is a 'Login' button. A horizontal line with the word 'or' in the center separates the login section from the social login section. The social login section contains two buttons: 'Login with Google' (with a Google 'G' icon) and 'Login with Facebook' (with a Facebook 'f' icon). At the bottom of the white box is a link that says 'Are you new here? Register now'.

*Figure 3.22: Login Page.*

In Figure 3.22, a user must fill in their username or email address and password, as registered in Figure 3.21 then click the ‘Login’ button to log into the web application. Users will also be able to login using their Google account or Facebook account.

### **3.6.1 Unregistered and Non-Logged In User Flow**

In this flow, users are assumed as a first-time user or regular unregistered user. The accessible features are limited to non-logged in user, particularly the chat history feature.

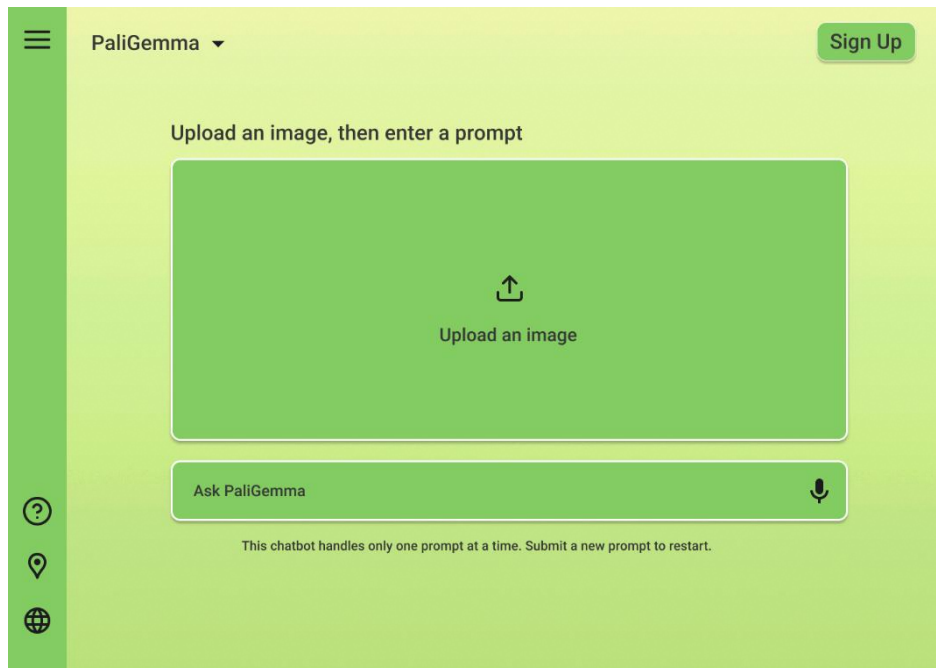


*Figure 3.23: Main page for Non-Logged In User.*

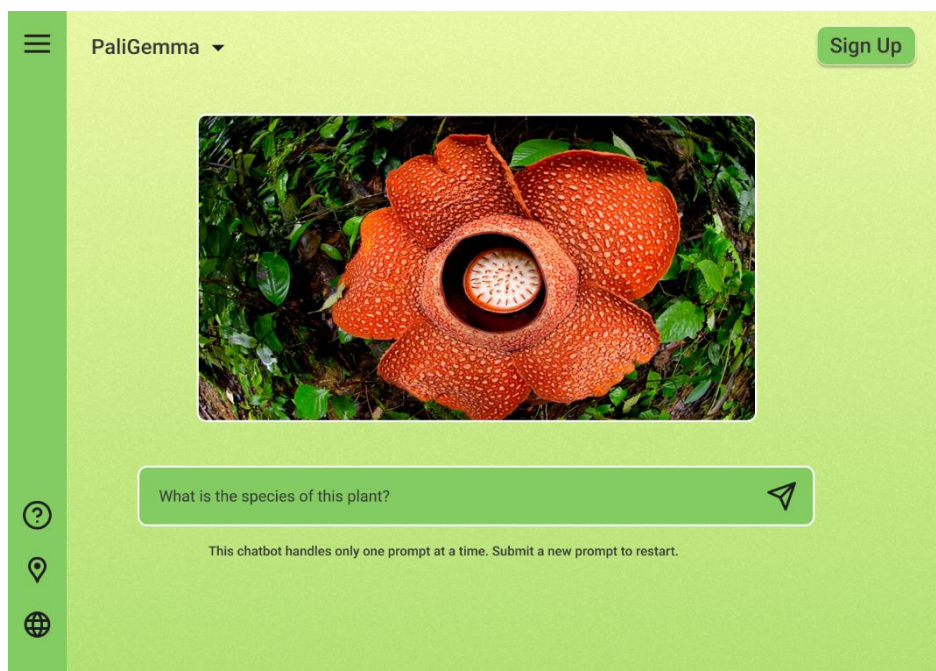


*Figure 3.24: Image Upload and Prompt Sample for Non-Logged In User.*

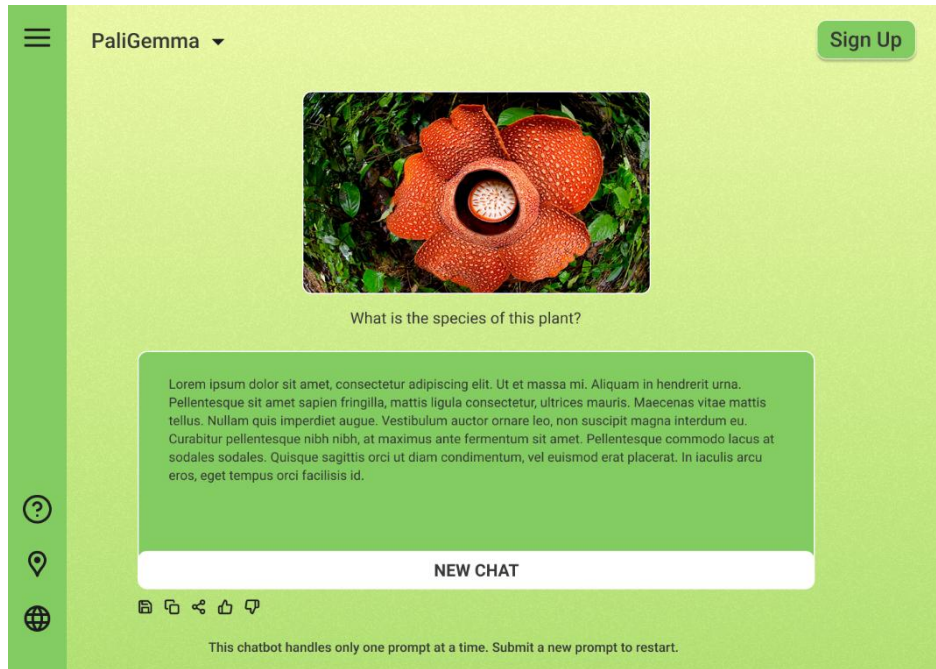


*Figure 3.25: Chatbot Response Sample for Non-Logged In User.*

Figure 3.23, Figure 3.24 and Figure 3.25 shows the pages to be navigated by a non-logged in users. When a non-logged in user decides to start a new chat and click on the ‘New Chat’ button in Figure 3.25, an overlay will pop-up and prompts the user to save their current chat as shown in Figure 3.26.

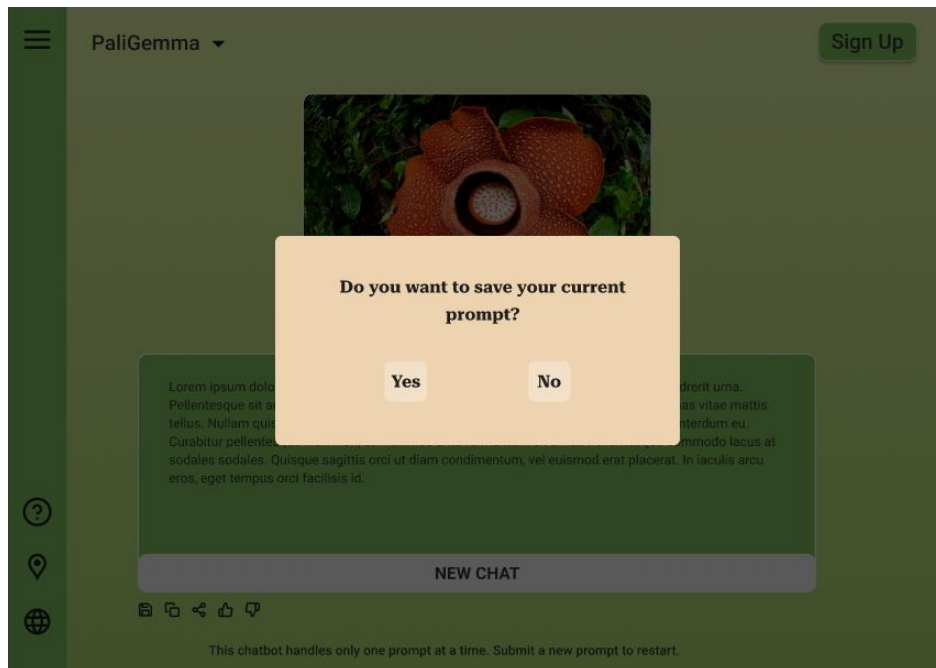


*Figure 3.26: A Pop-Up of Overlay Prompting User to Save Current Chat.*

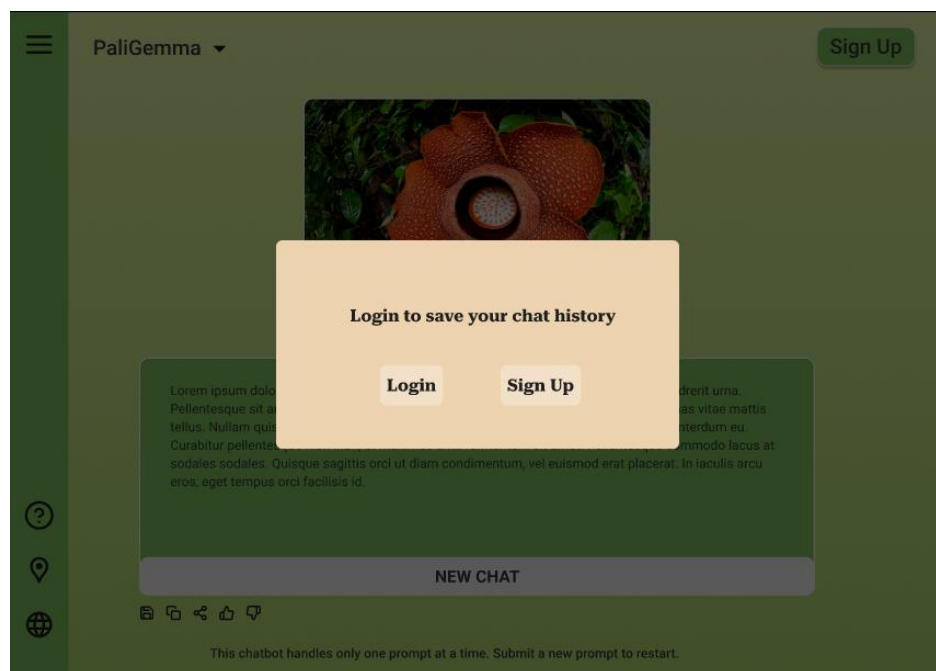


*Figure 3.27: User Choose 'Yes' in Figure 3.26.*

If a user decides to save their current prompt and chose 'Yes' in Figure 3.26, the web application will prompt the user to either 'Login' or 'Sign Up'. Clicking 'Login' will renavigate

the user to the Login page in Figure 3.22 and clicking ‘Sign Up’ will renavigate the user to the Sign Up page in Figure 3.21.

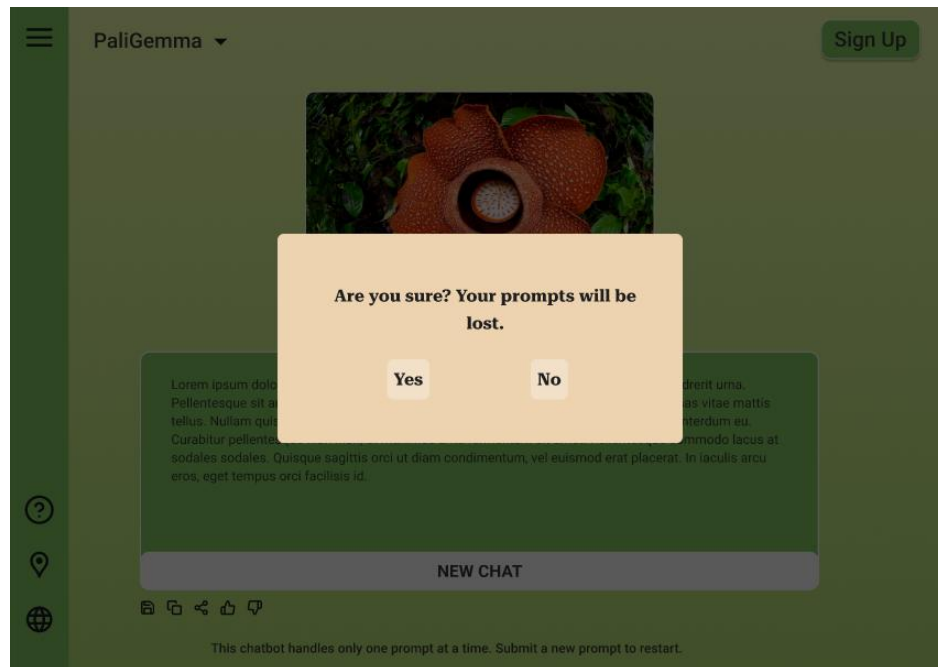


*Figure 3.28: User Choose ‘No’ from Figure 3.26.*

If a user chooses ‘Yes’ in Figure 3.28, the web application will renavigate the user to the main page in Figure 3.23. If a user chooses ‘No’ in Figure 3.28, the overlay from Figure 3.26 will reappear and user will have to choose ‘Yes’ to save their current prompt and proceed to a new chat.

### **3.6.2 Registered and Logged In User Flow**

In this flow, users are a registered entity in the web application and logged into the system to access the chatbot. Special feature access to this user is the chat history feature, where their past conversations or prompts with the web application chatbot will be saved in a database.

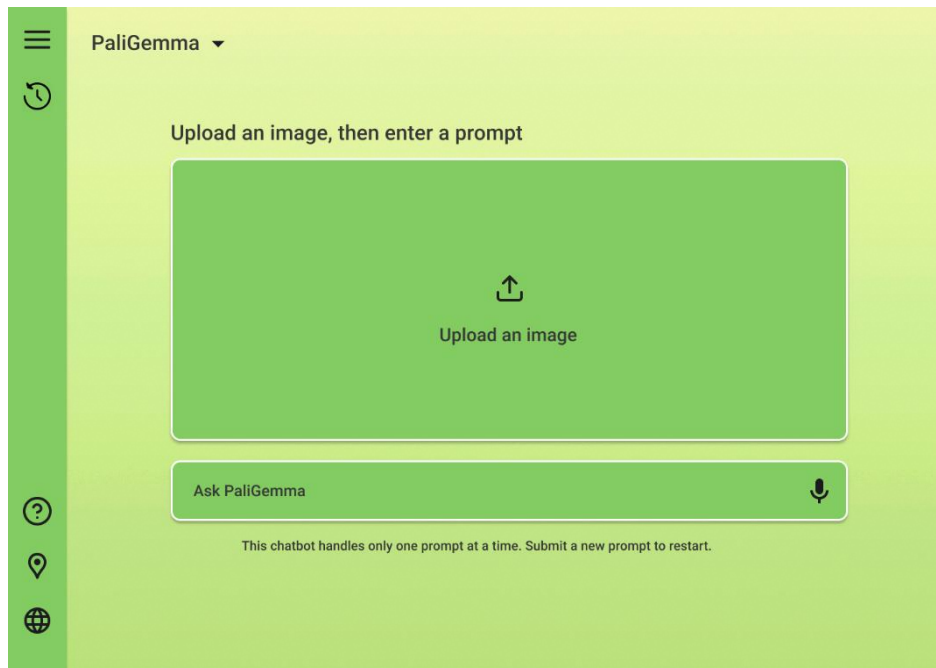


*Figure 3.29: Main Page for Logged In User.*

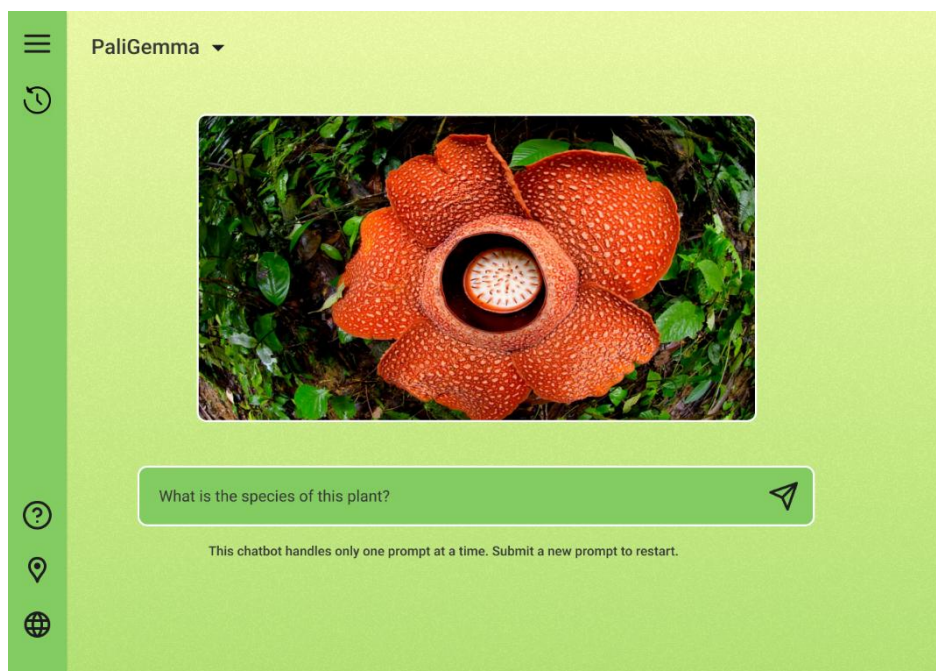


*Figure 3.30: Image Upload and Prompt Sample for Logged In User.*

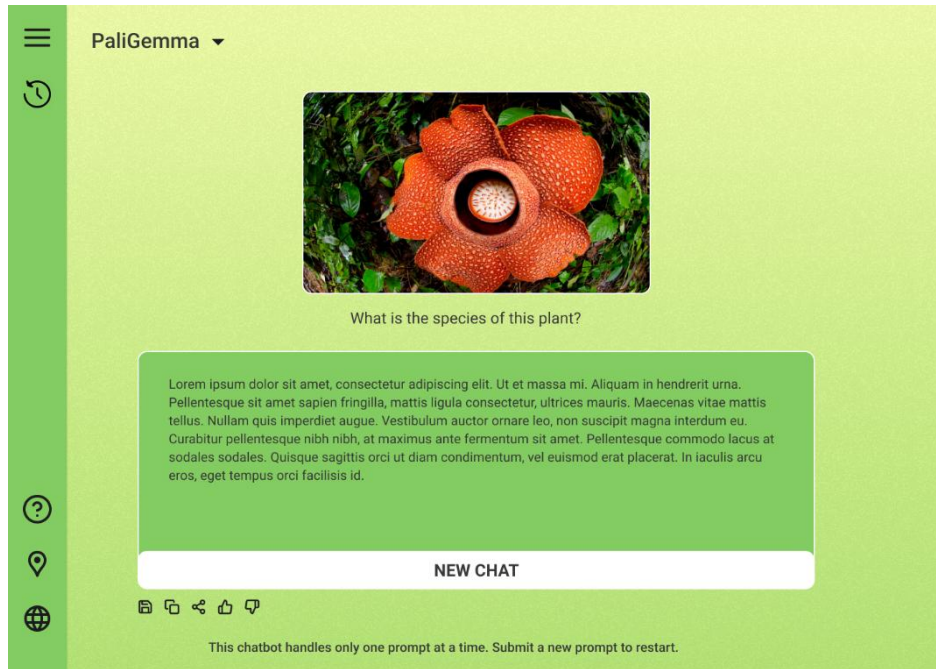


*Figure 3.31: Chatbot Response Sample for Logged In User.*

Figure 3.29, Figure 3.30 and Figure 3.31 are the pages to be navigated by a logged in user of the system. These figures' interface are similar to Figure 3.23, Figure 3.24 and Figure 3.25, however there are minor differences between these figures. Figure 3.23 – Figure 3.25 have a 'Sign Up' button on the top-right of the page while in Figure 3.29 – Figure 3.31, this button is removed because the user is already logged in.

In Figure 3.31, if a user decides to proceed further and start a new chat by clicking on the 'New Chat' button, the user will be renavigated to the main page in Figure 3.29. This section of the web application does not require a pop-up overlay as in Figure 3.26 – Figure 3.28 because the system will automatically save the prompts into the chat history.

Users are also able to click on the icons located at the bottom right of the response box in Figure 3.31. Each icon have a distinct function and these icons are the save icon, copy icon, share icon, thumbs-up icon and thumbs-down icon.

### 3.6.3 Additional features

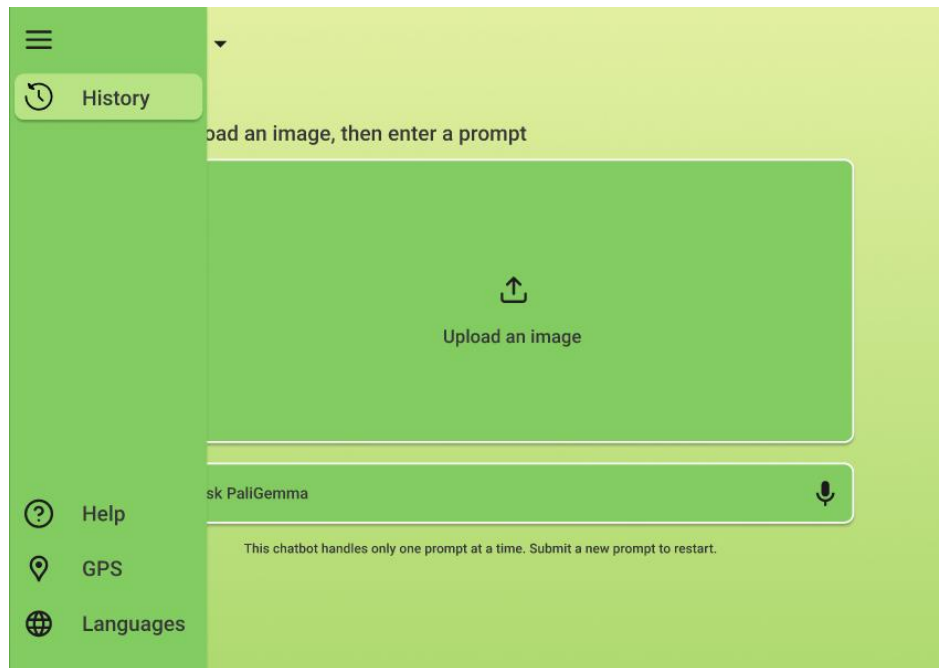


*Figure 3.32: Sidebar menu.*

The web application features a sidebar menu positioned on the left side of the page which can be accessed by clicking on the hamburger icon in the top-left corner. The menu includes options for History, Help, GPS and Languages. When user clicks on any of the options, the system will navigate to corresponding page.

### 3.7 Database Component

The database of this system will enforce MySQL and using XAMPP to locally host the database during development. A visualization of the database structure is shown in Figure 3.33.

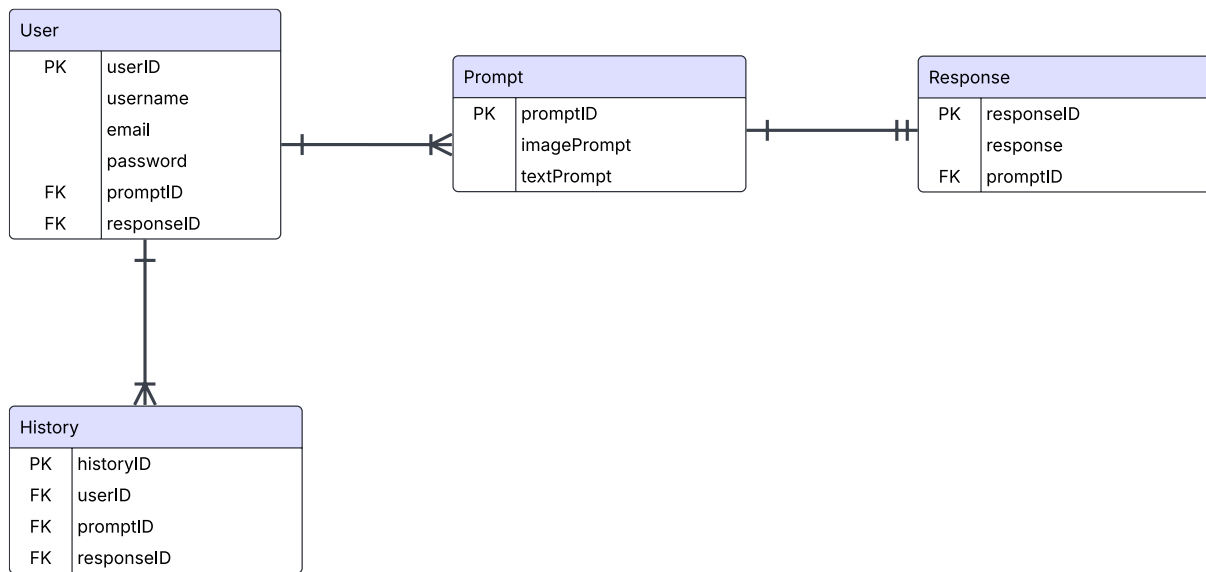


*Figure 3.33: Entity Relationship Diagram.*

Based on Figure 3.33, the database will be comprising of four tables which are User, Prompt, Response and History. The User table stores user's credentials with the attribute promptID from the Prompt table and attribute responseID from the Response table as its foreign keys and one user can submit multiple prompts and receive responses. The Prompt table stores the inputs submitted by user, in this case images and text, which are represented as imagePrompt and textPrompt in the table and its primary key is promptID.

The Response table stores the response generated by PaliGemma into the system's database represented by the attribute 'response' and responseID as its primary key. This table has promptID as its foreign key to link each response to its corresponding prompt. Additionally, each prompt can only generate one response. The History table stores user's past conversation with the chatbot into the database with the attribute historyID as its primary key to uniquely identify for each history record. This table contains three foreign keys namely, userID, promptID and responseID to track which user made the prompt and received the response.

Furthermore, this table maintains a log of all interactions by linking users to their prompts and responses.

### **3.8 Summary**

This chapter outlines the methodology, system requirements, and UML design approach for developing a web application that integrates PaliGemma for nature and wildlife identification. The Modified Waterfall Model with Iteration is chosen as the methodology for this proposed system which features phases for requirements gathering, design, implementation, testing and maintenance. This methodology approach also allows for iterative adjustments across phases. The system's software requirements include tools like Visual Studio Code, Django, MySQL and APIs such as HuggingFace for PaliGemma integration. Hardware specifications also emphasize GPUs and sufficient storage to handle PaliGemma's computational demands. A requirement elicitation was conducted via a questionnaire which targets 45 respondents from diverse backgrounds. This elicitation method gathers suggestions and feedback from respondents on the system's features. Additionally, the system design is illustrated through Unified Modelling Language (UML) diagrams such as a use case diagram, sequence diagrams and class diagram. These diagrams highlighted user interactions, backend processes and the integration of PaliGemma for analyzing images and generating captions based on user's inputs. The important components include user authentication, input handling and AI model integration. The development process will involve building the web application's frontend and backend using Django which implements PaliGemma via API as well as testing functionalities for usability and performance.

## **CHAPTER 4: IMPLEMENTATION AND TESTING**

### **4.1 Introduction**

Chapter 4 will be documenting the implementation of the system proposed mainly the technical aspects of the implementation. Commencing with the data collection for dataset preparation, setting up the development environment, fine-tuning of the PaliGemma model using custom prepared dataset, deployment of said fine-tuned model and the integration of frontend, backend, and database of web application.

### **4.2 Data Collection Implementation**

The implementation of this project's data collection will include downloading datasets from iNaturalist's website while also developing a web scraper for iNaturalist. The downloaded dataset from iNaturalist will be in an Excel CSV format in which the data is in tabular format for readability. This dataset will contain image URL alongside with scientific name and common name of each image.

A simple web scraper will also be developed to download the images from image URL extracted from the website. Downloaded images will be named in sequential numbering (ascending order) based on the data row it is downloaded from which is the CSV file. The dataset prepared here will be used as training set and validation set. A test set will be prepared by gathering data of local biodiversity species from Faculty of Resource Science and Technology.

The training steps of PaliGemma needs a JSONL file due to the model's incapability of processing basic JSON file. To achieve this, the Excel CSV file needs to be further modified by initially removing the header row. Next, the CSV file will be converted into JSON and will then be converted into JSONL. This is to ensure that smooth file conversion and avoid corrupted data rows. After obtaining the JSONL format, a prefix (prompt) must be added into the file. A prefix can be set to a specific prompt for instance, "what is this animal's species?", for a dataset containing wildlife's species dataset and "what is this plant's species?" for dataset containing plants' species. It is important to note that the prefix must be the first object in the JSONL file. The next step is to combine the suffix considering that the CSV file had separate columns for scientific names and common names. In PaliGemma, suffix is the output of the

prompts after recognizing the prefix and predicting a suffix after analysing an image. The suffix will be combined in a specific format like “scientific name (common name)”. After all the successful conversions and programming, it is important to note that the JSONL file still contains the image URLs from the CSV file and the image URLs needs to be modified into the file names after downloading the images (sequential numbering). For example, an image URL (<https://inaturalist-open-data.s3.amazonaws.com/photos/2277314/medium.JPG>) needs to be renamed to “1.jpg”.

#### 4.2.1 Dataset Preparation

As explained in 4.2 Data Collection Implementation, a series of datasets will be prepared by downloading Excel CSV through iNaturalist website. In this website, a query must be made before exporting it as CSV file. However, the query must include less than 200,000 observations. Observations in this context are biodiversity species observed by users of iNaturalist website and uploaded into the website’s database.

The interface of iNaturalist website is shown in Figure 4.1.

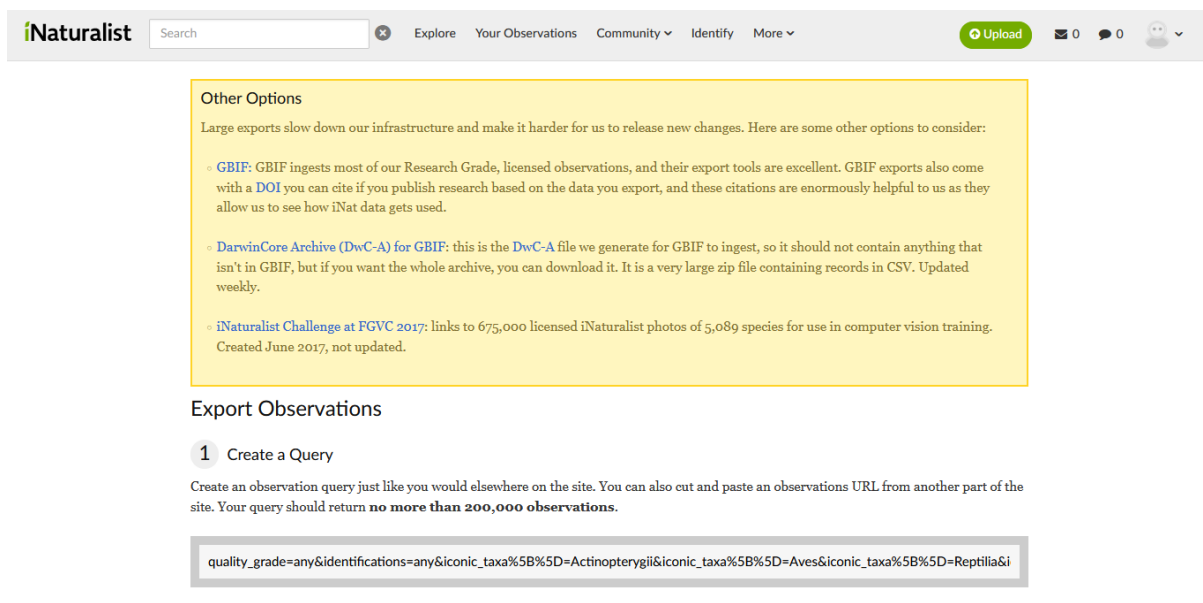


*Figure 4.1: iNaturalist website interface.*

In this interface, there are 4 steps required to export the dataset. The first step as shown in Figure 4.2 is to create a query by filtering and selecting categories needed to download into the dataset. The second step is previewing the filtered and categorized dataset as shown in Figure 4.3. At this stage, there will be warnings if the

data selected is more than 200,000 and the filter and categories need to be scaled down by configuring the verifiable, threatened, introduced, native and popular section in step 1. Next, step 3 involves choosing columns for the Excel CSV file and in this project, only 3 columns are needed which are the image\_url in Basic, scientific\_name and common\_name in Taxon as shown in Figure 4.4. The final step is to click on the “Create Export” button in step 4. This will trigger the CSV file download and recent exports or downloads will be shown in step 4 as shown in Figure 4.5.

## 1 Create a Query

Create an observation query just like you would elsewhere on the site. You can also cut and paste an observations URL from another part of the site. Your query should return **no more than 200,000 observations**.

`quality_grade=any&identifications=any&iconic_taxa%5B%5D=Actinopterygii&iconic_taxa%5B%5D=Aves&iconic_taxa%5B%5D=Reptilia&i`

Search 

Filter

☐ w/ photos
☐ w/ sounds
Quality grade
☒ Any
☐ Research
☐ Needs ID
Reviewed
☒ Any
☐ Yes
☐ No

Identifications
☒ any
☐ most agree
☐ some agree
☐ most disagree
Captive / Cultivated
☒ Any
☐ Yes
☐ No

Geoprivacy 
Taxon Geoprivacy

Show only
☐
☒
☒
☒
☒
☒
☒
☒
☒
☒
☒
☒
☒
☒
☒
☒
☒
☐
☒
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐
☐

Your export will be too big. Change your filters until there are fewer than 200,000 observations.

## 2 Preview

1 - 30 of 3713813

| Photos / Sounds                                                                   | Species / Taxon Name                                               | Observer                        | Date observed               | Place                                                                              |                                                            |
|-----------------------------------------------------------------------------------|--------------------------------------------------------------------|---------------------------------|-----------------------------|------------------------------------------------------------------------------------|------------------------------------------------------------|
|  | <b>Mabel Orchard Orbweaver</b><br>( <i>Leucauge argyrobrapta</i> ) | <a href="#">gabriel_victor1</a> | April 20, 2025 05:52 PM -03 |  | 1 ID<br><a href="#">Needs ID</a><br><a href="#">View »</a> |
|  | Something...                                                       | <a href="#">brent79246</a>      | April 20, 2025 10:53 AM EDT |  | <a href="#">Needs ID</a><br><a href="#">View »</a>         |

Figure 4.3: Step 2 preview.

## 3 Choose columns

Choose the columns you want to export

Basic (All | None)

|                                                        |                                                          |                                                                   |                                                                      |
|--------------------------------------------------------|----------------------------------------------------------|-------------------------------------------------------------------|----------------------------------------------------------------------|
| <input checked="" type="checkbox"/> id                 | <input checked="" type="checkbox"/> uuid                 | <input checked="" type="checkbox"/> observed_on_string            | <input checked="" type="checkbox"/> observed_on                      |
| <input checked="" type="checkbox"/> time_observed_at   | <input checked="" type="checkbox"/> time_zone            | <input checked="" type="checkbox"/> user_id                       | <input checked="" type="checkbox"/> user_login                       |
| <input checked="" type="checkbox"/> user_name          | <input checked="" type="checkbox"/> created_at           | <input checked="" type="checkbox"/> updated_at                    | <input checked="" type="checkbox"/> quality_grade                    |
| <input checked="" type="checkbox"/> license            | <input checked="" type="checkbox"/> url                  | <input checked="" type="checkbox"/> image_url                     | <input checked="" type="checkbox"/> sound_url                        |
| <input checked="" type="checkbox"/> tag_list           | <input checked="" type="checkbox"/> description          | <input checked="" type="checkbox"/> num_identification_agreements | <input checked="" type="checkbox"/> num_identification_disagreements |
| <input checked="" type="checkbox"/> captive_cultivated | <input checked="" type="checkbox"/> oauth_application_id |                                                                   |                                                                      |

Geo (All | None)

Town, county, state, country fields based on coordinates. Analogous administrative levels will be used for observations where these place types do not apply. The private\_\* fields will only be included when using the user filter and you are the user, or when using the project filter and you curate the project.

|                                                         |                                                      |                                                          |                                                                |
|---------------------------------------------------------|------------------------------------------------------|----------------------------------------------------------|----------------------------------------------------------------|
| <input checked="" type="checkbox"/> place_guess         | <input checked="" type="checkbox"/> latitude         | <input checked="" type="checkbox"/> longitude            | <input checked="" type="checkbox"/> positional_accuracy        |
| <input checked="" type="checkbox"/> private_place_guess | <input checked="" type="checkbox"/> private_latitude | <input checked="" type="checkbox"/> private_longitude    | <input checked="" type="checkbox"/> public_positional_accuracy |
| <input checked="" type="checkbox"/> geoprivacy          | <input checked="" type="checkbox"/> taxon_geoprivacy | <input checked="" type="checkbox"/> coordinates_obscured | <input checked="" type="checkbox"/> positioning_method         |
| <input checked="" type="checkbox"/> positioning_device  | <input type="checkbox"/> place_town_name             | <input type="checkbox"/> place_county_name               | <input type="checkbox"/> place_state_name                      |
| <input type="checkbox"/> place_country_name             | <input type="checkbox"/> place_admin1_name           | <input type="checkbox"/> place_admin2_name               |                                                                |

Taxon (All | None)

|                                                   |                                                     |                                                 |                                                       |
|---------------------------------------------------|-----------------------------------------------------|-------------------------------------------------|-------------------------------------------------------|
| <input checked="" type="checkbox"/> species_guess | <input checked="" type="checkbox"/> scientific_name | <input checked="" type="checkbox"/> common_name | <input checked="" type="checkbox"/> iconic_taxon_name |
| <input checked="" type="checkbox"/> taxon_id      |                                                     |                                                 |                                                       |

Figure 4.4: Step 3 choose column.

4 Create Export

Recent Exports

|          | File size | Queued          | Started         | Finished        | Inputs                                                                                                                                                                                                                                                                                                                                         |        |
|----------|-----------|-----------------|-----------------|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------|
| Download | 2.03 MB   | 07 Apr 10:34 PM | 07 Apr 10:34 PM | 07 Apr 11:11 PM | Query<br>quality_grade= <del>any</del> &identifications= <del>any</del> &iconic_taxa[]= <del>Plantae</del> &verifiable= <del>true</del> &threatened= <del>false</del> &introduced= <del>true</del> &native= <del>false</del> &popular= <del>true</del> &spam= <del>false</del><br>Columns image_url, description, scientific_name, common_name | Delete |
| Download | 413 KB    | 07 Apr 09:41 PM | 07 Apr 09:41 PM | 07 Apr 10:17 PM | Query<br>quality_grade= <del>any</del> &identifications= <del>any</del> &iconic_taxa[]= <del>Plantae</del> &verifiable= <del>true</del> &threatened= <del>false</del> &introduced= <del>true</del> &native= <del>false</del> &popular= <del>true</del> &spam= <del>false</del><br>Columns image_url                                            | Delete |

Figure 4.5: Step 4 create export.

### 4.2.2 Dataset description

Describe images (dataset) for training, validation and test set

In 4.2.1 Dataset Preparation, the dataset is exported into a .CSV file and the .CSV file contains image URLs with scientific name and common name associated to the wildlife's and plants' image URLs. Hence, a further step is required to download the images from each of the given URL by programming a scraper that automates downloading of the images. Using this scraper, the exported .CSV file must be pre-processed before initiating the downloads. This involves removing the column headers and two specific data columns which are the scientific name and common name, resulting in a dataset populated exclusively with image URLs. Once the .CSV file is done with pre-processing, it could be uploaded into the scraper and the scraper will automatically download the images from the URLs into a specified local folder.

The images downloaded consists of wildlife and plants species with varying nature backgrounds and brightness of image. It is important to note that not every image shows the wildlife's full view, some images only contain a wildlife's body part, for example, legs and close-up of their body due to images commonly captured using trail cameras while some plants' images only show the leaf, anther, roots or stems.

### 4.2.3 CSV file conversion into JSONL file

Subsequent to 4.2.1 Dataset Preparation, the downloaded CSV file must be reviewed before further processing by checking for any unwanted columns or rows to prevent corrupted rows during the conversion process. A series of Python scripts will be programmed to automate the conversions. The phases of conversion include converting CSV file into JSON file, then converting JSON file into JSONL file.

```
import csv
```

```

import json

# Input CSV file
input_file = 'all_animal2.csv'
# Output JSON file
output_file = 'all_train.json'

# Columns to extract
columns_to_extract = ['image_url', 'scientific_name', 'common_name']

# Read the CSV and extract specific columns
data = []
with open(input_file, mode='r', encoding='utf-8') as csv_file:
    reader = csv.DictReader(csv_file)
    for row in reader:
        extracted_row = {col: row[col] for col in columns_to_extract if col
in row}
        data.append(extracted_row)

# Write the extracted data to a JSON file
with open(output_file, mode='w', encoding='utf-8') as json_file:
    json.dump(data, json_file, indent=4)

print(f"Data has been successfully converted to {output_file}")

```

*Figure 4.6: Convert CSV to JSON Python script.*

Figure 4.6 shows the Python script programmed to convert the downloaded CSV file into JSON file. The `input_file` variable specifies the path of the CSV file name in local drive and the `output_file` specifies the path where JSON file will be saved after successfully executing this script. The columns to be extracted is set to the header row from the CSV file which are `image_url`, `scientific_name` and `common_name`. Next, `data = []` establishes lists of empty data in the JSON file which will be appended when the CSV file is open and read in the line “with open(input\_file, mode = ‘r’, encoding = ‘utf-8’) as csv\_file:”. The object `csv.DictReader` reads each row of the CSV as a dictionary where for each row, it creates a new dictionary with only the specified columns. After every row’s `common_name` column is read, `data.append(extracted_row)` will append that dictionary to the data list. The next line of code will open the `output_file` in write mode and uses the `json.dump()` to write the data list to the JSON file. The purpose of `indent=4` is to make the JSON readable (pretty print). The last line of the code is a verification success message when the conversion is successfully done. The JSON file should present as in Figure 4.7.

```
[
  {
    "image_url": "https://static.inaturalist.org/photos/9/medium.jpg",
    "scientific_name": "Scutigera coleoptrata",
    "common_name": "House Centipede"
  },
  {
    "image_url": "https://inaturalist-open-
data.s3.amazonaws.com/photos/113/medium.jpg",
    "scientific_name": "Badumna longinqua",
    "common_name": "Grey House Spider"
  },
  {
    "image_url": "https://static.inaturalist.org/photos/20/medium.jpg",
    "scientific_name": "Cepaea nemoralis",
    "common_name": "Brown-lipped Snail"
  },
]
```

*Figure 4.7: The converted CSV file to JSON file output.*

The next phase is to convert the JSON file into a JSONL file.

```
import json

# Input JSON file
input_file = 'all_train.json'
# Output JSONL file
output_file = 'all_train.jsonl'

# Load the JSON data
with open(input_file, 'r', encoding='utf-8') as f:
    data = json.load(f)

# Write to JSONL format
with open(output_file, 'w', encoding='utf-8') as f:
    for item in data:
        json_line = json.dumps(item)
        f.write(json_line + '\n')

print(f"Successfully converted to {output_file}")
```

*Figure 4.8: Convert JSON to JSONL Python script.*

Figure 4.8 shows the JSON to JSONL conversion Python script. The variable `input_file` defines the path of the previously converted CSV to JSON file which was named `trained_output.json` while the `output_file` defines the path of the JSONL output file. The JSON file is open and set to read mode and loads the entire JSON structure. Next, the `output_file` is then opened in write mode and each of the item in the data list is looped through while the `json.dumps(item)` converts each dictionary into a JSON-formatted string. The file will be written following a newline (`\n`). A verification success message will be printed when the conversion is successfully done.

```
{
  "image_url": "https://static.inaturalist.org/photos/9/medium.jpg",
  "scientific_name": "Scutigera coleoptrata",
  "common_name": "House Centipede"
}
{
  "image_url": "https://inaturalist-open-data.s3.amazonaws.com/photos/113/medium.jpg",
  "scientific_name": "Badumna longinqua",
  "common_name": "Grey House Spider"
}
{
  "image_url": "https://static.inaturalist.org/photos/20/medium.jpg",
  "scientific_name": "Cepaea nemoralis",
  "common_name": "Brown-lipped Snail"
}
```

*Figure 4.9: The converted JSON to JSONL file output.*

Figure 4.9 shows the output file of the converted JSON to JSONL file. In this file, the data has not yet been structured according to the PaliGemma JSONL annotation format. It contains the `image_url` as the first field, then `scientific_name` and `common_name` separated into 2 fields. A further reformatting will be done to ensure that the dataset conforms to the PaliGemma JSONL annotation format.

Afterwards, a reformatting of the existing JSONL file is done to follow PaliGemma JSONL annotation format based on the examples in Roboflow<sup>4</sup>. The steps needed to reformat the JSONL file into PaliGemma JSONL is adding prompt prefix, combining suffix and changing the image URLs into the image file name downloaded into local drive.

```
import json

input_file = 'all_train.jsonl'          # original jsonl
output_file = 'all_added_prefix.jsonl'  # Output with prompt prefix

# Define the constant prompt prefix
prompt_prefix = "what is this animal's species?"
```

---

<sup>4</sup> <https://roboflow.com/formats/paligemma-jsonl>

```

with open(input_file, 'r', encoding='utf-8') as infile, \
    open(output_file, 'w', encoding='utf-8') as outfile:

    for line in infile:
        item = json.loads(line.strip())

        # Add the prompt
        item['prefix'] = prompt_prefix

        # Reorder keys if needed (optional)
        reordered = {
            "prefix": item["prefix"],
            "image_url": item.get("image_url", ""),
            "scientific_name": item.get("scientific_name", ""),
            "common_name": item.get("common_name", "")
        }

        # Write as a JSONL line
        outfile.write(json.dumps(reordered) + '\n')

print(f"Formatted JSONL saved as {output_file}")

```

*Figure 4.10: A Python script to add prefix into existing JSONL file.*

Figure 4.10 shows the Python script programmed to add prompt prefix into the JSONL file from the previous phase. The `input_file` in this script is the path to the existing JSONL file and `output_file` is the destination path of the to-be generated reformatted JSONL file. A variable called `prompt_prefix` is defined to a string “what is this animal’s species?”. This string variable will be appended as a new key-value pair to each line of the JSONL file making it a constant value of the dataset. Next, the `input_file` will be opened for reading and the `output_file` will be opened for writing. The next for loop reads the file line by line and parses each line into a Python dictionary while stripping any extra whitespace or newline characters. The `item['prefix'] = prompt_prefix` line adds a new key-value pair into each line of the JSONL file. The proper arrangement of the fields in the JSONL file will be prefix, suffix and image. Therefore, the fields will need rearrangement by using the dictionary called ‘reordered’ while `.get()` is used in case any field is missing and defaulting the field to an empty string. Finally, the `outfile.write(json.dumps(reordered) + '\n')` will convert the reordered dictionary to a JSON-formatted string and writing it to the new JSONL file followed by a newline. A success

verification message will be printed once the file has been successfully written and reformatting is successful.

```
{"prefix": "what is this animal's species?", "image_url":  
"https://static.inaturalist.org/photos/9/medium.jpg", "scientific_name":  
"Scutigera coleoptrata", "common_name": "House Centipede"}  
{"prefix": "what is this animal's species?", "image_url":  
"https://inaturalist-open-data.s3.amazonaws.com/photos/113/medium.jpg",  
"scientific_name": "Badumna longinqua", "common_name": "Grey House Spider"}  
{"prefix": "what is this animal's species?", "image_url":  
"https://static.inaturalist.org/photos/20/medium.jpg", "scientific_name":  
"Cepaea nemoralis", "common_name": "Brown-lipped Snail"}
```

*Figure 4.11: Added prefix into JSONL file.*

Figure 4.11 shows the output file (JSONL) of a reformatted existing JSONL file. In this figure, only 3 lines are displayed to illustrate a preview of the reformatted output. The prefix is successfully added and reordered as the first field in each JSONL line.

```
import json  
  
input_file = 'all_added_prefix.jsonl'           # current jsonl  
output_file = 'all_prefix_suffix.jsonl'         # New output with prefix and  
combined suffix  
  
# Define the constant prompt prefix  
prompt_prefix = "what is this animal's species?"  
  
with open(input_file, 'r', encoding='utf-8') as infile, \  
    open(output_file, 'w', encoding='utf-8') as outfile:  
  
    for line in infile:  
        item = json.loads(line.strip())  
  
        # Combine scientific_name and common_name into suffix  
        scientific = item.get("scientific_name", "")  
        common = item.get("common_name", "")  
        suffix = f"{scientific} ({common})" if common else scientific  
  
        # Construct new JSON object  
        new_item = {  
            "prefix": prompt_prefix,  
            "image": item.get("image_url", ""),  
            "suffix": suffix  
        }
```

```

        outfile.write(json.dumps(new_item) + '\n')

print(f"Final formatted JSONL saved as {output_file}")

```

*Figure 4.12: A Python script to combine suffix.*

Figure 4.12 shows a Python script written to combine the `scientific_name` and `common_name` fields in Figure 4.7, Figure 4.9 and Figure 4.11 into one combined field called the suffix. The `input_file` are the path to the previously generated JSONL file and `output_file` is the destination path of the JSONL file to be produced. The string variable `prompt_prefix` is maintained from a previous Python script. Next, the `input_file` is opened for reading while the `output_file` is opened for writing. The definition `item = json.loads(line.strip())` will strip newline characters and parses each JSON line into a Python dictionary. The next indented block of code fetches the `scientific_name` and `common_name` then combines them into a readable format. For instance, the format would be “`scientific_name (common_name)`” which will appear as such “`Loxodonta Africana (African Elephant)`” and if `common_name` is missing, it will be replaced with the `scientific_name` instead. Afterwards, a new JSON object will be built using a new dictionary called `new_item` while also maintaining the prefix as the first field. Finally, the `outfile.write(json.dumps(new_item) + '\n')` will convert the new dictionary to a JSON string and writes it as one line in the output JSONL file. After a successful combination of suffix and writing of a new JSONL file, a verification success message will be printed.

```

{"prefix": "what is this animal's species?", "image":
"https://static.inaturalist.org/photos/9/medium.jpg", "suffix": "Scutigera
coleoptrata (House Centipede)"}
{"prefix": "what is this animal's species?", "image": "https://inaturalist-
open-data.s3.amazonaws.com/photos/113/medium.jpg", "suffix": "Badumna
longinqua (Grey House Spider)"}
{"prefix": "what is this animal's species?", "image":
"https://static.inaturalist.org/photos/20/medium.jpg", "suffix": "Cepaea
nemoralis (Brown-lipped Snail)"}

```

*Figure 4.13: Combined suffix (scientific\_name & common\_name).*

Figure 4.13 shows the output file of the newly generated JSONL file containing the combined suffix from the field `scientific_name` and `common_name`. These 2 fields are successfully combined and follows the defined format in the Python script from Figure 4.12.

```

import json

```

```

input_file = 'all_prefix_suffix.jsonl'          # Your current JSONL
output_file = 'all_reordered.jsonl'           # Final output with reordered fields

with open(input_file, 'r', encoding='utf-8') as infile, \
    open(output_file, 'w', encoding='utf-8') as outfile:

    for line in infile:
        item = json.loads(line.strip())

        # Reorder the fields
        reordered = {
            "prefix": item.get("prefix", ""),
            "suffix": item.get("suffix", ""),
            "image": item.get("image", "")
        }

        outfile.write(json.dumps(reordered) + '\n')

print(f"Reordered JSONL saved as {output_file}")

```

*Figure 4.14: A Python script to reorder fields in JSONL file according to PaliGemma*

*JSONL annotation format.*

Figure 4.14 shows the Python script written to reorder the fields in the previously generated JSONL file to follow the PaliGemma JSONL annotation format from Roboflow. This Python script is done to reorder the fields in the existing JSONL file into prefix, suffix and image. This step is crucial because some ML models expects fields in a particular order and makes the file more consistent and readable. As usual, the `input_file` defines the path of the previously generated JSONL file and `output_file` defines the destination path of the new JSONL file to be generated. The `input_file` and `output_file` are both opened for reading and writing, respectively. To read one line at a time, the `item = json.loads(line.strip())` is defined while stripping any extra whitespace and converts the JSON string into a Python dictionary. Next, a dictionary is named 'reordered' is defined to specify the reordering of the fields in the JSONL file, arranging them in the order: prefix, suffix and image. Finally, the `outfile.write(json.dumps(reordered) + '\n')` converts the reordered dictionary into a JSON string and writes it to the output file, one line at a time. To ensure that the reordering is successful, a verification success message will be printed when the code is executed successfully.

```

{"prefix": "what is this animal's species?", "suffix": "Scutigera
coleoptrata (House Centipede)", "image":
"https://static.inaturalist.org/photos/9/medium.jpg"}
{"prefix": "what is this animal's species?", "suffix": "Badumna longinqua
(Grey House Spider)", "image": "https://inaturalist-open-
data.s3.amazonaws.com/photos/113/medium.jpg"}
{"prefix": "what is this animal's species?", "suffix": "Cepaea nemoralis
(Brown-lipped Snail)", "image":
"https://static.inaturalist.org/photos/20/medium.jpg"}

```

*Figure 4.15: The output file of the reordered JSONL fields.*

Figure 4.15 shows the output file of the newly generated JSONL file with proper ordering according to the PaliGemma JSONL annotation format. Now, the sole remaining action is to change the `image_url` field value into the actual file name of the downloaded images.

```

import json

# File paths
input_file = 'all_reordered.jsonl' # replace with the path to input file
output_file = 'all_numbering.jsonl' # replace with the desired output file

def add_numbering_to_jsonl(input_file, output_file):
    try:
        with open(input_file, 'r') as infile:
            lines = infile.readlines()

        # Open the output file for writing
        with open(output_file, 'w') as outfile:
            for image, line in enumerate(lines, 1): # Start numbering from
1
                # Parse the JSON line
                data = json.loads(line.strip())

                # Add the index to the dictionary
                data['image'] = image

                # Write the updated JSON with the new index to the output
file
                outfile.write(json.dumps(data) + '\n')

            print(f"Numbering added and saved to {output_file}")
    except Exception as e:
        print(f"Error: {e}")

```

```
# Run the function
add_numbering_to_jsonl(input_file, output_file)
```

*Figure 4.16: A Python script to convert image URLs into ascending numbers.*

Figure 4.16 shows a Python script programmed to convert the image URLs in the field `image_url` into ascending numbers. The reason for this is because the downloaded images are automatically named in ascending numerical according to its row number in the initial CSV file followed by a .jpg file extension. The Python script in Figure 4.16 is responsible with changing the image URLs into numbers according to their assigned file name. The initial step is to define the path of the `input_file` and `output_file`. A function called `add_numbering_to_json` is defined to handle reading, modifying and writing of the JSONL file. The code line `lines = infile.readlines()` reads all lines of the `input_file` into a list. Next, the `output_file` is opened for writing and `enumerate(lines, 1)` loops over the lines while also keeping a count starting at 1. The counter, `image`, becomes the new number assigned. The line `data['image'] = image` replaces the `image` field with the index number. Finally, the `outfile.write(json.dumps(data) + '\n')` serializes the modified dictionary into a JSON string and writes it as a new line in the output file. An error handling is added to detect and prints any error if a file is not found or if a JSON file is invalid. The following image shows the output as a result of successful execution of the Python script in Figure 4.16.

```
{"prefix": "what is this animal's species?", "suffix": "Scutigera
coleoptrata (House Centipede)", "image": 1}
{"prefix": "what is this animal's species?", "suffix": "Badumna longinqua
(Grey House Spider)", "image": 2}
{"prefix": "what is this animal's species?", "suffix": "Cepaea nemoralis
(Brown-lipped Snail)", "image": 3}
```

*Figure 4.17: The output file of converted image URLs.*

Figure 4.17 shows the output file of the converted image URLs in ascending numerical order. The `image_url` is changed to `image` instead and each line of the `image` field is changed to numbers.

```
import json

# Function to update the number field by appending ".jpg"
def add_jpg_suffix_to_jsonl(input_file, output_file):
    with open(input_file, 'r') as infile, open(output_file, 'w') as outfile:
        for line in infile:
```

```

# Load the current JSON object
data = json.loads(line.strip())

# Assuming the number you want to modify is under the key
'number'
    if 'image' in data:
        data['image'] = str(data['image']) + '.jpg' # Append ".jpg"
to the number

# Write the updated JSON object to the output file
json.dump(data, outfile)
outfile.write('\n')

# Example usage
input_file = 'all_numbering.jsonl'
output_file = 'all_final.jsonl'
add_jpg_suffix_to_jsonl(input_file, output_file)

```

*Figure 4.18: A Python script to add .jpg file extension into the image field.*

Figure 4.18 shows a Python script written to add a following .jpg file extension into the numbers added into the image field earlier. A function is defined and named `add_jpg_suffix_to_jsonl` with the `input_file` and `output_file` as its parameters. This function reads a JSONL file then adds .jpg to the image value and saves the result. The `input_file` and `output_file` are both opened for reading and writing, respectively. The for-loop loops through each JSONL line and strips any extra whitespace while loading it into a Python dictionary called `data`. Next, the image field is modified using an if conditional block to check if the image exists in the data list then `data['image'] = str(data['image']) + '.jpg'` converts the value to a string in case it is an integer and adds .jpg file extension at the end. Afterwards, the updated line is updated by serializing the updated dictionary back into JSON and writes it as a single line into the output line using the `json.dump(data, outfile)` and `outfile.write('\n')`, respectively. Consistently, the `input_file` is the path of the previously generated JSONL file and `output_file` is the destination path of the JSONL file to be generated. And finally, the function is called to generate the new JSONL file.

```

{"prefix": "what is this animal's species?", "suffix": "Scutigera
coleoptrata (House Centipede)", "image": "1.jpg"}
{"prefix": "what is this animal's species?", "suffix": "Badumna longinqua
(Grey House Spider)", "image": "2.jpg"}

```

```
{"prefix": "what is this animal's species?", "suffix": "Cepaea nemoralis  
(Brown-lipped Snail)", "image": "3.jpg"}
```

*Figure 4.19: The output file of the added .jpg file extension in the image field.*

Figure 4.19 shows the output file of the newly generated JSONL file. The .jpg file extension is successfully added into each line of the JSONL file in image field.

## 4.3 Development Environment Setup

### 4.2.1 Visual Studio Code

Visual Studio Code will be used as the main code editor of this project. In this software, the backend and frontend of the web application are developed by using Python's Django framework.

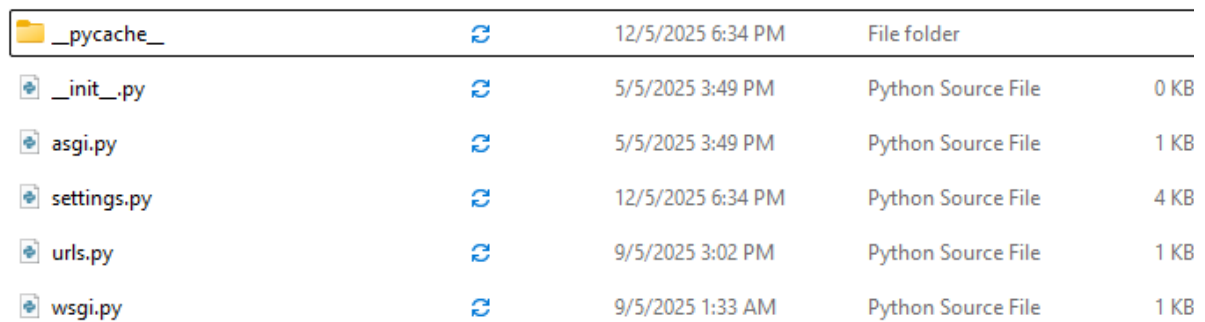
Python will be the main programming language used in this project because of its compatibility and usability when it comes to utilizing AI/ML models. This language will also be used for the web application development and integrating its framework, Django. Furthermore, a web scraper for iNaturalist dataset is also developed using Python and the dataset cleaning will be done manually by executing Python scripts. Python interpreter must be downloaded locally to have full control of the project's development through management of local directory, library and packages. This is to avoid any possible complications like crashing or unstable internet connection if an online compilers and Integrated Development Environment (IDE) is opted. Other than downloading a Python interpreter, a Python extension must be installed into Visual Studio Code to support a Python development environment. After installing Python interpreter and the necessary extensions, a directory for the project folder must be created for the purpose of efficient organisations and management of code files.

Through the project directory folder, a terminal can be directly accessed for commands especially when setting up a virtual environment for the project. A virtual environment is crucial for the project as it provides a container for the web application development to avoid possible version incompatibilities when using a global environment.

### 4.2.2 Django

Django is a Python framework used in this project development. The use of such framework will ease the web application development by reducing time in programming codes and functions from scratch. To start off with the web development, Django must be installed into a virtual environment of the project. This is to ensure that the project is not affected by the global environment.

To start the project, a designated folder is created to cater to the project's needs and requirements while also ease code files management. Once a designated folder is created, a command prompt terminal could be opened to start a Django project. By sending the command “django-admin startproject [name]”, this command will initialize a folder creation inside the directory used in the terminal, which is the main project folder and a subfolder which is a Django project folder will be created inside the initially created designated folder. By default, the new folder path will contain a Python file named “manage.py” and a project package containing “settings.py”, “urls.py”, “asgi.py” and “wsgi.py”. Figure 4.20 shows the directory of the newly created Django project folder.



|             |  |                   |                    |      |
|-------------|--|-------------------|--------------------|------|
| __pycache__ |  | 12/5/2025 6:34 PM | File folder        |      |
| __init__.py |  | 5/5/2025 3:49 PM  | Python Source File | 0 KB |
| asgi.py     |  | 5/5/2025 3:49 PM  | Python Source File | 1 KB |
| settings.py |  | 12/5/2025 6:34 PM | Python Source File | 4 KB |
| urls.py     |  | 9/5/2025 3:02 PM  | Python Source File | 1 KB |
| wsgi.py     |  | 9/5/2025 1:33 AM  | Python Source File | 1 KB |

*Figure 4.20: Django project folder.*

Next, an app must be created because it is needed for the main development of the web application's frontend and backend aspects. ‘App’ in this context refers to a subfolder of the main project folder which is a Django app folder. Using the command “django-admin startapp [name]”, the subfolder will be created and by default, this subfolder will contain “admin.py”, “apps.py”, “models.py”, “tests.py” and “views.py”. Figure 4.21 shows the Django app folder.

|             |   |                   |                    |      |
|-------------|---|-------------------|--------------------|------|
| __pycache__ | 🔗 | 14/5/2025 8:52 AM | File folder        |      |
| migrations  | 🔗 | 12/5/2025 7:03 PM | File folder        |      |
| __init__.py | 🔗 | 5/5/2025 3:54 PM  | Python Source File | 0 KB |
| admin.py    | 🔗 | 12/5/2025 7:07 PM | Python Source File | 1 KB |
| apps.py     | 🔗 | 9/5/2025 1:34 AM  | Python Source File | 1 KB |
| models.py   | 🔗 | 12/5/2025 7:05 PM | Python Source File | 1 KB |
| tests.py    | 🔗 | 5/5/2025 3:54 PM  | Python Source File | 1 KB |
| urls.py     | 🔗 | 12/5/2025 6:38 PM | Python Source File | 2 KB |
| views.py    | 🔗 | 14/5/2025 8:52 AM | Python Source File | 9 KB |

*Figure 4.21: Django app folder.*

By default, a Django app folder does not include a “urls.py” module. Hence, to establish URL routing at the app level, the “urls.py” file must be manually duplicated from Django’s project folder into Django’s app folder. In the project’s “urls.py”, import the include function from django.urls to define a path that links to the Django’s app URL configurations using the include() method. Subsequently, in Django’s app’s “urls.py”, URL patterns must be specified to map to the corresponding views and render the associated HTML templates. In Django, templates represent the HTML structure of the application’s front-end.

In Django’s project’s “settings.py”, add Django app into application definition of installed apps to allow Django to recognize and integrate created Django app into the project which is as shown in Figure 4.22.

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'app_name.apps.MainConfig',
]
```

*Figure 4.22: Application definition*

Following the completion of Django’s environment setup, the front-end of forms will be prepared in Hypertext Markup Language (HTML). In Django, these HTML files will be gathered in the same folder called “templates” and Django templates are text files made to separate the front-end interface from the code logics. Multiple pages are needed for the web application such as the homepage (chatbot), login page, sign up page, response page, history page and help page.

In this project, user authentication modules will be developed first which includes the login page and sign-up page. For this module, six templates are prepared in HTML format namely “login.html”, “register.html”, “forgot\_password.html”, “reset\_password.html”, “password\_reset\_sent.html” and “index.html”.

Before applying any of the templates in the Django app, the template path must be configured in “settings.py” to ensure that Django is fetching the correct templates. The configuration could be done by simply changing the value of DIRS to the path of the templates folder in the project. In the case of this project, the templates directory is the same as Django app and project folder. Generally, the directory of the templates folder would be configured as ‘BASE\_DIR/templates’ however this caters to the project’s development environment and management. This configuration could be seen in Figure 4.23.

```
TEMPLATES = [  
    {  
        'BACKEND': 'django.template.backends.django.DjangoTemplates',  
        'DIRS': ['templates'],  
        'APP_DIRS': True,  
        'OPTIONS': {  
            'context_processors': [  
                'django.template.context_processors.request',  
                'django.contrib.auth.context_processors.auth',  
                'django.contrib.messages.context_processors.messages',  
            ],  
        },  
    },  
]
```

*Figure 4.23: Template directory configuration.*

Once the configurations are done, the templates can be created in HTML format into the templates folder. A further explanation of the web application development will be done in 4.6 Web Application Implementation.

### 4.2.3 MySQL

MySQL is a relational database management system that will be utilized in this project. This project's database will be using MySQL to store user data and PaliGemma's generated output. Additionally, the database will also be used to store users' feedback of the web application.

A database is initially created to accommodate the web application's data by writing a command into MariaDB's terminal opened through XAMPP. The command used to create a database is "create database [name]" then by clicking 'Enter', a database is successfully created.

The benefit of Django framework is that it comes with a default built-in database called SQLite to ease database management. However, SQLite only supports applications that are predominantly read-only or require a smaller installation footprint. Therefore, MySQL is needed for this project because of larger data capacity. To set up XAMPP's MySQL into Django, the project folder's settings.py must be configured to suit the database needs especially when using a database management system other than SQLite, in this case, it is MySQL. Figure .. below shows the configuration needed to apply XAMPP's MySQL into Django's environment. As seen in Figure 4.24, the default database of Django is SQLite therefore a configuration is done for MySQL connection by adding a database named 'users' with engine, host, database name, password, root user and port as the parameters.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    },
    'users': {
        'ENGINE': 'django.db.backends.mysql',
        'HOST': '127.0.0.1',
        'NAME': 'paligemma',
        'PASSWORD': '',
```

```

        'USER': 'root',
        'PORT': '3306'
    }
}

```

*Figure 4.24: Database configuration for Django.*

Next, because of Django's ability to write into database similarly to MariaDB's scripting, the tables are created by writing a Django Python script into app folder's models.py. In Django terms, database tables are called models for easy distinction. Figure 4.25 shows the Django Python script written for table creation. Based on Figure 4.25, each class contains a table's attribute and there are 5 tables created into the database namely Prompt, Response, Profile, History and PasswordReset.

```

class Prompt(models.Model):
    promptID = models.AutoField(primary_key=True)
    imagePrompt = models.ImageField(upload_to = 'uploads/')
    textPrompt = models.CharField(max_length=255, blank=True, null=True)

    def __str__(self):
        return f'Prompt {self.promptID}'

class Response(models.Model):
    responseID = models.AutoField(primary_key=True)
    response = models.CharField(max_length=50)
    feedback = models.IntegerField()
    prompt = models.ForeignKey(Prompt, on_delete=models.CASCADE)

    def __str__(self):
        return f'Response {self.responseID}'

class Profile(models.Model):
    userID = models.AutoField(primary_key=True)
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    prompt = models.ForeignKey(Prompt, on_delete=models.SET_NULL, null=True,
blank=True)
    response = models.ForeignKey(Response, on_delete=models.SET_NULL,
null=True, blank=True)

    def __str__(self):
        return f'{self.user.username} Profile'

class History(models.Model):

```

```

historyID = models.AutoField(primary_key=True)
user = models.ForeignKey(User, on_delete=models.CASCADE)
prompt = models.ForeignKey(Prompt, on_delete=models.CASCADE)
response = models.ForeignKey(Response, on_delete=models.CASCADE)

def __str__(self):
    return f'History {self.historyID}'

class PasswordReset(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    reset_id = models.UUIDField(default=uuid.uuid4, unique=True,
editable=False)
    created_when = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return f"Password reset for {self.user.username} at
{self.created_when}"

```

*Figure 4.25: Model creation for paligemma database.*

Once the models' script is written, a migration must be made into the database to initialize the table creation by sending a command using Visual Studio Code's built-in terminal. Before sending the command, the correct directory to run it in is the Django app folder by simply opening the directory in the terminal and send the command "cd [django\_project\_name]". The next command to send after changing the directory is "python manage.py makemigrations" to initialize the database migrations in Django's environment. Afterwards, to migrate all the tables into XAMPP's MySQL, the command "python manage.py migrate --database=users" is sent through the terminal. Once these commands are sent, a migration folder will be created under the app folder and two python files will be created namely `_init_.py` and `0001_initial.py`.

By opening localhost/phpMyAdmin through XAMPP, the database must be checked to ensure that the tables are properly migrated into XAMPP's MySQL database. The database is finally set up when the required steps are done.

#### 4.2.4 Google Colab

Google Colab is used in this project to fine-tune the pre-trained PaliGemma model on custom datasets. The environment must be setup in Google Colab to ease the fine-tuning process by installing dependencies and saving secret keys in Colab environment. Simplistically, the workflow of the fine-tuning process will be illustrated in Figure 4.26.

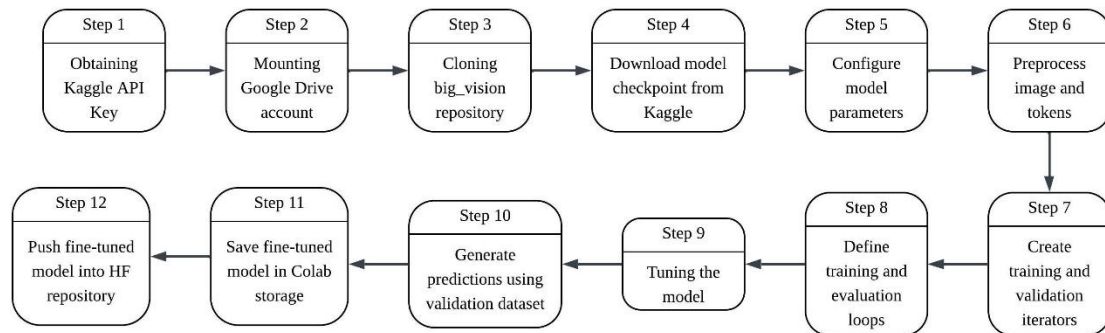


*Figure 4.26: Fine-tuning process.*

Step 1: Obtaining Kaggle API key

Step 2: Mounting Google Drive account

Step 3: Cloning big\_vision repository and install dependencies

Step 4: Download model checkpoint from Kaggle

Step 5: Configuring model parameters

Step 6: Preprocessing image and tokens

Step 7: Create training and validation iterators

Step 8: Define training and evaluation loops

Step 9: Tuning the model

Step 10: Generating predictions using validation set

Step 11: Saving the fine-tuned model in Google Colab's temporary storage

Step 12: Pushing the saved fine-tuned model into HuggingFace's repository

#### 4.2.5 HuggingFace

The fine-tuned model's deployment will be done in HuggingFace to reduce cost. This is because a mock deployment was previously done in Azure Machine Learning Studio to use the services while also using Azure's Student Credit given by UNIMAS. However, a large capacity storage is needed in Azure due to PaliGemma's large size and it is expensive. Because of the cost-ineffectiveness, this platform will not be used and considered for future works. Instead, HuggingFace will be used to upload the fine-tuned PaliGemma model as a cheaper alternative.

HuggingFace provides an inference service called Inference Endpoint that deploys an existing model in HuggingFace's models database. As an experiment, PaliGemma's pre-trained model with 224 pixels resolution was deployed using Inference Endpoint service to evaluate the performance of the inference service and decide whether it is suitable for use and able to accommodate PaliGemma with less cost and effort.

#### 4.4 Deployment of Finetuned PaliGemma Model

##### 4.4.1 Saving Finetuned PaliGemma Model into Google Colab

```
import os

TARGET_MODEL_DIR = f"/ft-model-v1"
TARGET_MODEL_PATH = f"{TARGET_MODEL_DIR}/paligemma-3b-pt-224.fl16.npz"

os.makedirs(TARGET_MODEL_DIR, exist_ok=True)
```

*Figure 4.27: Creating a directory for model download.*

When models are trained on Google Colab, it is not automatically downloaded into a Colab storage. Therefore, a download initialization must be done to download the model into Colab's local storage by implementing a code cell in Figure 4.27. Based on Figure 4.27, a target directory is defined along with the target directory path. TARGET\_MODEL\_DIR has a string value of "/ft-model-v1" which initiates a folder creation in the Colab's root directory while TARGET\_MODEL\_PATH has a string value of "{TARGET\_MODEL\_DIR}/paligemma-3b-

pt-224.fl6.npz which defines the directory where the fine-tuned model will be downloaded into. The fine-tuned model is saved in a NumPy-compressed format (.npz) to save multiple NumPy arrays to a single file which is commonly used in model storage.

```
flat, _ = big_vision.utils.tree_flatten_with_names(params)
with open("/ft-model-v1/fine-tuned-PaliGemma2-3b-pt-224.fl6.npz",
"wb") as f:
    np.savez(f, **{k: v for k, v in flat})
```

*Figure 4.28: Saving the fine-tuned parameters of the model.*

Figure 4.28 shows a code cell initializing the model download using a utility function from the `big_vision` library called “`tree_flatten_with_names`” that takes a structured parameter tree then flattens it into a list of key-value pairs. Next, the target file will be opened in write binary mode which means the file will be overwritten if it already exists or created if it does not. Finally, `np.save()` is a NumPy function that saves multiple arrays into a single .npz file to allow for reloading when needed.

#### 4.4.2 Upload Finetuned Model to HuggingFace

```
!pip install -q huggingface_hub
```

*Figure 4.29: Install huggingface\_hub library.*

Once the model is successfully downloaded into Colab’s storage, the next step is to push the downloaded model into HuggingFace repository. A HuggingFace library must be installed before pushing the model into its repository by implementing a pip install in Google Colab using the command in Figure 4.29.

```
from huggingface_hub import notebook_login
notebook_login()
```

*Figure 4.30: Initialize login into HuggingFace.*

Next, a login into HuggingFace are initialized to allow access into a HuggingFace account for a repository commit by running the code cell in Figure 4.30. This will trigger a login with username and password input to HuggingFace in Google Colab.

```
from huggingface_hub import create_repo, upload_folder

# Replace with your actual Hugging Face username and desired model
name
username = "makcikkado"
model_name = "paligemma-ft-v1" # any name you want

# Create the repo (you can set private=True if needed)
# create_repo(repo_id=f"{username}/{model_name}", private=False)

# Upload the entire model folder (which includes config, model
weights, etc.)
upload_folder(
    repo_id=f"{username}/{model_name}",
    folder_path="/ft-model-v1", # your saved model directory
    path_in_repo=".",          # upload everything to the root of the
repo
    commit_message="Upload fine-tuned model from Colab"
)
```

*Figure 4.31: Upload downloaded fine-tuned model to HuggingFace Model hub.*

Prior to initiating the commit, a model repository must be created in HuggingFace to allow a smooth commit into its repository. However, if a model repository is not created yet, the function `create_repo` will help with a model repository creation through Google Colab. Functions like `create_repo` and `upload_folder` must be imported from the `huggingface_hub` library installed initially through Figure 4.29 and defining the account's username along with the model's name to start the commit.

Next, the `upload_folder` method is utilized to trigger the upload while also defining the repository ID created in HuggingFace, the folder path, the path to upload it to in the repository and a commit message. These definitions contain the value needed to upload the model. Once the code cell in Figure 4.31 has been successfully executed, the model repository in HuggingFace must be checked to confirm a successful upload of the model's weights.

## **4.5 Web Application Deployment**

The deployment of PaliGemma web application involves three main components which are database hosting, web hosting and inference hosting. Each component is hosted on a dedicated platform that best support its requirements. The platforms to be used are Render, Clever Cloud and HuggingFace Inference Endpoint.

### **4.5.1 Render**

The web application will be which was developed using Django will be deployed on Render which is a modern cloud platform that supports automatic deployment from GitHub repositories and caters to Django-based projects. Render handles the backend logic, user interface rendering, static file serving and routing. This allows the application to be continuously updated and deployed with minimal downtime.

Render also has a free subscription plan that serves a memory with a RAM of 512MB which is sufficient for a small-scale project. This free subscription plan helps with minimizing financial burden and enables a successful deployment for testing purposes.

In 4.5.1.1 Render Configuration, the web application deployment process will be thoroughly explained.

#### **4.5.1.1 GitHub Repository**

Before deploying the web application on Render, the project file that is stored locally must be uploaded to a GitHub repository to allow Render to deploy GitHub commits. In GitHub repository, the project folder must be structured properly to ensure that Render can execute build commands smoothly without error. The repository structure will be visualized in Figure 4.31.

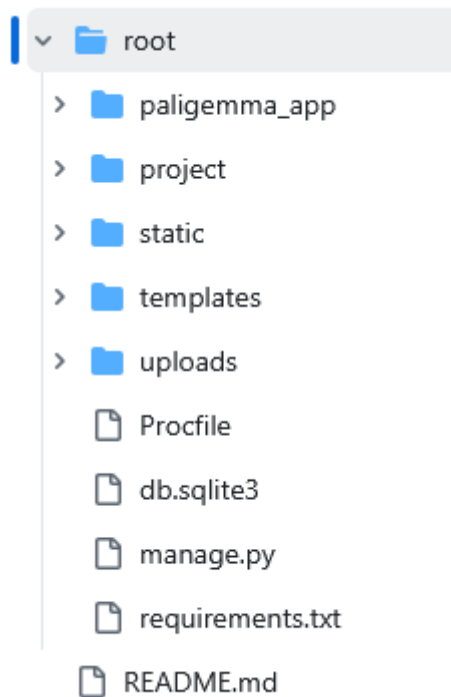


*Figure 4.32: GitHub repository structure.*

In Figure 4.32, the repository contains all the required folders of a Django-based application namely the Django app (paligemma\_app), the Django project (project), a static file (style.css) contained in static folder, a templates folder containing the web application's frontend and manage.py which is a Python script that executes in Render's build command. However, Render requires additional files for the app building which are 'Procfile' and 'requirements.txt'. Procfile is a special single line plain-text file that explicitly declares what command should be executed to start the application on the hosting platform. The content of Procfile is as shown in Figure 4.33.

```
1  web: gunicorn root.wsgi:application
```

*Figure 4.33: Procfile content.*

Figure 4.33 shows the one-line command in Procfile. In this project, the Procfile contains a single line that starts the Django server using Gunicorn which is a production-grade WSGI HTTP server for Python web applications. This ensures that the application runs efficiently in

a cloud environment. The command follows the format “web: gunicorn project.wsgi” where project is the name of the Django project directory. This file is essential for platforms like Render to recognize and correctly launch the Django application.

While the ‘requirements.txt’ file is an important component of the PaliGemma web application deployment process. It contains a list of all Python packages and libraries required by the project with their specific versions. The file includes core dependencies such as Django, MySQL client libraries for database connectivity, Gunicorn for serving the application in production and the requests and transformers libraries used for API calls to the HuggingFace Inference Endpoint. By including this file, Render can automatically install all required dependencies in the virtual environment during the build process and ensures consistency between development and production environments. This simplifies the setup and helps prevent compatibility issues due to missing packages or version mismatches. Figure 4.34 shows the content of the file ‘requirements.txt’.

---

```
1  asgiref==3.8.1
2  certifi==2025.4.26
3  charset-normalizer==3.4.2
4  Django==4.2
5  idna==3.10
6  mysqlclient==2.2.7
7  pillow==11.2.1
8  python-dotenv==1.1.0
9  pytz==2025.2
10 requests==2.32.4
11 sqlparse==0.5.3
12 tzdata==2025.2
13 urllib3==2.4.0
14 gunicorn
15 dj-database-url
16 whitenoise
```

*Figure 4.34: Contents of ‘requirements.txt’.*

#### **4.5.1.2 Render Configuration**

When the project is ready to be deployed in the Render platform, an important part of the configuration includes the build command and environment variables setting. The build

command essentially contains all the command to be executed when initializing deployment which is shown in Figure 4.34.

```
1  pip install -r requirements.txt
2  python manage.py makemigrations
3  python manage.py migrate --noinput
4  python manage.py collectstatic --noinput
```

*Figure 4.35: Build commands for Render.*

The commands in Figure 4.35 will be executed when deployment is initialized in Render. The command “pip install -r requirements.txt” will install all necessary dependencies listed in ‘requirements.txt’. To initialize the database tables, the command “python manage.py makemigrations” is needed to be able to migrate the database using the command “python manage.py migrate”. While the command “python manage.py collectstatic” is used to collect all static files from the individual Django app directories and third-party packages. By running these commands, the system ensures that all required backend and frontend assets are accessible and correctly loaded in the deployed web application.

Additionally, environment variables are added to configure database connections and API access. Once the configuration is completed, Render automatically builds and deploy the application and provides a public URL for access. The platform also enables continuous deployment and allows future updates pushed to GitHub to be automatically reflected in the live application.

#### **4.5.2 MySQL**

To host the MySQL database for PaliGemma web application, Clever Cloud was selected due to Render’s lack of built-in MySQL support. Clever Cloud is a platform-as-a-service (PaaS) provider that allows easy deployment and management of databases. Clever Cloud was used to provide a remote MySQL instance for the PaliGemma web application. The process involved creating a MySQL add-on in Clever Cloud to retrieve the auto-generated database credentials and configure Render to connect using these credentials. The credentials are stored in Render’s environment variables configuration.

Moreover, a support request was submitted to Clever Cloud to whitelist Render’s public IP address to allow remote access to the database. Once the connection was established, a database migration was applied to set up the database schema. This configuration enables efficient data storage and allows for retrieval of user submissions and model responses.

### 4.5.3 HuggingFace Inference Endpoint

The core functionality of the web application which is image and text-based reasoning is powered by a pretrained PaliGemma model trained on a mix of datasets. In this project, a “mix” model of PaliGemma was used to integrate with the Django backend. This “mix” model is a fine-tuned version of PaliGemma done by Google developers. The model is then deployed using HuggingFace’s Inference Endpoint service. With this approach, the model is accessed via a secure RESTful API without the need to host or manage GPU infrastructure directly on the web server. After deploying the model to the inference endpoint, the endpoint will then provide a persistent URL that accepts JSON payload consisting of base64-encoded image and a text prompt. Upon receiving a request, the endpoint returns the model’s generated response that is then processed and displayed by the Django application. Figure 4.36 shows the inference endpoint’s dashboard.

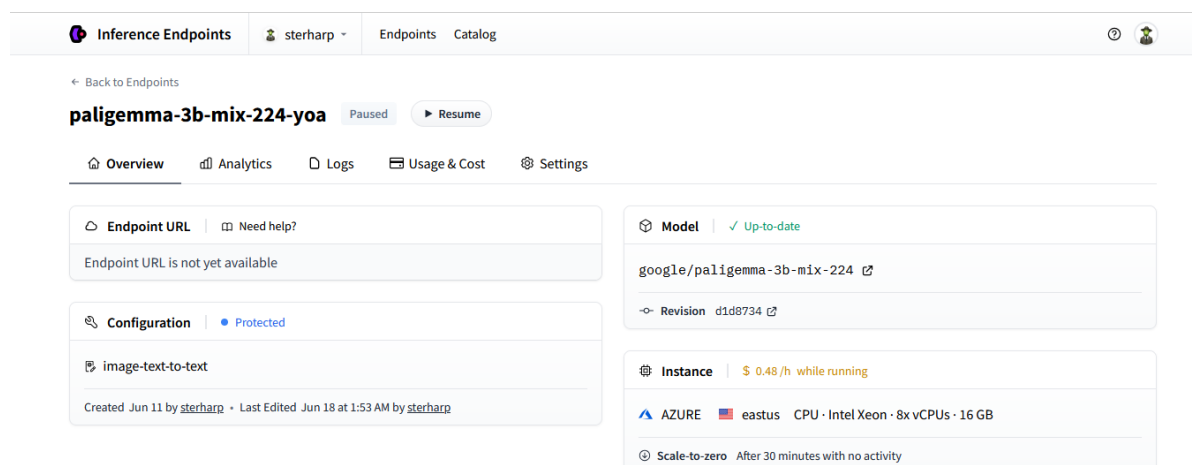


Figure 4.36: HuggingFace Inference Endpoint dashboard.

This method of AI model deployment allows a scalable and modular approach to integrating AI models into web applications. It separates inference from the main web server to ensure

faster development and testing cycles and utilizes HuggingFace's optimized backend infrastructure to handle heavy AI computation.

By combining Render, Clever Cloud and HuggingFace, the deployment ensures system reliability and ease of maintenance. Each component is optimized for its respective task and contributes to a robust and responsive web application architecture.

#### 4.5.4 Web Application Interface

##### 4.5.4.1 Non-logged In User

After all necessary deployment is carried out, the web application could be accessed through this link(insert footnote here). When clicking on the link, users will be met with a main page of the web application as shown in Figure 4.37.

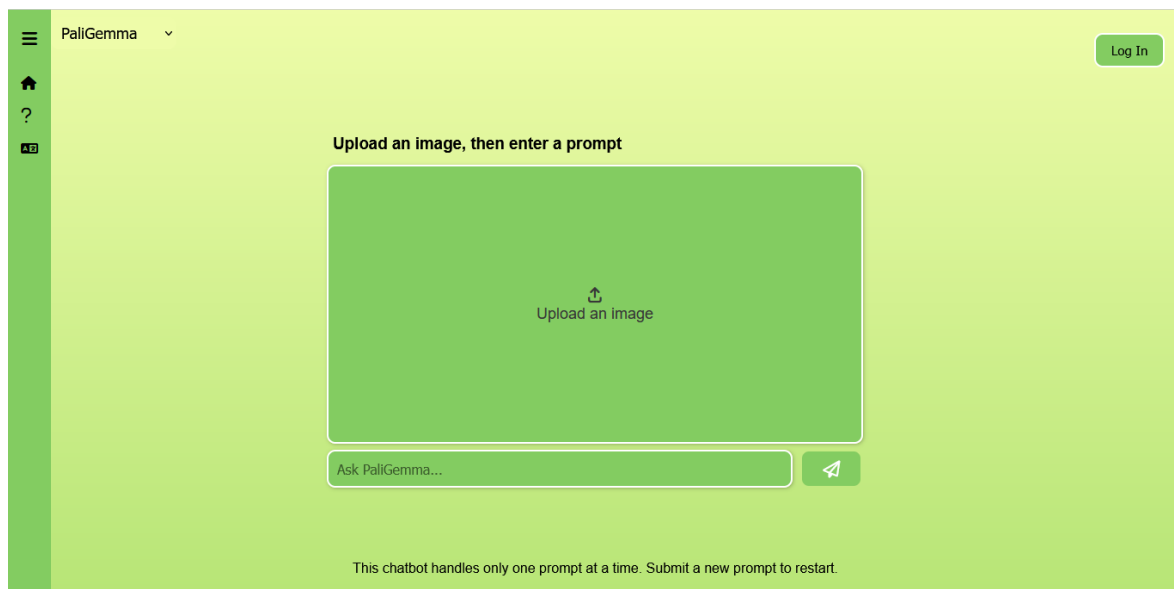


*Figure 4.37: Main page.*

Figure 4.37 shows the main page of the project's web application. On the top right corner, a log in button can be seen and it leads to a login page of the web application. Users can also submit a prompt without signing up and logging in the system. However, any prompt or response generated without a user's ID are not saved into the system and progress will be lost once the web application is refreshed.

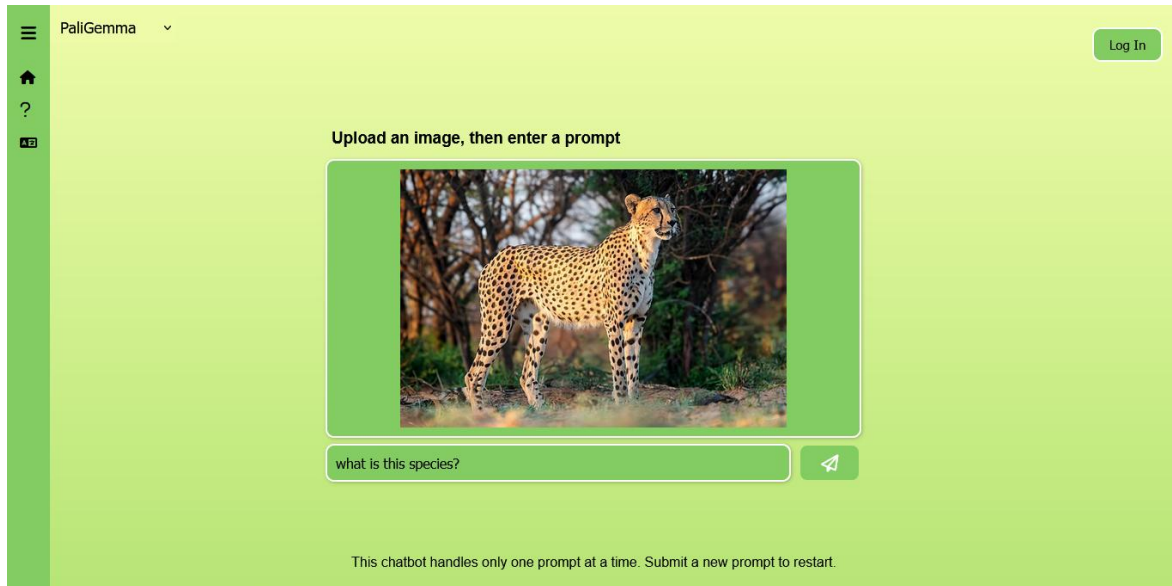


*Figure 4.38: Image preview after upload.*

Figure 4.38 shows a user uploading an image and typing a prompt. The web application will automatically preview the image to be sent to ensure the user is aware of what is being sent. A submit button next to the text box will redirect user to a page where the model's response will be displayed.

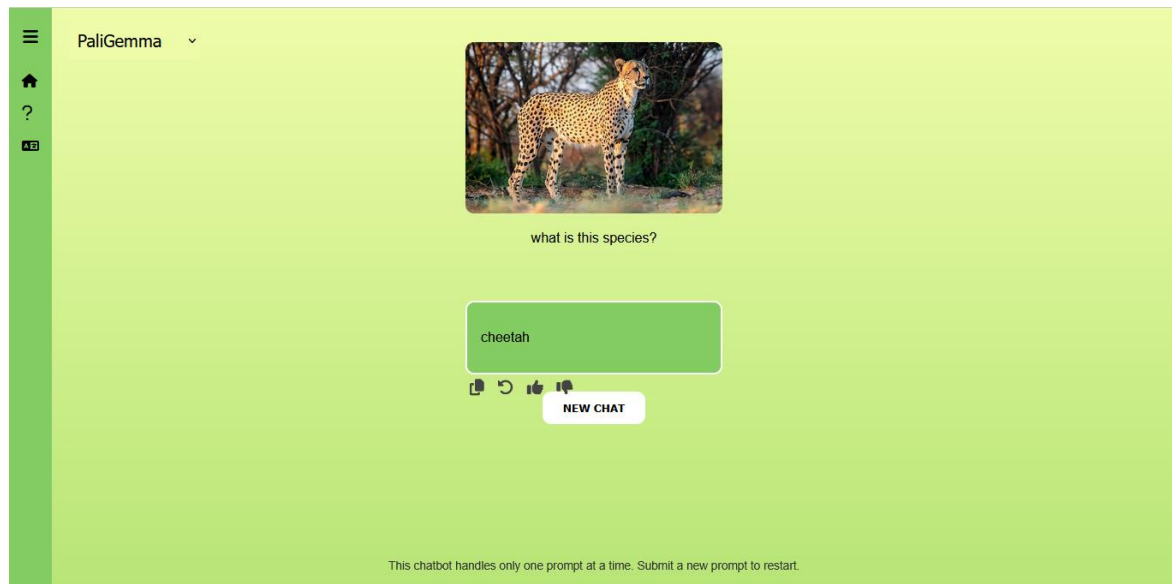


*Figure 4.39: Response page.*

Figure 4.39 demonstrates the model's generated response by displaying it in the response page. In this web page, the user's uploaded image and prompt will be displayed atop

of the response box that contains the model’s response to imitate a conversation situation but in a more organized manner catered to the purpose of this project. Below the right corner of the response box are icons that can be toggled, whether to copy the response, give a thumbs up or down to provide feedback on the generated response. A ‘New Chat’ button at the bottom of the response box will trigger a pop-up overlay as in Figure 4.40.

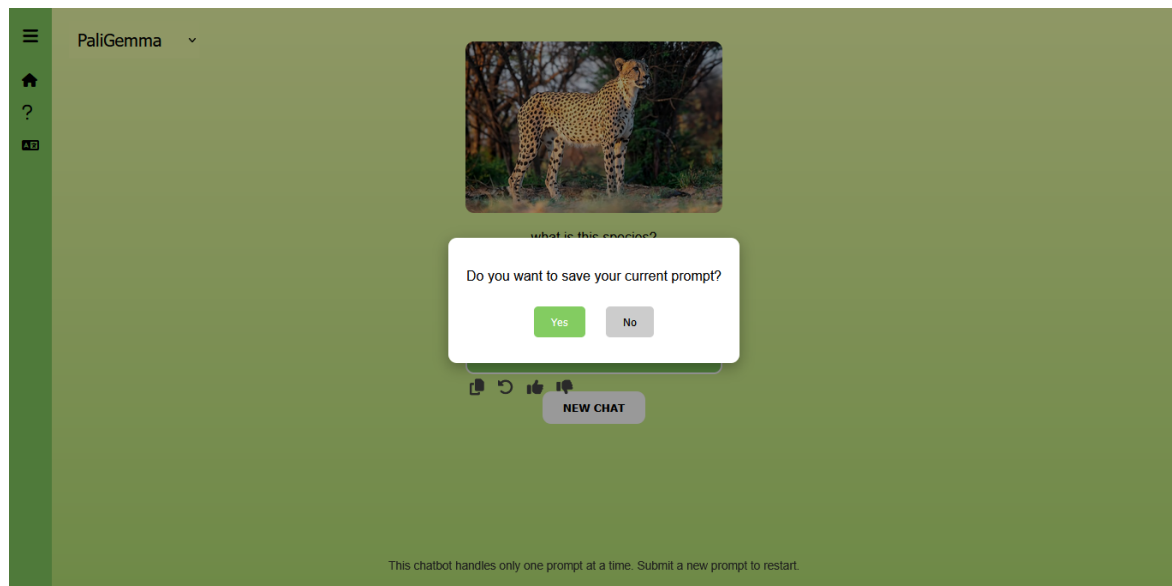


*Figure 4.40: First pop-up overlay.*

Figure 4.40 presents the triggered pop-up overlay showing a question to the user “Do you want to save your current prompt?” with a “Yes” and “No” option, after clicking on the ‘New Chat’ button. The option “Yes” will trigger another pop-up overlay as shown in Figure 4.41 while the option “No” will trigger the third pop-up overlay demonstrated in Figure 4.42.

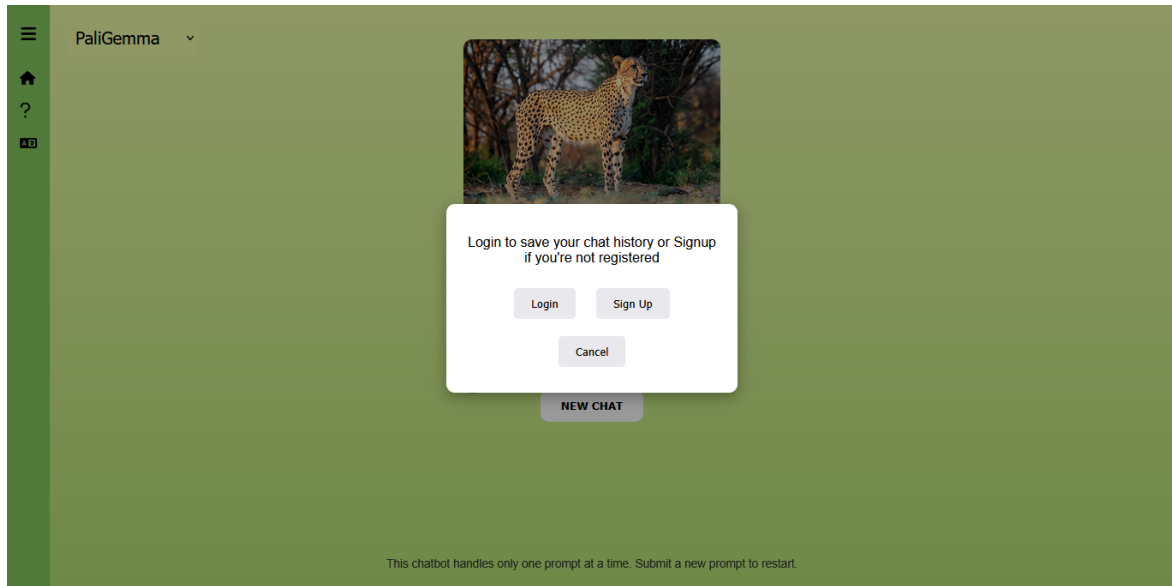


*Figure 4.41: “Yes” option pop-up overlay.*

Based on Figure 4.41, the second pop-up overlay presents three options for the user to choose from. “Login” will redirect to a login page, “Sign Up” will redirect to a register page and “Cancel” will close the pop-up overlay.

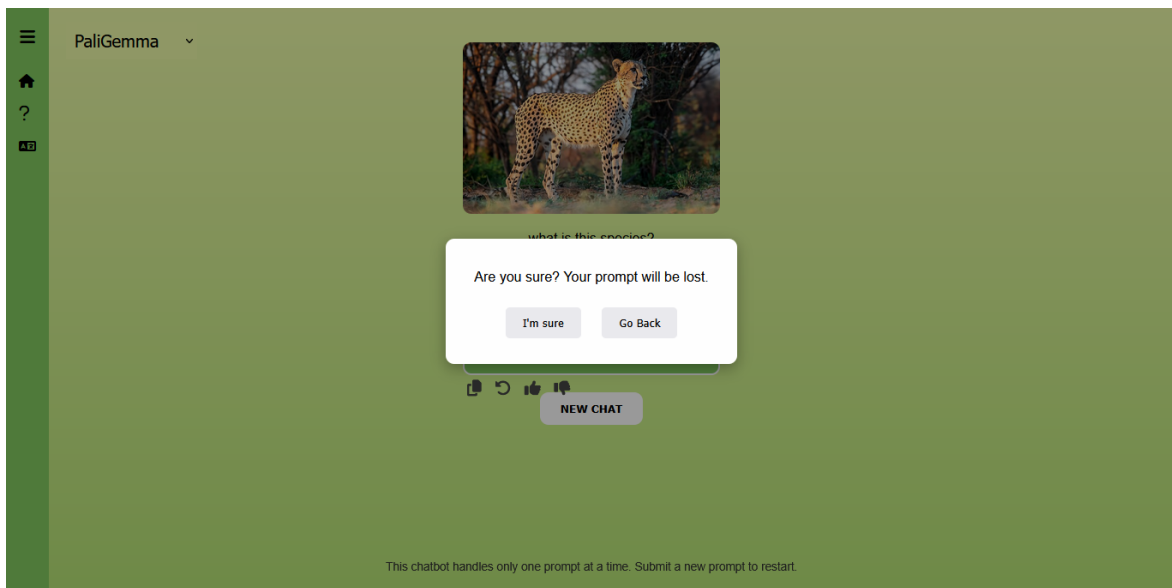


*Figure 4.42: “No” option pop-up overlay.*

As shown in Figure 4.42, the “No” option from Figure 4.40 will trigger the pop-up overlay in Figure 4.42. The pop-up presents a question “Are you sure? Your prompt will be

lost.” to the user. The option “I’m sure” will close the pop-up overlay while the option “Go Back” will return to the pop-up overlay in Figure 4.40.

#### 4.5.4.2 Logged-In User

As for a logged-in user, a click on the “Log In” button in the top right corner of Figure 4.37 will redirect user to a login page. From then on, users will have access to more features of the web application such as the “Chat History” feature. In Figure 4.43, the interface of the login page will be shown after clicking on the “Log In” button.

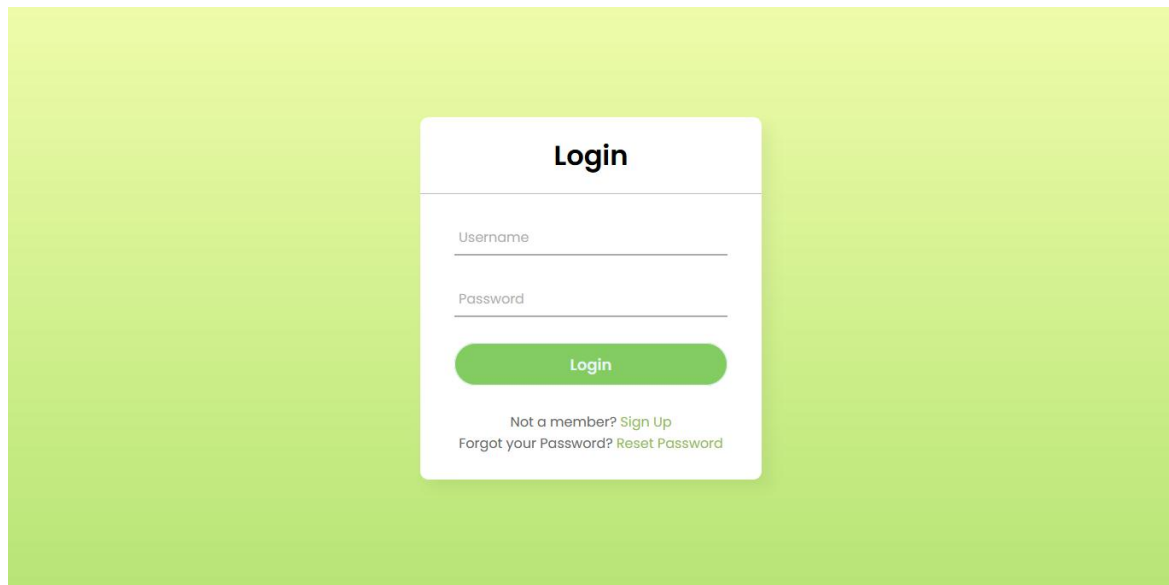


*Figure 4.43: Login page.*

Figure 4.43 shows the login page of the system. Registered users will be able to log in to the system and access more features immediately. However, unregistered users will need to click on the “Sign Up” link at the bottom of the “Login” button. This will redirect users to a register page to allow users to register as a user of the system. The register page is as shown in Figure 4.44.

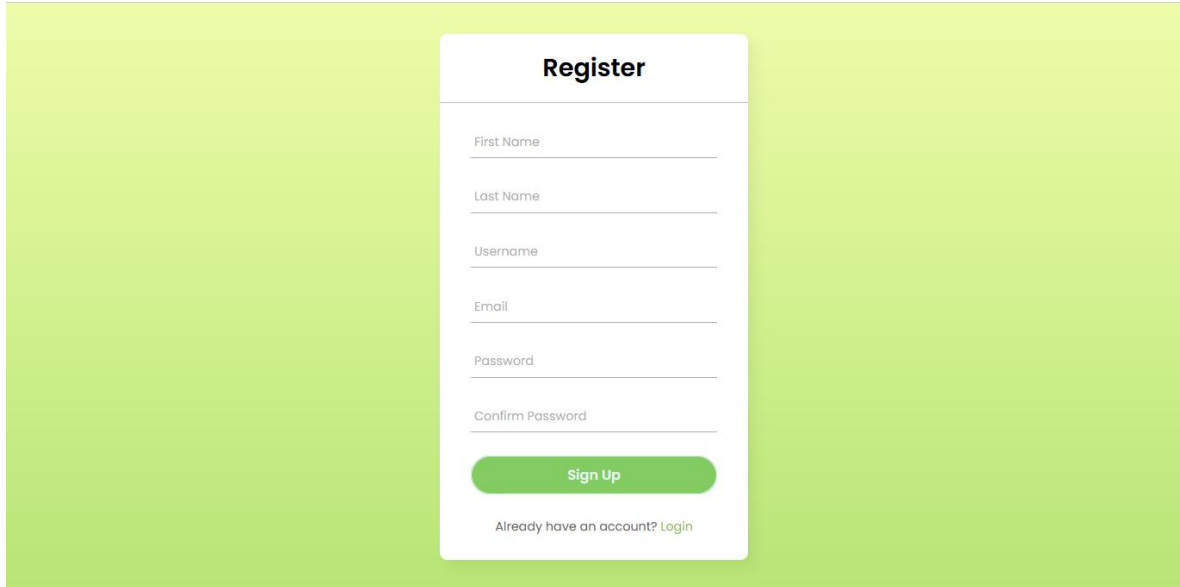
A registration form titled "Register" is centered on a light green background. The form is a white card with rounded corners. It contains six text input fields: "First Name", "Last Name", "Username", "Email", "Password", and "Confirm Password". Below these fields is a green "Sign Up" button. At the bottom of the card, there is a link that says "Already have an account? Login".

*Figure 4.44: Register page.*

Figure 4.44 shows the register page for users to sign up as a user of the system. The form requires user's first name, last name, username, e-mail and password. A text field for confirming password is also prepared to ensure that user entered their desired password correctly. Next, the button "Sign Up" will send the form to the database and create the user.

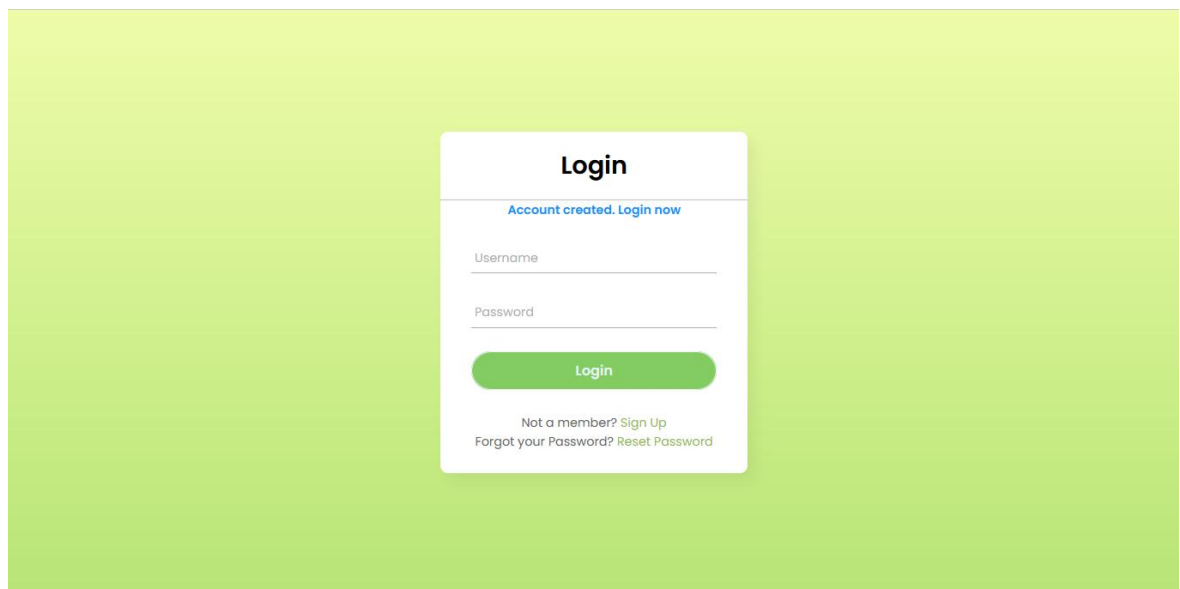
A login form titled "Login" is centered on a light green background. The form is a white card with rounded corners. At the top, there is a blue message: "Account created. Login now". Below this are two text input fields: "Username" and "Password". A green "Login" button is positioned below the fields. At the bottom of the card, there are two links: "Not a member? Sign Up" and "Forgot your Password? Reset Password".

*Figure 4.45: Redirect to login page.*

After a user has successfully registered an account, they will be redirected to the login page again with a validation success message shown to indicate that a user has successfully

registered an account as in Figure 4.45. User can then enter their login credentials and will be redirected to a home page shown in Figure 4.46.

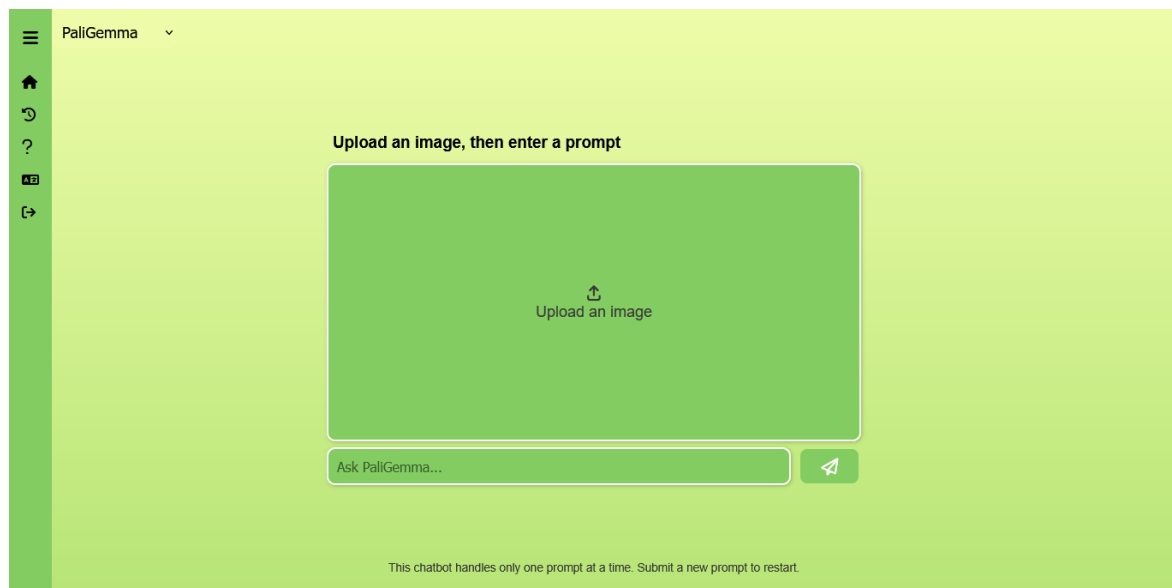


*Figure 4.46: Home page.*

Figure 4.46 shows the home page interface for logged-in users. The sidebar on the left of the page shows multiple icons that defines the additional features for a logged-in user such as the “Home” icon, “Chat History” icon and a “Logout” icon. The “Log In” button initially present in Figure 4.40 has been removed once a user is logged in.

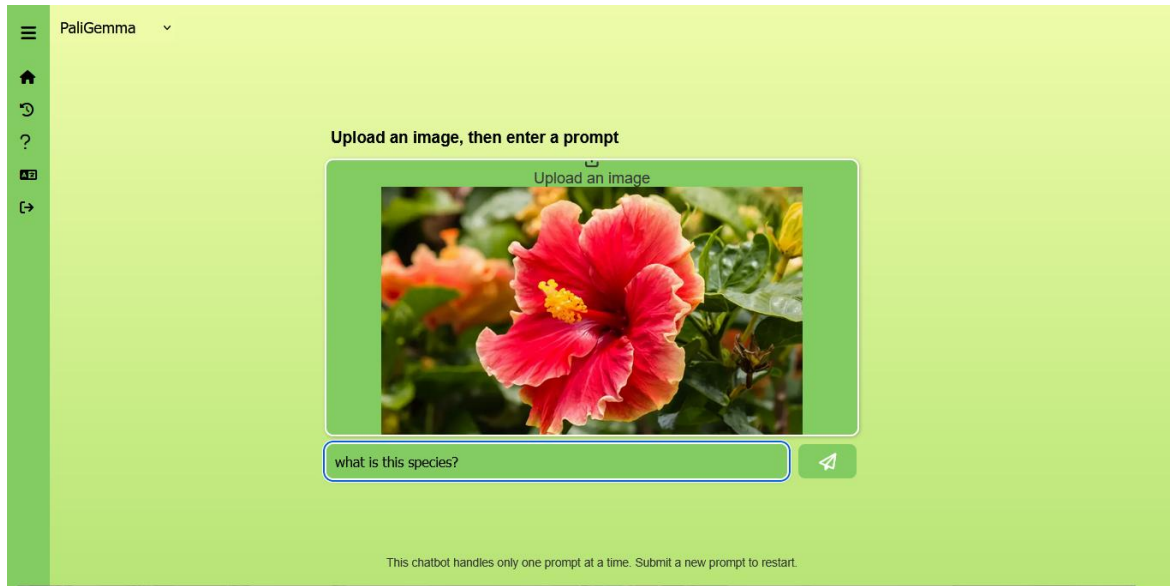


*Figure 4.47: Image preview.*

Figure 4.47 shows the home page when a user uploads an image and typed in a prompt, similar to Figure 4.38. The distinction between the two figures are the sidebar and the absence of a “Log In” button in Figure 4.47.

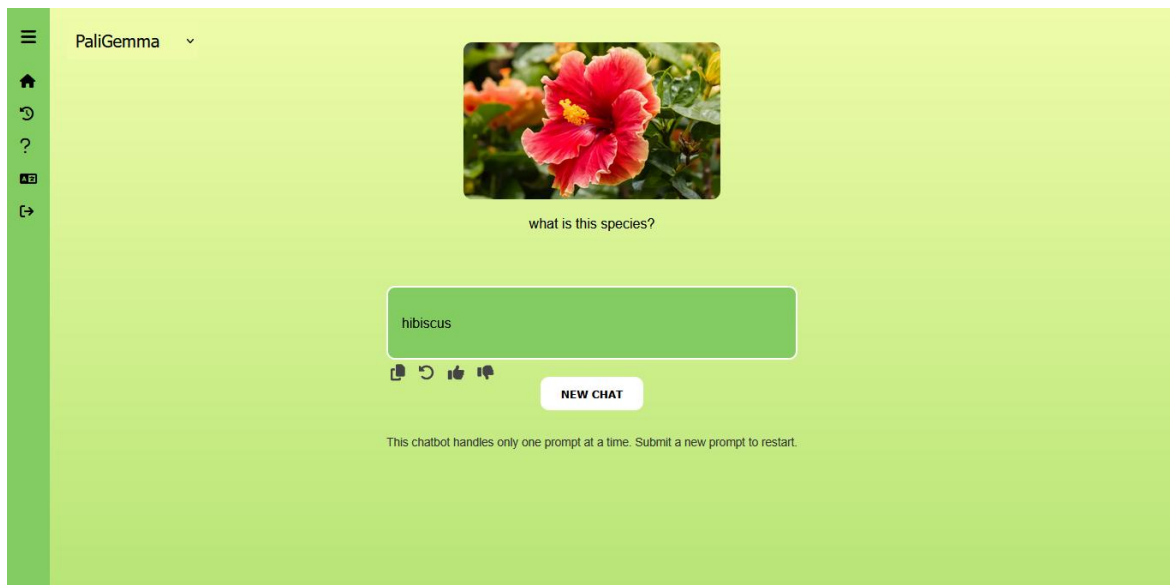


*Figure 4.48: Response page.*

Figure 4.48 shows the response page where the prompts are submitted and the AI model generates a real-time response. In this page, users can copy the response, give a thumbs up or down based on their expectation of the model’s response. The feedback will be recorded in the

“Chat History” page. User can also click on the “New Chat” button to start a new conversation with the chatbot. However, it is important to note that the “New Chat” button in Figure 4.40 has a different function than the button in Figure 4.48. The current “New Chat” button in Figure 4.48 will lead users to the home page when clicked because their prompt and response history has already been recorded into the database.

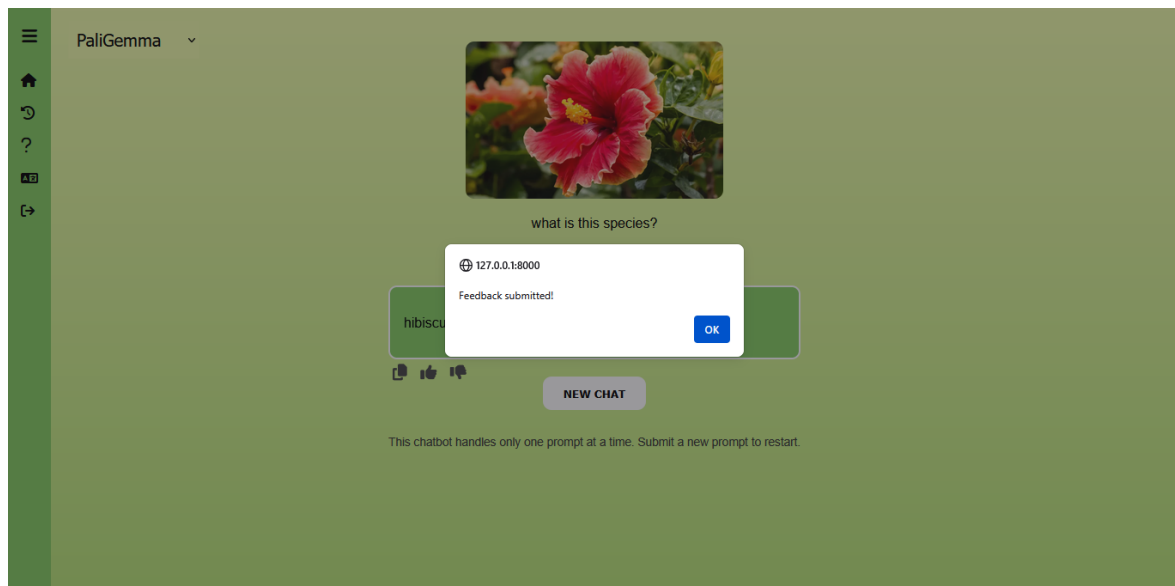


*Figure 4.49: Feedback validation.*

When a user click on the thumbs up or down icon in Figure 4.48, a pop-up overlay of a validation success message will appear as in Figure 4.49 and inform user that the feedback has been submitted and received by the system. The feedback function is crucial for this project because good feedback for a prompt will be used to improve the model by training it again using the given feedback.

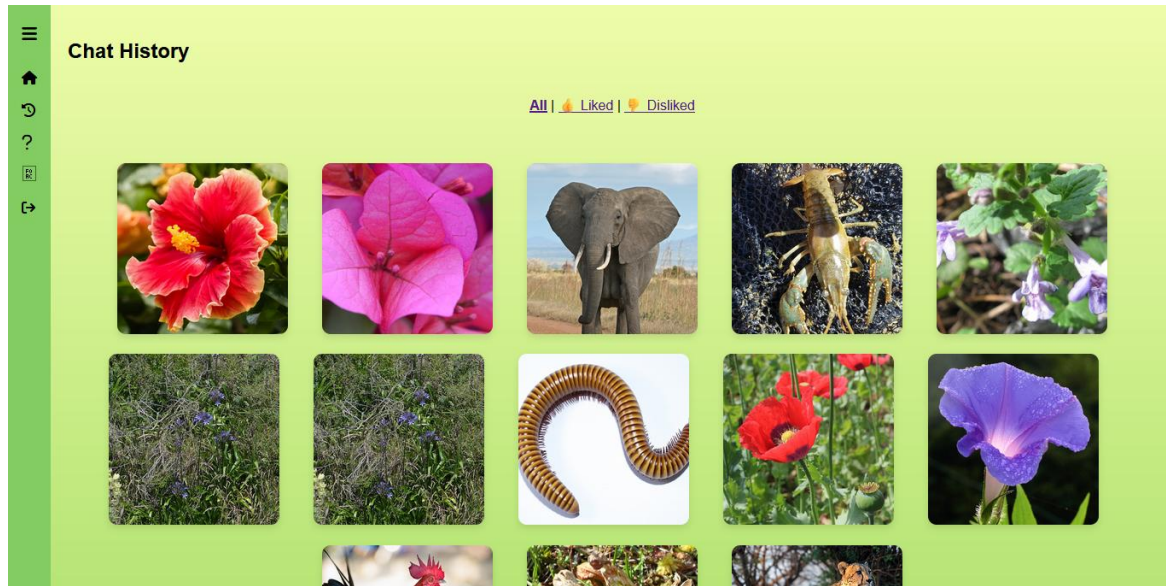


*Figure 4.50: Chat history page.*

Figure 4.50 shows the chat history page where it displays all of the user's prompts history in image format. This page also allows users to filter between the prompt and response that they ranked as liked or disliked using the thumbs up or down icon in Figure 4.48. In addition, a user can also click on any images shown in their chat history page and view their prompt and response history as shown in Figure 4.53.

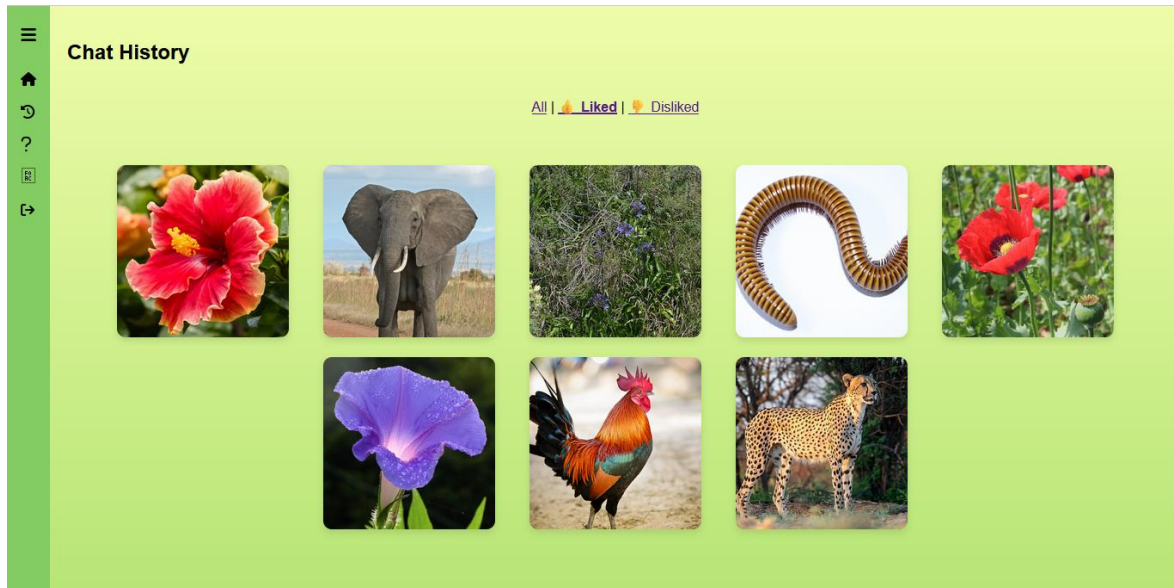


*Figure 4.51: Thumbs up feedback filter.*

Figure 4.51 shows the filtered history of user's liked responses. A response that user deemed meet their expectations were given a thumbs up and will be able to filter and look through the response history that they liked. While also able to do so with their disliked responses which is shown in Figure 4.52.



*Figure 4.52: Thumbs down feedback filter.*

Figure 4.52 shows the filtered history of user's disliked responses. These are the result of responses that do not meet user's expectations and were given thumbs down feedback. The

results of these responses are mostly “unanswerable” or the model gives a response that is inaccurate.

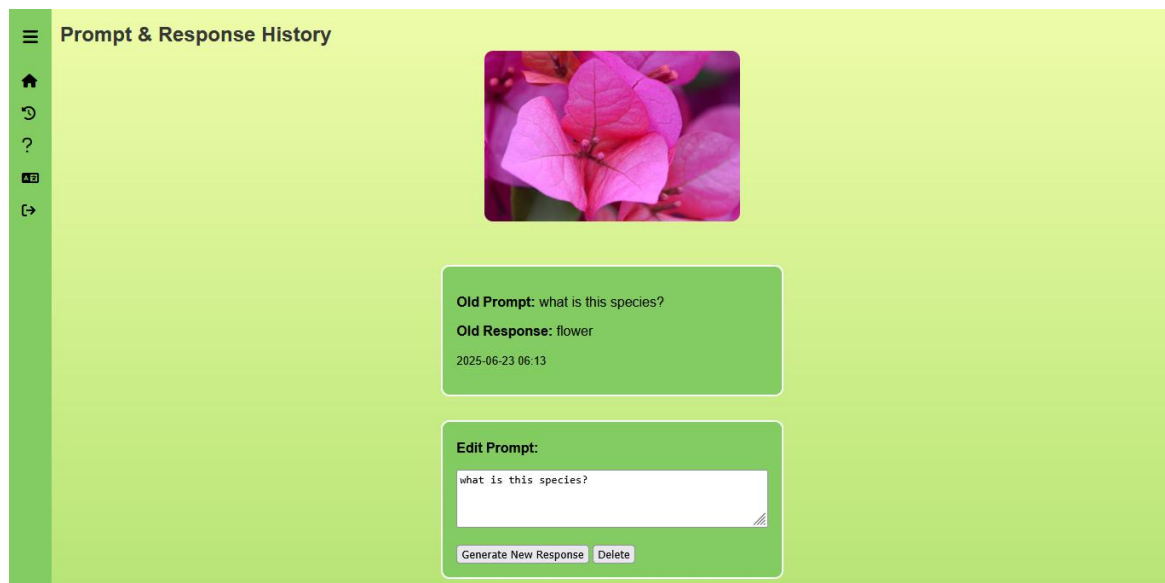


*Figure 4.53: Viewing prompt and response history.*

Figure 4.53 shows the prompt and response history of an image clicked in Figure 4.52. In this page, user can view the image’s previous prompt and response history along with the datetime stamp of the prompt submission encased in a container right below the image shown. The container below it, which is the “Edit Prompt” container allows user to edit their previous prompt and regenerate a new response to check if the model is still providing the same inaccurate response or it generates a better response than the previous one. User will also be able to delete a prompt and response history.

## 4.7 Chapter Summary

Overall, this chapter provides a comprehensive overview of the development process, beginning with the proper setup of the development environment, including the necessary software installations, configurations, and dependencies required for seamless project execution. It further elaborates on the steps involved in dataset preparation, detailing the processes for data collection, pre-processing, and annotation to ensure compatibility with the system's requirements. Finally, the chapter presents an in-depth discussion on the design and development of the web application, covering its architecture, user interface design, and

integration with the PaliGemma model, highlighting the key implementation strategies and technologies employed.

## CHAPTER 5: TESTING

### 5.1 Introduction

Testing is one of this project's methodology phases and is crucial to implement for system evaluation and ensuring the developed web application works as intended. This chapter will explicitly discuss the developed web application's testing by running both functional testing and non-functional testing. By doing this, the web application's performance can be evaluated and further improved to suit user's preferences and needs. Furthermore, the functional testing of this project will be done by the developer party and non-functional testing will be done by the developer with a user through an interview process and an evaluation form.

### 5.2 Functional Testing

In this project, functional testing will be conducted to ensure that the web application performs according to its specified requirements. The primary focus is to validate all key features function correctly such as image upload, prompt handling and response display from the PaliGemma model. Additionally, other features such as user module and chat history module will be tested accordingly as well. A unit testing will be carried out to test each of these modules and features by designing a test case based on the system's functional specifications to verify correct behaviour under normal and edge-case scenarios. This phase is crucial in helping to identify and resolve functional issues early to ensure a smooth user experience.

#### 5.2.1 Unit Testing

The purpose of unit testing is to carry out the isolation and testing of individual components of the web application such as the image upload module, form handling and API integration with HuggingFace's fine-tuned PaliGemma model. These unit tests will help detect and fix bugs at the component level before integrating them into the main application workflow.

##### 5.2.1.1 Unit Testing on User Authentication module

*Table 5.1: User registration page.*

|                         |            |                     |             |
|-------------------------|------------|---------------------|-------------|
| <b>Test Scenario ID</b> | Register-1 | <b>Test Case ID</b> | Register-1A |
|-------------------------|------------|---------------------|-------------|

| Test Case Description |                                                       | Register a user account      |                                                 | Test Priority       |                     | High               |
|-----------------------|-------------------------------------------------------|------------------------------|-------------------------------------------------|---------------------|---------------------|--------------------|
| Pre-requisite         |                                                       | N/A                          |                                                 | Post-requisite      |                     | N/A                |
| No.                   | Test Case                                             | Test Steps                   | Expected Result                                 | Post-Condition      | Actual Result       | Status (Pass/Fail) |
| 1                     | Enter valid username, valid e-mail and valid password | Enter first name             | A message “Account created. Login now” is shown | Login page is shown | Login page is shown | Pass               |
|                       |                                                       | Enter last name              |                                                 |                     |                     |                    |
|                       |                                                       | Enter valid username         |                                                 |                     |                     |                    |
|                       |                                                       | Enter valid email            |                                                 |                     |                     |                    |
|                       |                                                       | Enter valid password         |                                                 |                     |                     |                    |
|                       |                                                       | Enter valid confirm password |                                                 |                     |                     |                    |

Table 5.2: User login page.

| Test Scenario ID      |             | Login-1              |                | Test Case ID     |                | Login-1A      |                     |
|-----------------------|-------------|----------------------|----------------|------------------|----------------|---------------|---------------------|
| Test Case Description |             | Login                |                | Test Priority    |                | High          |                     |
| Pre-requisite         |             | A valid user account |                | Post-requisite   |                | N/A           |                     |
| Test Scenario         |             | Verify user login    |                |                  |                |               |                     |
| No .                  | Test Case   | Test Steps           | Test Data      | Expected Result  | Post-Condition | Actual Result | Status (Pass/Fail ) |
| 1                     | Enter valid | Enter username       | Valid username | Successful login |                | Home page     | Pass                |

|   |                                             |                      |                  |                                               |                                                                   |                                                    |      |
|---|---------------------------------------------|----------------------|------------------|-----------------------------------------------|-------------------------------------------------------------------|----------------------------------------------------|------|
|   | username and valid password                 | Enter password       | Valid password   |                                               | Home page is shown                                                | is shown                                           |      |
|   |                                             | Click “Login” button |                  |                                               |                                                                   |                                                    |      |
| 2 | Enter valid username and invalid password   | Enter username       | Valid username   | A message “Invalid user credentials” is shown | A validation error message “Invalid login credentials.” is shown. | Error message is shown and remained in login page. | Pass |
|   |                                             | Enter password       | Invalid password |                                               |                                                                   |                                                    |      |
|   |                                             | Click “Login” button |                  |                                               |                                                                   |                                                    |      |
| 3 | Enter invalid username and valid password   | Enter username       | Invalid username | A message “Invalid user credentials” is shown | A validation error message “Invalid login credentials.” is shown. | Error message is shown and remained in login page. | Pass |
|   |                                             | Enter password       | Valid password   |                                               |                                                                   |                                                    |      |
|   |                                             | Click “Login” button |                  |                                               |                                                                   |                                                    |      |
| 4 | Enter invalid username and invalid password | Enter username       | Invalid username | A message “Invalid user credentials” is shown | A validation error message “Invalid login credentials.” is shown. | Error message is shown and remained in login page. | Pass |
|   |                                             | Enter password       | Invalid password |                                               |                                                                   |                                                    |      |
|   |                                             | Click “Login” button |                  |                                               |                                                                   |                                                    |      |

Table 5.3: Forgot password page.

|                       |                                       |                                             |                                                                                                                    |                                                                        |                                                             |                    |
|-----------------------|---------------------------------------|---------------------------------------------|--------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------|-------------------------------------------------------------|--------------------|
| Test Scenario ID      |                                       | Forgot Password-1                           |                                                                                                                    | Test Case ID                                                           |                                                             | Forgot Password-1A |
| Test Case Description |                                       | Forgot Password                             |                                                                                                                    | Test Priority                                                          |                                                             | High               |
| Pre-requisite         |                                       | A valid user account                        |                                                                                                                    | Post-requisite                                                         |                                                             | N/A                |
| Test Scenario         |                                       | Verify user can send password reset request |                                                                                                                    |                                                                        |                                                             |                    |
| No.                   | Test Case                             | Test Steps                                  | Expected Result                                                                                                    | Post-Condition                                                         | Actual Result                                               | Status (Pass/Fail) |
| 1                     | Enter valid and registered e-mail     | Enter e-mail                                | A message “A password reset link has been sent to your email. Please note that it expires in 10 minutes!” is shown | An email containing password reset link is sent to user’s e-mail       | Email is received and a reset link is provided.             | Pass               |
| 2                     | Enter invalid and unregistered e-mail | Enter e-mail                                | A message “No user with email [e-mail] found” is shown.                                                            | A validation error message “No user with email [email] found” is shown | Error message is shown and remained in reset password page. | Pass               |

Table 5.4: Password reset page.

|                       |                           |                                                |                                                        |                                                               |                                                             |                    |
|-----------------------|---------------------------|------------------------------------------------|--------------------------------------------------------|---------------------------------------------------------------|-------------------------------------------------------------|--------------------|
| Test Scenario ID      |                           | Password Reset-1                               |                                                        | Test Case ID                                                  |                                                             | Password Reset-1A  |
| Test Case Description |                           | A user resetting password for existing account |                                                        | Test Priority                                                 |                                                             | High               |
| Pre-requisite         |                           | A valid user account                           |                                                        | Post-requisite                                                |                                                             | N/A                |
| Test Scenario         |                           | Verify user can reset password                 |                                                        |                                                               |                                                             |                    |
| No.                   | Test Case                 | Test Steps                                     | Expected Result                                        | Post-Condition                                                | Actual Result                                               | Status (Pass/Fail) |
| 1                     | Enter new password        | Enter password                                 | A message “Password reset. Proceed to login.” is shown | Login page is shown                                           | Redirected to login page                                    | Pass               |
|                       |                           | Enter confirm password                         |                                                        |                                                               |                                                             |                    |
|                       |                           | Click “Reset” button                           |                                                        |                                                               |                                                             |                    |
| 2                     | Enter mismatched password | Enter password                                 | A message “Passwords do not match” is shown            | A validation error message “Passwords do not match” is shown. | Error message is shown and remained in reset password page. | Pass               |
|                       |                           | Enter confirm password                         |                                                        |                                                               |                                                             |                    |
|                       |                           | Click “Reset” button                           |                                                        |                                                               |                                                             |                    |
| 3                     | Enter short password      | Enter password                                 | A message “Password must be at least 5 characters      | A validation error messafe “Password                          | Error message is shown and remained in reset                | Pass               |
|                       |                           | Enter confirm password                         |                                                        |                                                               |                                                             |                    |

|   |                                    |                                 |                                                                                                           |                                                          |                                                                                  |      |
|---|------------------------------------|---------------------------------|-----------------------------------------------------------------------------------------------------------|----------------------------------------------------------|----------------------------------------------------------------------------------|------|
|   |                                    | Click<br>“Reset”<br>button      | long” is<br>shown                                                                                         | must be at<br>least 5<br>characters<br>long” is<br>shown | password<br>page.                                                                |      |
| 4 | Reset password<br>after 10 minutes | Enter<br>password               | A<br>validation<br>error<br>message<br>“Reset link<br>has<br>expired.<br>Invalid<br>reset ID” is<br>shown | Forgot<br>Password<br>page is<br>shown                   | Error<br>message is<br>shown and<br>redirected<br>to forgot<br>password<br>page. | Pass |
|   |                                    | Enter<br>confirm<br>password    |                                                                                                           |                                                          |                                                                                  |      |
|   |                                    | Click<br>“Reset”<br>button      |                                                                                                           |                                                          |                                                                                  |      |
| 5 | Reset password<br>after 10 minutes | Click<br>password<br>reset link | A<br>validation<br>error<br>message<br>“Invalid<br>reset id” is<br>shown                                  | Remain in<br>reset<br>password<br>page.                  | Error<br>message is<br>shown and<br>remained<br>in reset<br>password<br>page.    | Pass |

#### 5.2.1.2 Unit Testing on Core Functionality Module

*Table 5.5: Image Upload*

|                              |                                         |                       |         |
|------------------------------|-----------------------------------------|-----------------------|---------|
| <b>Test Scenario ID</b>      | IMG - 1                                 | <b>Test Case ID</b>   | IMG -1A |
| <b>Test Case Description</b> | Upload image from file explorer         | <b>Test Priority</b>  | High    |
| <b>Pre-requisite</b>         | A valid user account                    | <b>Post-requisite</b> | N/A     |
| <b>Test Scenario</b>         | User can upload image and submit prompt |                       |         |

| No. | Test Case                          | Test Steps                                                | Expected Result                                                    | Post-Condition                                          | Actual Result                                         | Status (Pass/Fail) |
|-----|------------------------------------|-----------------------------------------------------------|--------------------------------------------------------------------|---------------------------------------------------------|-------------------------------------------------------|--------------------|
| 1   | Upload image from file explorer    | 1. Click upload box<br>2. Choose image from file explorer | Selected image appears as a preview                                | The uploaded image is previewed in upload box container | Image is previewed                                    | Pass               |
| 2   | Display image preview after upload | Upload a valid image                                      | Image is displayed in preview area                                 |                                                         | Image is previewed                                    | Pass               |
| 3   | Show success message after upload  | Upload valid image                                        | Success message "Image upload successful" is shown                 | Image is previewed and success message shown.           | Image is previewed and success message shown.         | Pass               |
| 4   | Upload invalid image file          | Upload an unsupported file format                         | Error message "Failed to upload image. Please try again" is shown. | Error message is shown and image upload is cancelled.   | Error message is shown and image upload is cancelled. | Pass               |
| 5   | Upload empty or no image           | Submit form without selecting an image                    | Page reloads and clears input fields                               | Remained in same page.                                  | Page reloaded and remained                            | Pass               |

|  |  |  |  |  |                  |  |
|--|--|--|--|--|------------------|--|
|  |  |  |  |  | in the same page |  |
|--|--|--|--|--|------------------|--|

Table 5.6: Response Generation

| Test Scenario ID      |                            | Response - 1                                                           |                             | Test Case ID                                          |                                                       | Response - 1A      |
|-----------------------|----------------------------|------------------------------------------------------------------------|-----------------------------|-------------------------------------------------------|-------------------------------------------------------|--------------------|
| Test Case Description |                            | Web app produce a response from AI model                               |                             | Test Priority                                         |                                                       | High               |
| Pre-requisite         |                            | A valid user account                                                   |                             | Post-requisite                                        |                                                       | N/A                |
| Test Scenario         |                            | Web app can produce a response from AI model                           |                             |                                                       |                                                       |                    |
| No.                   | Test Case                  | Test Steps                                                             | Expected Result             | Post-Condition                                        | Actual Result                                         | Status (Pass/Fail) |
| 1                     | Type a prompt              | 1. Enter a question (e.g: What animal is this?)                        | Text appears in input field | Text remains in input field                           | Text appears and remains in input field               | Pass               |
| 2                     | Submit prompt and image    | 1. Upload a valid image<br>2. Type prompt<br>3. Click “Submit” button. | Redirects to response page  | Response is generated and displayed in response page. | Redirected to response page and response is generated | Pass               |
| 3                     | Generate accurate response | Submit image and prompt                                                | Redirects to response page  | System returns accurate or                            | System returns accurate or                            | Pass               |

|   |                         |                                                                                      |                                                                                         |                                                                                 |                                                                                    |      |
|---|-------------------------|--------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------|---------------------------------------------------------------------------------|------------------------------------------------------------------------------------|------|
|   |                         |                                                                                      |                                                                                         | inaccurate<br>description                                                       | inaccurate<br>description                                                          |      |
| 4 | Submit blank<br>prompt  | 1. Upload<br>image<br>2. Leave<br>prompt<br>blank<br>3. Click<br>“Submit”<br>button. | System<br>shows<br>validation<br>error<br>message<br>“Please<br>fill out<br>this field” | Prompt<br>will not<br>submit and<br>validation<br>error<br>message is<br>shown. | Prompt is<br>not<br>submitted<br>and<br>validation<br>error<br>message is<br>shown | Pass |
| 5 | Start new<br>chat       | Click “New<br>Chat” button                                                           | Redirects<br>to home<br>page                                                            | Home<br>page is<br>displayed                                                    | Redirected<br>to home<br>page                                                      | Pass |
| 6 | Redirect to<br>homepage | Click on home<br>icon in sidebar                                                     | Redirect<br>to home<br>page                                                             | Home<br>page is<br>displayed                                                    | Redirected<br>to home<br>page.                                                     | Pass |

*Table 5.7: Chat History*

| Test Scenario ID      |                   | History - 1                |                                         | Test Case ID                                 |                             | History - 1A       |
|-----------------------|-------------------|----------------------------|-----------------------------------------|----------------------------------------------|-----------------------------|--------------------|
| Test Case Description |                   | Viewing chat history       |                                         | Test Priority                                |                             | High               |
| Pre-requisite         |                   | A valid user account       |                                         | Post-requisite                               |                             | N/A                |
| Test Scenario         |                   | User can view chat history |                                         |                                              |                             |                    |
| No.                   | Test Case         | Test Steps                 | Expected Result                         | Post-Condition                               | Actual Result               | Status (Pass/Fail) |
| 1                     | Open sidebar menu | 1. Click on menu icon      | Sidebar expands with navigation options | Sidebar expands and shows navigation options | Sidebar expanded and showed | Pass               |

|   |                    |                                                    |                                                      |                               |                                                        |      |
|---|--------------------|----------------------------------------------------|------------------------------------------------------|-------------------------------|--------------------------------------------------------|------|
|   |                    |                                                    |                                                      |                               | navigation options                                     |      |
| 2 | View chat history  | 1. Open sidebar menu<br>2. Click on “Chat History” | List of previous image and response displayed        | Redirect to chat history page | Redirected to chat history page                        | Pass |
| 3 | History accuracy   | Browse history                                     | Previous results are correctly matched with the user | Results are accurate          | Results are displayed and correctly linked to user ID. | Pass |
| 4 | Logout from system | 1. Open sidebar menu.<br>2. Click “Logout” button  | Session ends                                         | Redirect to login page        | Redirected to login page                               | Pass |

Table 5.8: Help Menu

|                       |           |                           |                 |                |               |                    |
|-----------------------|-----------|---------------------------|-----------------|----------------|---------------|--------------------|
| Test Scenario ID      |           | Help - 1                  |                 | Test Case ID   |               | Help -1A           |
| Test Case Description |           | View “Help” page          |                 | Test Priority  |               | High               |
| Pre-requisite         |           | N/A                       |                 | Post-requisite |               | N/A                |
| Test Scenario         |           | User can view “Help” page |                 |                |               |                    |
| No.                   | Test Case | Test Steps                | Expected Result | Post-Condition | Actual Result | Status (Pass/Fail) |

|   |                |                                              |                       |                         |                        |      |
|---|----------------|----------------------------------------------|-----------------------|-------------------------|------------------------|------|
| 1 | View Help page | 1. Open sidebar menu<br>2. Click “Help” icon | Redirect to Help page | Redirected to Help page | Redirect to Help page. | Pass |
|---|----------------|----------------------------------------------|-----------------------|-------------------------|------------------------|------|

*Table 5.9: Language Menu*

| Test Scenario ID      |                    | Lang - 1                                         |                           | Test Case ID                |                            | Lang -1A           |
|-----------------------|--------------------|--------------------------------------------------|---------------------------|-----------------------------|----------------------------|--------------------|
| Test Case Description |                    | View Language page                               |                           | Test Priority               |                            | High               |
| Pre-requisite         |                    | N/A                                              |                           | Post-requisite              |                            | N/A                |
| Test Scenario         |                    | User can view Language page                      |                           |                             |                            |                    |
| No.                   | Test Case          | Test Steps                                       | Expected Result           | Post-Condition              | Actual Result              | Status (Pass/Fail) |
| 1                     | View Language page | 1. Open sidebar menu<br>2. Click “Language” icon | Redirect to Language page | Redirected to Language page | Redirect to language page. | Pass               |

### 5.2.2 Integration Testing

The integration testing for the interaction between the Django web application and the PaliGemma AI model ensures that the web application communicates effectively with the AI model. This involves sending requests, handling responses, and verifying that the system as a whole functions as expected when these components are integrated.

#### 5.2.2.1 Integration Testing for Image Upload and AI Response

*Table 5.10: Image and Prompt Submission*

|                       |                                                             |                                                                                                      |                                                                                       |                                                                                      |                                                                                                  |                     |
|-----------------------|-------------------------------------------------------------|------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------|---------------------|
| Test Scenario ID      |                                                             | IMG - 2                                                                                              |                                                                                       | Test Case ID                                                                         |                                                                                                  | IMG -2A             |
| Test Case Description |                                                             | Verify that an image uploaded through web app is correctly passed to PaliGemma for processing        |                                                                                       | Test Priority                                                                        |                                                                                                  | High                |
| Pre-requisite         |                                                             | N/A                                                                                                  |                                                                                       | Post-requisite                                                                       |                                                                                                  | N/A                 |
| Test Scenario         |                                                             | User can view Language page                                                                          |                                                                                       |                                                                                      |                                                                                                  |                     |
| No.                   | Test Case                                                   | Test Steps                                                                                           | Expected Result                                                                       | Post-Condition                                                                       | Actual Result                                                                                    | Status (Pass/Fail ) |
| 1                     | Verify correct passing between web application and AI model | 1. Log in to web app using valid credentials<br>2. Upload valid image<br>3. Submit image with prompt | Image is correctly passed to the AI model and web app receives response from AI model | Redirected to Response page and receives response from PaliGemm a based on the image | Image successfully submitted and redirected to response page that displays response by AI model. | Pass                |

*Table 5.11: AI Model Generates Response*

|                              |                                                      |                      |                     |
|------------------------------|------------------------------------------------------|----------------------|---------------------|
| <b>Test Scenario ID</b>      | Model-Response - 1                                   | <b>Test Case ID</b>  | Model-Response - 1A |
| <b>Test Case Description</b> | Verify image and prompt submission and receive valid | <b>Test Priority</b> | High                |

|               |                                                                     |                                                                                                                                                                                                        |                                                                                            |                                                                         |                                                                                 |                    |
|---------------|---------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------|-------------------------------------------------------------------------|---------------------------------------------------------------------------------|--------------------|
|               |                                                                     | response then web app displays response                                                                                                                                                                |                                                                                            |                                                                         |                                                                                 |                    |
| Pre-requisite |                                                                     | N/A                                                                                                                                                                                                    |                                                                                            | Post-requisite                                                          |                                                                                 | N/A                |
| Test Scenario |                                                                     | Verify AI model response generation                                                                                                                                                                    |                                                                                            |                                                                         |                                                                                 |                    |
| No.           | Test Case                                                           | Test Steps                                                                                                                                                                                             | Expected Result                                                                            | Post-Condition                                                          | Actual Result                                                                   | Status (Pass/Fail) |
| 1             | Verify input submission and web app displays response from AI model | <div>1. After upload image, web app sends image and prompt to AI model</div> <div>2. Model process data and send back a response</div> <div>3. Web app receives response and displays it to user</div> | The system displays the correct response after receiving successful response from AI model | Redirected to response page and user sees the response on web interface | The system submits the prompts and loads the model’s response in response page. | Pass               |

*Table 5.12: AI Model Failure*

|                         |                   |                     |                    |
|-------------------------|-------------------|---------------------|--------------------|
| <b>Test Scenario ID</b> | Model-Failure - 1 | <b>Test Case ID</b> | Model-Failure - 1A |
|-------------------------|-------------------|---------------------|--------------------|

|                              |                                |                                                                                                                                                                                                                                                     |                                                                         |                         |                                                      |                            |
|------------------------------|--------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------|-------------------------|------------------------------------------------------|----------------------------|
| <b>Test Case Description</b> |                                | Verify how the Django web app handles situations where PaliGemma is unavailable or responds with an error                                                                                                                                           |                                                                         | <b>Test Priority</b>    |                                                      | High                       |
| <b>Pre-requisite</b>         |                                | N/A                                                                                                                                                                                                                                                 |                                                                         | <b>Post-requisite</b>   |                                                      | N/A                        |
| <b>Test Scenario</b>         |                                | Verify AI model error handling                                                                                                                                                                                                                      |                                                                         |                         |                                                      |                            |
| <b>No .</b>                  | <b>Test Case</b>               | <b>Test Steps</b>                                                                                                                                                                                                                                   | <b>Expected Result</b>                                                  | <b>Post-Condition</b>   | <b>Actual Result</b>                                 | <b>Status (Pass/Fail )</b> |
| 1                            | Verify AI model error handling | <div>1. Log in to web app and upload a valid image and prompt</div> <div>2. Simulate failure in communication with the AI model</div> <div>3. The web app should attempt to send the image and prompt to model but fail to receive a response</div> | Error message “Error 503: {"error":"503 Service Unavailable"}” is shown | Remain on the same page | Error message is shown and remained in the same page | Pass                       |

### **5.3 Non-Functional Testing**

In this section, a non-functional testing is carried out to evaluate the system's overall performance, usability, reliability and user satisfaction. This section will highlight on two questionnaire feedback forms prepared which is the User Acceptance Testing (UAT) survey's result and Model Performance Testing via a Google Form survey.

The UAT feedback highlights the user experience, ease of use and any area requiring improvement based on real-world usage. The user group of this testing are 11 students from Universiti Malaysia Sarawak (UNIMAS), University of Technology Sarawak (UTS), UniKL and other university. The survey results will help identify potential usability issues and further enhance the system's interface to better meet user needs.

The Model Performance Testing is another questionnaire prepared to evaluate the model's performance in response accuracy. There are 5 respondents that helped evaluate the model's performance based on specific instructions given via Google Form.

The feedback from Google Form will be analysed and the results will be presented in the section below to review on user satisfaction.

#### **5.3.1 Usability Testing**

##### **5.3.1.1 User Acceptance Testing**

The usability testing will contain user feedback collected through a Google Form mainly covering user's feedback. The other feedback result will be attached in Appendix B.

What did you like most about the system?

9 responses

Green colour is my favourite colour!!!

The sleek, simple design of the system

The languages of the system is very simple and understandable

Easy to use

Ui

The species identification system

it generate the same as chatgpt so no problem at all using it

Straight to the point

easy interface

*Figure 5.1: What did you like most about the system?*

As for the first question “What did you like most about the system?” in Figure 5.1, nine out of eleven respondents answered and the answers are mostly about the user interface with some mentioning that the interface is straightforward and easy to understand.

What problems did you encounter, if any?

9 responses

i couldn't submit on the first time as the page refreshed, but after that the 2nd time and after, it works!

First time using with a prompt what is this animal? lead to no generation from the chatbot

It cannot answer my request/question to identify my plant

the preview of the image is not full, it takes a bit too long to get a response

Just redirection after clicking new chat button

Nothing

none

nothing much

no

*Figure 5.2: What problems did you encounter, if any?*

In Figure 5.2, the second question is “What problems did you encounter, if any?” which received nine responses where four respondents answered “None” and the other five respondents mentions of the response generation of the AI model.

What suggestions do you have for improvement?

9 responses

Better, easier to navigate UI such as the send prompt button.

The appearance of the systems

Fine tune the model

A faster load time.

Correct redirection of page/session, add pass requirements/criteria below when setting password and resetting password

The response time of the prompt

none

perhaps, icons could be bigger and a bit more textual

accurate answer

*Figure 5.3: What suggestions do you have for improvement?*

In Figure 5.3, the third question is “What suggestions do you have for improvement?” where nine respondents answered. While some respondent encountered accurate response generation, others recommended for a faster load/response time from the model.

Based on these users’ feedback, most are satisfied with the web application’s usability and the straightforward interface. However, there are conflicting feedback where some users like the user interface of the system and some other users suggested for improvement on the user interface. Other than that, users also suggested to fasten the response generation time of the model and proper redirection of the web application’s buttons. It is important to note that the latter has been debugged and fixed after the user’s feedback.

#### **5.3.1.2 Model Performance Testing**

In this section, the PaliGemma model performance will be evaluated based on user’s feedback related to the model’s response accuracy. This testing was done in the same manner as the UAT but respondents were sent a Google Form link and instructions of the testing was

explained via Google Form. In this testing, respondents were required to access a Google Drive link and download a folder containing 10 images to be used for the testing while also answering the Google Form as they test each image. The response chart of each question will be attached in Appendix C. This section will summarize the image that receives 100 percent accuracy and images that did not receive 100 percent accuracy from respondents.

The images prepared for testing are as shown in Figure 5.4.

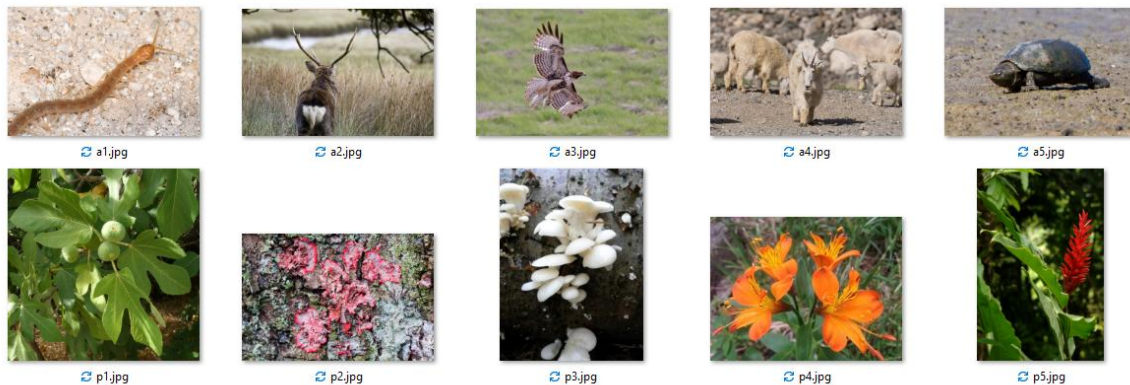


*Figure 5.4: Images for testing.*

In Figure 5.4, the images prepared for testing consists of five animal images and five plants images. The animal images are labelled as a1.jpg until a5.jpg which means animal 1 until animal 5 while the plants images are labelled as p1.jpg until p5.jpg which means plant 1 until plant 5. The questionnaire prepared through Google Form are based off the image labels and each question comprises of the answer options of ‘Yes’, ‘No’ and ‘Other’. The questionnaire and results will be summarised in Table 5.13.

The questions prepared for the respondents are shown in Table 5.13.

*Table 5.13: Model Performance Testing Results*

| No. | Question                                        | Response            |
|-----|-------------------------------------------------|---------------------|
| 1   | Did image a1.jpg gave the response ‘centipede’? | 100% answered ‘Yes’ |
| 2   | Did image a2.jpg gave the response ‘deer’?      | 100% answered ‘Yes’ |
| 3   | Did image a3.jpg gave the response ‘hawk’?      | 100% answered ‘Yes’ |
| 4   | Did image a4.jpg gave the response ‘goat’?      | 100% answered ‘Yes’ |

|    |                                                  |                                                                           |
|----|--------------------------------------------------|---------------------------------------------------------------------------|
| 5  | Did image a5.jpg gave the response ‘turtle’?     | 80% answered ‘Yes’, 20% answered ‘It recognises it as part of the ground’ |
| 6  | Did image p1.jpg gave the response ‘fig’?        | 80% answered ‘Yes’, 20% answered ‘It recognises it as a regular plant’    |
| 7  | Did image p2.jpg gave the response ‘lichen’?     | 100% answered ‘Yes’                                                       |
| 8  | Did image p3.jpg gave the response ‘mushroom’?   | 100% answered ‘Yes’                                                       |
| 9  | Did image p4.jpg gave the response ‘lily’?       | 80% answered ‘Yes’, 20% answered ‘orchid’                                 |
| 10 | Did image p5.jpg gave the response ‘red ginger’? | 100% answered ‘Yes’                                                       |

$$\text{Accuracy} = \frac{\text{TP} + (\text{PC} \times 0.5)}{\text{TOTAL}} = \frac{7 + (3 \times 0.5)}{10} = \frac{8.5}{10} = 85\%$$

*Figure 5.5: Formula to calculate PaliGemma’s accuracy.*

Based on table 5.13, the PaliGemma model correctly identified the objects in 70% of the test images with full confidence meaning a 100 percent respondent agreement. An additional 30% of the responses were partially correct where at least 80 percent of users agreed with the model’s prediction but some reported alternative interpretations.

According to Figure 5.5, TP stands for ‘True Positive’ which means the model returns the expected response while PC stands for ‘Partial Correctness’ which means response that are partially correct according to the image however the model does not return the expected response. ‘Partial Correctness’ takes the value of 0.5 due to the returned response not returning the subject of the image which is an entity that is either a wildlife or plant, but rather returning response related to the background of the subject, therefore the response is not entirely incorrect. Adjusted for partial correctness, the model demonstrates an estimated accuracy of

85% based on the formula in Figure 5.5, indicating robust performance in object recognition across the sample images.

## **CHAPTER 6: CONCLUSION AND FUTURE WORK**

### **6.1 Introduction**

This chapter will explore the project's achievement of objectives, limitations, future work and conclusion were discussed. The discussion is made according to results obtained from the implementation and testing chapter where the PaliGemma model was fine-tuned based on custom dataset and deployed to HuggingFace for project use.

### **6.2 Achievement of Objectives**

This project was aiming to achieve all objectives defined in the first chapter of this report:

- 1) To review existing systems of Large Language Model and compare their capabilities with PaliGemma as the Vision-Language Model.
- 2) To develop a web application chatbot which integrates PaliGemma for both visual and text-based input for educational purposes.
- 3) To evaluate the performance of PaliGemma in describing images related to nature and wildlife based on user prompts.

The first objective was successfully addressed in Chapter 2: Literature Review where the PaliGemma was compared to three other large language models in terms of their architecture, performance and suitability for visual-language tasks, thus providing a clear vision and solid foundation for selecting the most appropriate models for system development. Models that were compared are ChatGPT, Google Gemini and DALL-E 3. Each of these models were evaluated based on the features that are available in their respective web application system. Though all three reviewed models are Question-Answering based system, DALL-E 3 works differently compared to ChatGPT and Google Gemini where DALL-E 3 accepts text as input and generates image as output, while both ChatGPT and Google Gemini works as a text-to-text model generally. By taking inspiration from these systems' architecture, PaliGemma is chosen to develop a web application that accepts both image and text as inputs then generates text as output. The system developed works as an image classification system where it identifies images uploaded by a user and answers user's prompt based on the uploaded image.

To visualize the first objective's achievement, the second objective was accomplished by developing the web application using Python's Django framework to build the backend,

frontend and database altogether. The PaliGemma model was finetuned to custom datasets and deployed on HuggingFace Inference Endpoint. The deployed model is then integrated into the developed web application using HuggingFace API. These integrations determine the achievement of the second objective by having a complete chatbot interface to upload image and submit prompts then displays the response generated by the model. Moreover, the web application was deployed to allow for online access and users or testers can access the web application on the Internet.

The third objective was successfully achieved by conducting performance testing involving ten sample images related to nature and wildlife, which were evaluated by five human respondents. Each image was processed through the developed chatbot system, and the output generated by the PaliGemma model was compared against the expected response. Respondents were then asked to verify the correctness of each response. The results, summarized in Chapter 5, Table 5.13, showed that 70% of the images received full agreement (100% “Yes” responses) from users, confirming that the model’s interpretation was accurate. The remaining 30% received partial agreement, where 80% of respondents acknowledged the model’s answer as correct, while 20% provided alternative interpretations. These findings indicate a high level of accuracy in image understanding and prompt-response relevance. By combining both quantitative feedback and qualitative analysis from users, the system demonstrated an estimated adjusted accuracy of 85%. This validates the capability of PaliGemma as a reliable Vision-Language Model for educational applications, thereby fulfilling the third and final objective of the project.

### **6.3 Limitations**

There were limitations and constraints throughout the project’s development that affects the objectives achievement. First of all, a vision-language model like PaliGemma is a large model that requires high-performance hardware and resources to train and finetune. The main issue faced when trying to finetune PaliGemma is the lack of proper and suitable hardware. Even though the existence of free TPU usage is available in Google Colab, it will start crashing whenever trying to load the model’s configuration files.

The dataset preparation for PaliGemma was challenging due to the PaliGemma’s recently published nature and this limited the availability of existing dataset that caters to PaliGemma’s architecture. Specifically, PaliGemma requires datasets in JSONL format and the

project centers around the case study of nature and wildlife species identification. These requirements made it difficult to find a suitable pre-existing dataset which leads to the alternative of manually curating and formatting a custom dataset to meet the model's input specifications.

The model requires a substantial amount of VRAM for both training and deployment. However, the existing hardware lacks the VRAM capacity which significantly impacts the project's progress. To overcome this, cloud-based solutions are approached, utilizing resources such as Google Colab's notebook and HuggingFace Inference Endpoint to fulfil the necessary computational resources to support the model's training and deployment. The cloud-based approach will charge their services in a pay-as-you-go method and that costs money. In this project, a huge VRAM capacity is needed to complete the project, for instance, an Nvidia A100 TPU which comes with a cost of 4 USD per hour of usage. With such expensive service, testing time becomes valuable and must be used efficiently to minimize unnecessary charges.

#### **6.4 Future Work**

For future works, the project will finetune the PaliGemma model with a larger dataset consisting of more than 100,000 rows of data to produce a model with better performance and inference. To facilitate this, datasets will continue to be sourced from iNaturalist website due to its large database of plants and wildlife species. The reason for this huge collection of datasets is to enhance the model's accuracy in identifying plants and wildlife species.

Furthermore, a high-performance hardware, particularly a Graphics Processing Unit (GPU) with 16GB VRAM is needed to train the model where system crashing is prevalent if using an average hardware and must be avoided to achieve the model's training requirement. Other than using GPU, another option is available to execute the model training which is a Tensor Processing Unit (TPU). A TPU is a cloud-based solutions that is available for use in multiple platforms such as Amazon Web Services, Microsoft Azure, Kaggle, Google Colab and Google Cloud Platform. However, high-performance hardware and resources comes with a high cost for both GPU and TPU. This will enable a smooth model training and improve model's accuracy and performance.

Moreover, the web application's functionality such as user feedback can be useful in making further improvements for the PaliGemma model by extracting responses with thumbs up feedback from users and using that data to retrain the model.

## **6.5 Conclusion**

To conclude, PaliGemma is a suitable model for this project for its multimodal capabilities and aligns with the goal of nature and wildlife species identification. Its ability to process both visual and textual inputs enables a more interactive and intelligent system which will be integrated into the web application.

Moreover, the web application developed using Python's Django framework proved to be efficient and timesaving due to its built-in features such as a user authentication module that eliminates the need for custom development. This streamlined the development process and allowed for greater focus on the integration of the AI model.

## References

- Beyer, L., Steiner, A., Pinto, A. S., Kolesnikov, A., Wang, X., Salz, D., Neumann, M., Alabdulmohsin, I., Tschannen, M., Bugliarello, E., Unterthiner, T., Keysers, D., Koppula, S., Liu, F., Grycner, A., Gritsenko, A., Houlsby, N., Kumar, M., Rong, K., . . . Zhai, X. (2024). PaliGemma: A versatile 3B VLM for transfer. *arXiv (Cornell University)*. <https://doi.org/10.48550/arxiv.2407.07726>
- Goh, J. Y., Yunus, Y. M., Sheikh, U. U., & Ali, M. S. M. (2024). Vision Language Models for Oil Palm Fresh Fruit Bunch Ripeness Classification. *2024 IEEE 8th International Conference on Signal and Image Processing Applications (ICSIPA)*, Kuala Lumpur, Malaysia, 2024, pp. 1–6. <https://doi.org/10.1109/icsipa62061.2024.10686009>
- Leo Ueno, Trevor Lynn. (May 15, 2024). *PaliGemma: An Open Multimodal Model by Google*. Roboflow Blog. <https://blog.roboflow.com/paligemma-multimodal-vision/>
- Steiner, A., Pinto, A. S., Tschannen, M., Keysers, D., Wang, X., Bitton, Y., Gritsenko, A., Minderer, M., Sherbondy, A., Long, S., Qin, S., Ingle, R., Bugliarello, E., Kazemzadeh, S., Mesnard, T., Alabdulmohsin, I., Beyer, L., & Zhai, X. (2024). PaliGemma 2: A family of versatile VLMs for transfer. *arXiv (Cornell University)*. <https://doi.org/10.48550/arxiv.2412.03555>
- Carlà, M. M., Gambini, G., Baldascino, A., Boselli, F., Giannuzzi, F., Margollicci, F., & Rizzo, S. (2024). Large language models as assistance for glaucoma surgical cases: a ChatGPT vs. Google Gemini comparison. *Graefes S Archive for Clinical and Experimental Ophthalmology*, 262(9), 2945–2959. <https://doi.org/10.1007/s00417-024-06470-5>
- Rane, N., Choudhary, S., & Rane, J. (2024). Gemini versus ChatGPT: applications, performance, architecture, capabilities, and implementation. *Journal of Applied Artificial Intelligence*, 5(1), 69–93. <https://doi.org/10.48185/jaai.v5i1.1052>

- Team, G., Reid, M., Savinov, N., Teplyashin, D., Dmitry, Lepikhin, Lillicrap, T., Alayrac, J., Soricut, R., Lazaridou, A., Firat, O., Schrittwieser, J., Antonoglou, I., Anil, R., Borgeaud, S., Dai, A., Millican, K., Dyer, E., Glaese, M., . . . Vinyals, O. (2024). Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv (Cornell University)*. <https://doi.org/10.48550/arxiv.2403.05530>
- Wu, T., He, S., Liu, J., Sun, S., Liu, K., Han, Q., & Tang, Y. (2023). A brief overview of ChatGPT: the history, status quo and potential future development. *IEEE/CAA Journal of Automatica Sinica*, 10(5), 1122–1136. <https://doi.org/10.1109/jas.2023.123618>
- Kounte, M. R., Tripathy, P. K., P, P., & Bajpai, H. (2020). Analysis of Intelligent Machines using Deep learning and Natural Language Processing. *2020 4th International Conference on Trends in Electronics and Informatics (ICOEI)*(48184), 956–960. <https://doi.org/10.1109/icoei48184.2020.9142886>
- Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., ... Lowe, R. (2022). Training language models to follow instructions with human feedback. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, & A. Oh (Eds.), *Advances in Neural Information Processing Systems* (Vol. 35, pp. 27730–27744). Retrieved from [https://proceedings.neurips.cc/paper\\_files/paper/2022/file/b1efde53be364a73914f58805a001731-Paper-Conference.pdf](https://proceedings.neurips.cc/paper_files/paper/2022/file/b1efde53be364a73914f58805a001731-Paper-Conference.pdf)
- OpenAI. (2023b). DALL E 3 System Card. [https://cdn.openai.com/papers/DALL\\_E\\_3\\_System\\_Card.pdf](https://cdn.openai.com/papers/DALL_E_3_System_Card.pdf)
- Betker, J., Goh, G., Jing, L., Brooks, T., Wang, J., Li, L., Ouyang, L., Zhuang, J., Lee, J., Guo, Y., Manassra, W., Dhariwal, P., Chu, C., Jiao, Y., & Ramesh, A. (n.d.). Improving Image Generation with Better Captions. <https://cdn.openai.com/papers/dall-e-3.pdf>

Ramesh, A., Pavlov, M., Goh, G., Gray, S., Voss, C., Radford, A., Chen, M., & Sutskever, I. (2021). Zero-Shot Text-to-Image Generation. *arXiv (Cornell University)*.  
<https://doi.org/10.48550/arxiv.2102.12092>

Ramesh, A., Dhariwal, P., Nichol, A., Chu, C., & Chen, M. (n.d.). Hierarchical Text-Conditional Image Generation with CLIP Latents. <https://cdn.openai.com/papers/dalle-2.pdf>

## Appendices

### Appendix A

#### Survey Questions

##### Section 1: Demographics

Age \*

- ☐ 10 -14
- ☐ 15 -19
- ☐ 20 - 24
- ☐ 25 - 29
- ☐ 30 and above

*Figure 7.1: Question 1 of Section 1.*

Gender \*

- ☐ Male
- ☐ Female

*Figure 7.2: Question 2 of Section 1.*

Current/Pursuing education level \*

- ☐ Primary school level
- ☐ Secondary school level
- ☐ Matriculation/Form 6/Foundation/Diploma
- ☐ Bachelor's degree
- ☐ Master's degree

*Figure 7.3: Question 3 of Section 1.*

Faculty (UNIMAS) \*

- ☐ Faculty of Economics and Business
- ☐ Faculty of Engineering
- ☐ Faculty of Applied and Creative Arts
- ☐ Faculty of Cognitive Sciences and Human Development
- ☐ Faculty of Social Sciences and Humanities
- ☐ Faculty of Resource Science and Technology
- ☐ Faculty of Computer Science and Information Technology
- ☐ Faculty of Built Environment
- ☐ Non-UNIMAS

*Figure 7.4: Question 4 of Section 1.*

## Section 2: Familiarity with AI Chatbot

Are you familiar with AI chatbot such as ChatGPT, Google Gemini and other similar system? \*

- ☐ Yes
- ☐ No

*Figure 7.5: Question 1 of Section 2.*

Have you used the services of AI chatbot? \*

- ☐ Yes
- ☐ No

*Figure 7.6: Question 2 of Section 2.*

How often do you use AI chatbot? \*

|        | 1                     | 2                     | 3                     | 4                     | 5                     |            |
|--------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|------------|
| Seldom | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | Very often |

Figure 7.7: Question 3 of Section 2.

What do you usually use the AI chatbot for? \*

- ☐ Learning and Education
- ☐ Programming and Development
- ☐ Daily life planning
- ☐ Tech support (Troubleshooting)
- ☐ Entertainment and Fun

Figure 7.8: Question 4 of Section 2.

Would you consider using AI chat for the species identification of nature and wildlife? \*

- ☐ Yes
- ☐ Maybe
- ☐ No

Figure 7.9: Question 5 of Section 2.

### Section 3: Web Application Chatbot Feature

Are you familiar with the AI model used within these chatbots? (E.g. GPT4o-mini, Gemma) \*

- ☐ Yes
- ☐ No

Figure 7.10: Question 1 of Section 3.

Do you know of PaliGemma as a vision-language model? \*

☐ Yes

☐ No

*Figure 7.11: Question 2 of Section 3.*

Do you know of any system that is able to describe image after prompting? (image captioning) \*

☐ Yes

☐ No

*Figure 7.12: Question 3 of Section 3.*

If your answer for the question above is Yes, state the name of the app/website \*

Short answer text

*Figure 7.13: Question 4 of Section 3.*

Would you be interested to know how PaliGemma works to describe an image? \*

☐ Yes

☐ No

*Figure 7.14: Question 5 of Section 3.*

If there is a chance to build such system, what features would you expect/prefer? \*

- ☐ Textbox prompt
- ☐ Image upload
- ☐ Login
- ☐ User rating (for each response)
- ☐ Chat sharing
- ☐ Video upload

*Figure 7.15: Question 6 of Section 3.*

What other features would you suggest for this web application chatbot?

Long answer text

---

*Figure 7.16: Question 7 of Section 3.*

## Appendix B

### User Authentication Module

Were you able to sign up/register as a user successfully?

11 responses

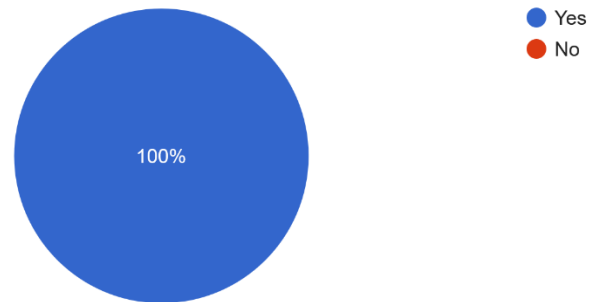


*Figure 7.17: Were you able to sign up/register as a user successfully?*

Were you able to log in after signing up?

11 responses

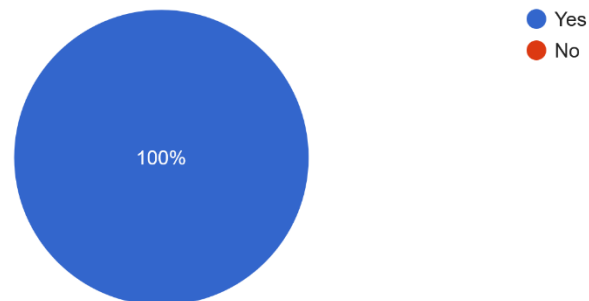


*Figure 7.18: Were you able to log in after signing up?*

Did you receive an email after trying to reset password?

11 responses

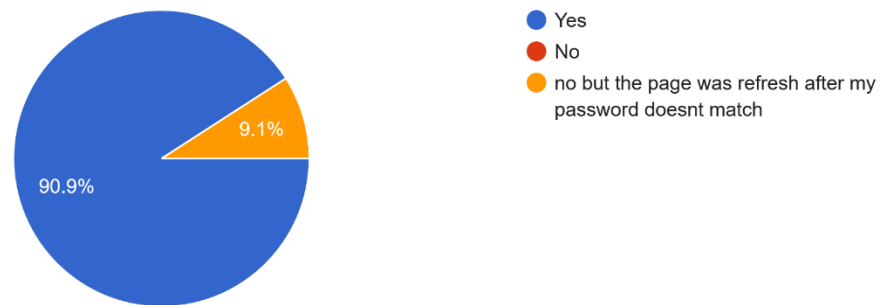


*Figure 7.19: Did you receive an email after trying to reset password?*

Were you able to reset your password successfully?

11 responses

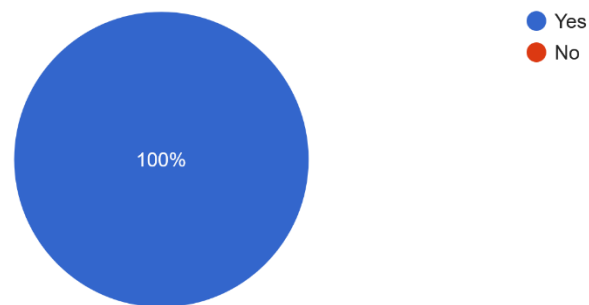


*Figure 7.20: Were you able to reset your password successfully?*

Was there an error message when resetting password format was not followed?

11 responses

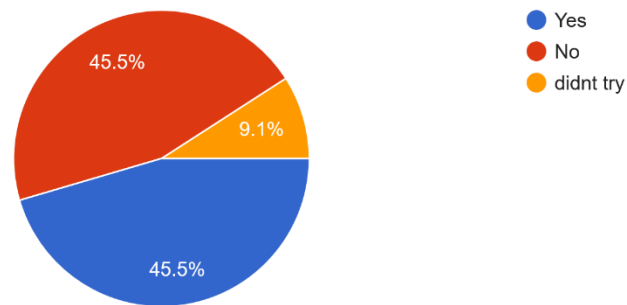


*Figure 7.21: Was there an error message when resetting password format was not followed?*

Were you able to logout?

11 responses

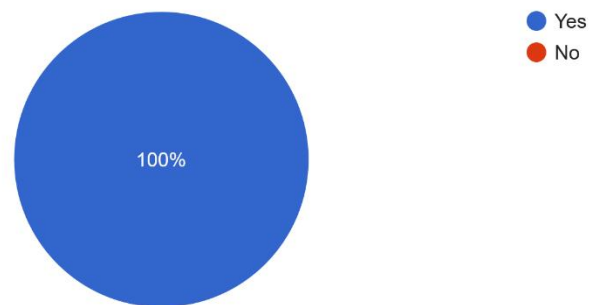


*Figure 7.22: Were you able to logout?*

## Web Application Main Functionality

Were you able to upload an image?

11 responses

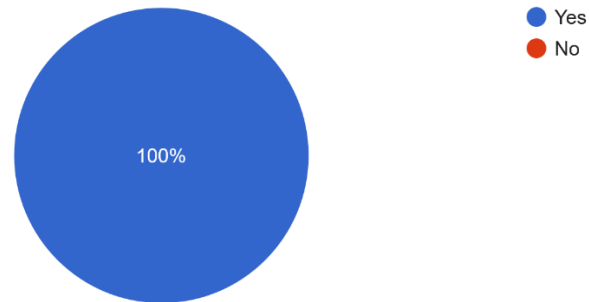


*Figure 7.23: Were you able to upload an image?*

Was the image previewed after uploading?

11 responses

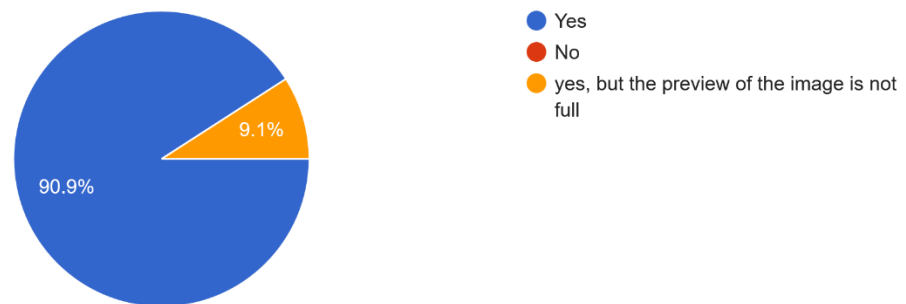


*Figure 7.24: Was the image previewed after uploading?*

Was a success message displayed after you uploaded an image?

10 responses

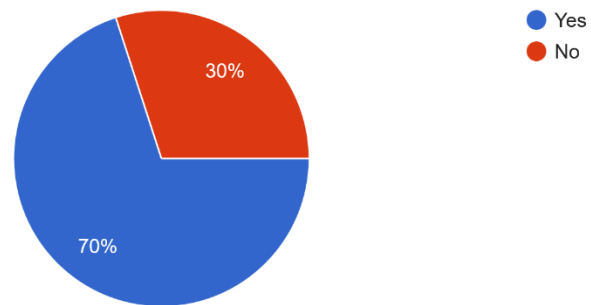


*Figure 7.25: Was a success message displayed after you uploaded an image?*

Did the system allow you to upload an image without issues?

11 responses

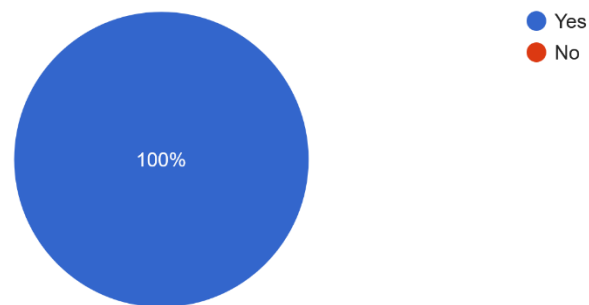


*Figure 7.26: Did the system allow you to upload an image without issues?*

Were you able to type a question?

11 responses

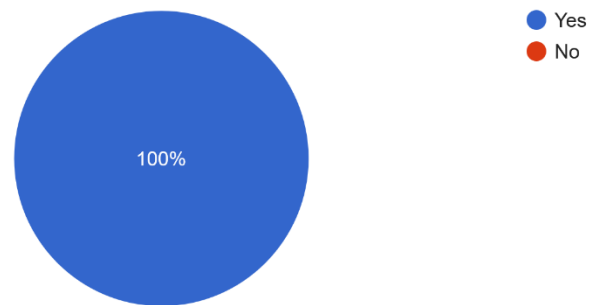


*Figure 7.27: Were you able to type a question?*

Were you able to click on the submit button?

11 responses

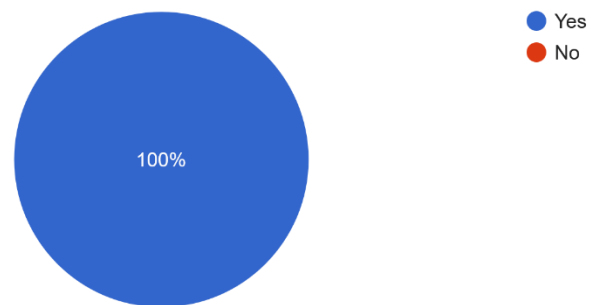


*Figure 7.28: Were you able to click on the submit button?*

Did the web application redirect you to a response page after clicking submit?

11 responses

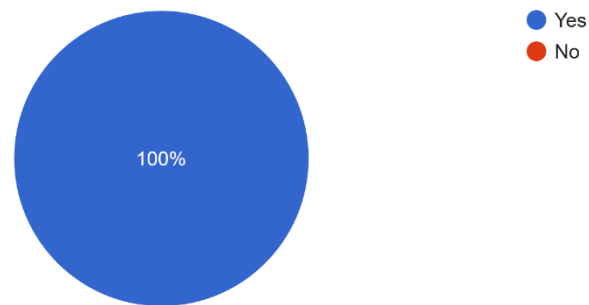


*Figure 7.29: Did the web application redirect you to a response page after clicking submit?*

Did the generated result match what you expected?

11 responses

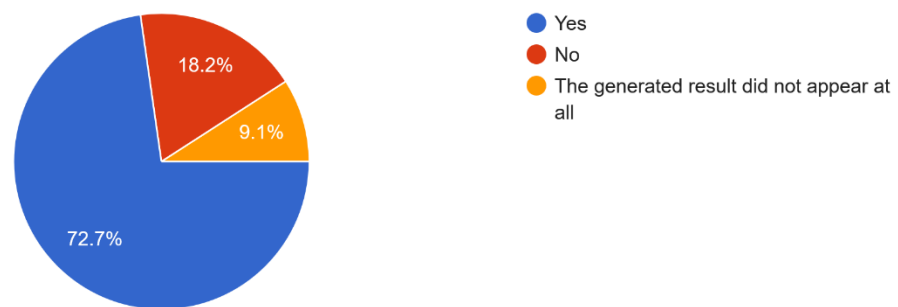


*Figure 7.30: Did the generated result match what you expected?*

Were you able to start a new chat?

11 responses

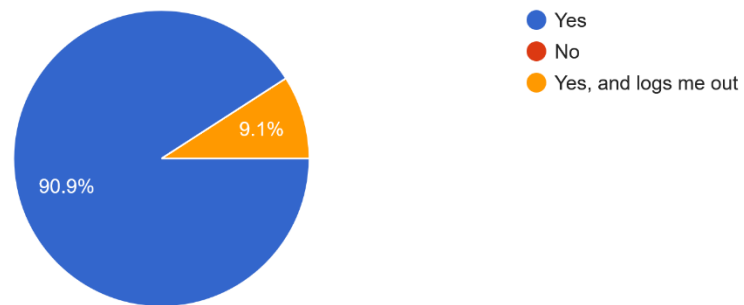


*Figure 7.31: Were you able to start a new chat?*

Were you redirected to the homepage?

11 responses

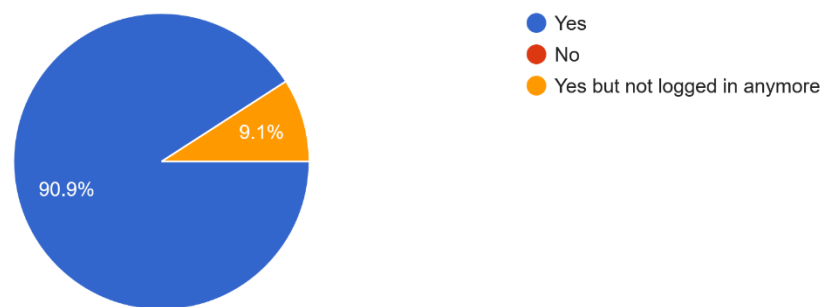


*Figure 7.32: Were you redirected to the homepage?*

Were you able to click and open the sidebar menu?

11 responses

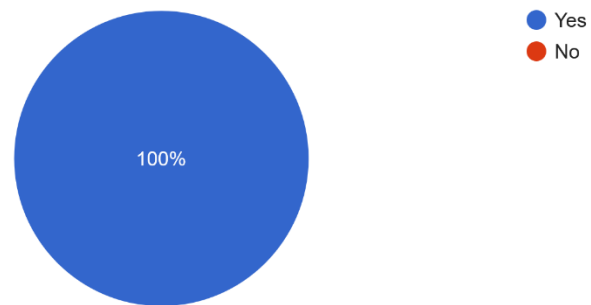


*Figure 7.33: Were you able to click and open the sidebar menu?*

Were you able to view your account's chat history?

11 responses

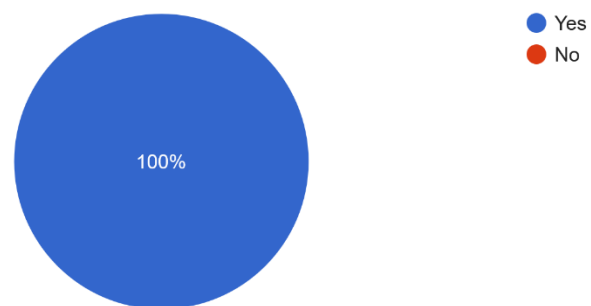


*Figure 7.34: Were you able to view your account's chat history?*

Was your account's chat history correct?

11 responses

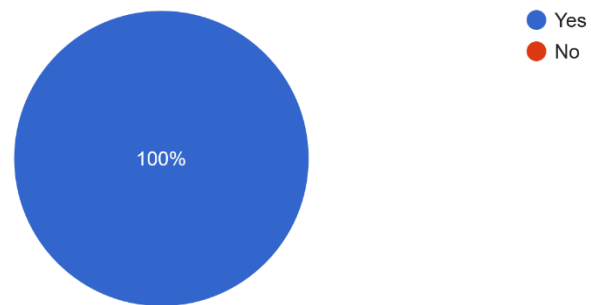


*Figure 7.35: Was your account's chat history correct?*

## **Usability**

Was the interface easy to use?

11 responses

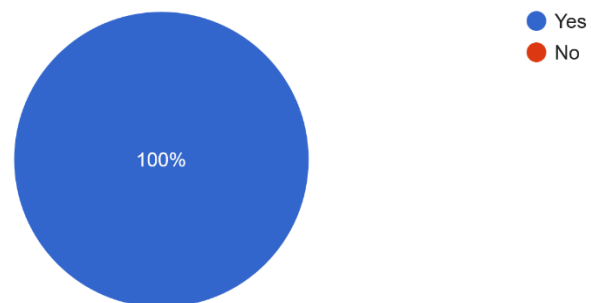


*Figure 7.36: Was the interface easy to use?*

Did you find it easy to understand what to do on each page?

11 responses

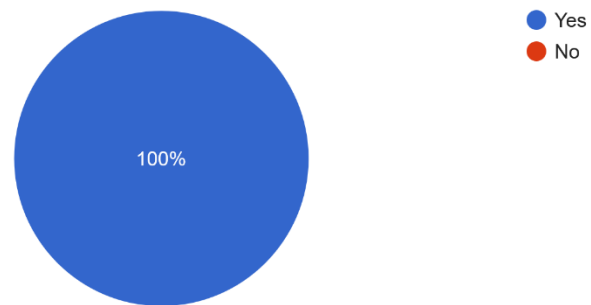


*Figure 7.37: Did you find it easy to understand what to do on each page?*

Was it easy to navigate the web application?

11 responses

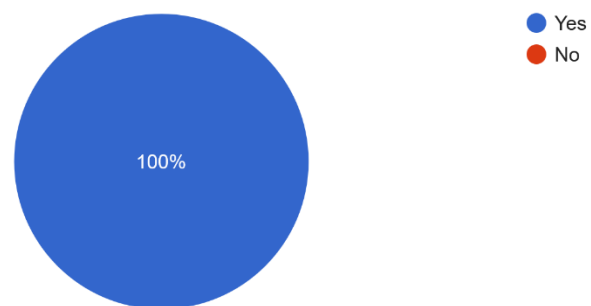


*Figure 7.38: Was it easy to navigate the web application?*

Did the layout and design feel clear and intuitive?

11 responses

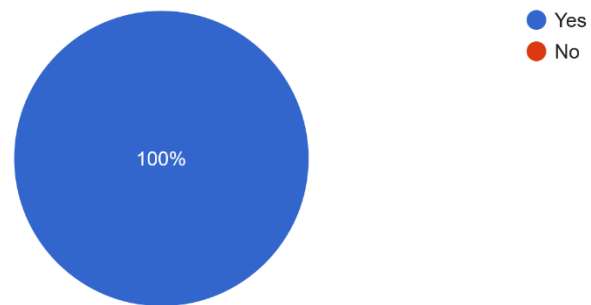


*Figure 7.39: Did the layout and design feel clear and intuitive?*

Were instructions easy to understand?

11 responses

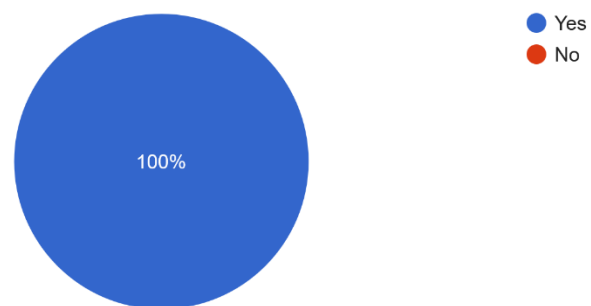


*Figure 7.40: Were instructions easy to understand?*

## Performance

Was the response time acceptable?

11 responses

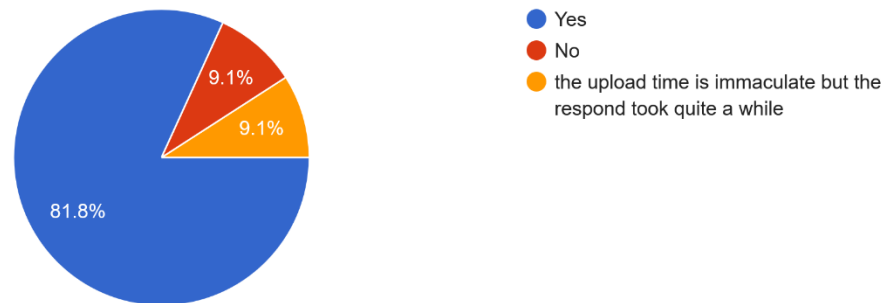


*Figure 7.41: Was the response time acceptable?*

Did you experience any crashes, freezes or bugs?

11 responses

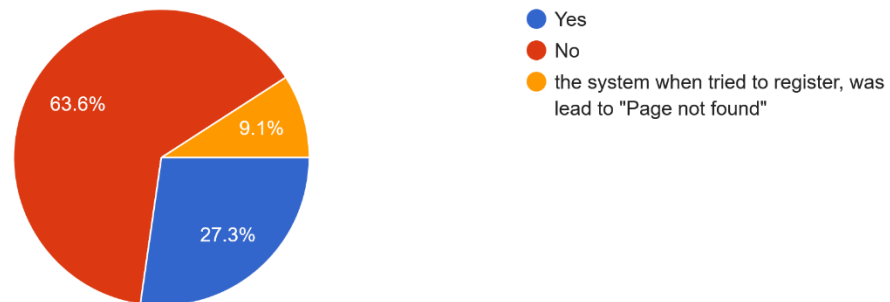


*Figure 7.42: Did you experience any crashes, freezes or bugs?*

## **Error Handling**

If you provided invalid input, did the system give you a helpful error message?

11 responses

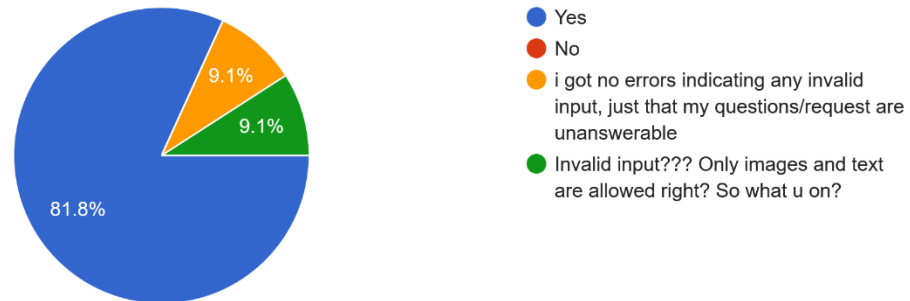


*Figure 7.43: If you provided invalid input, did the system give you a helpful error message?*

If you submitted a blank prompt, did the system handle it gracefully?

11 responses

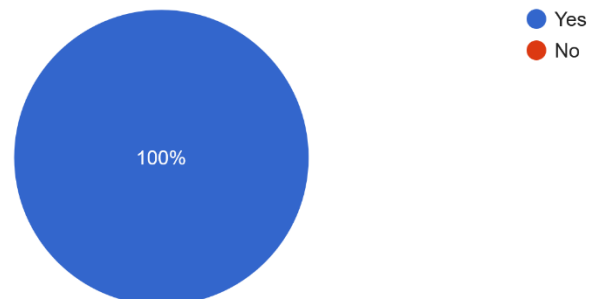


*Figure 7.44: If you submitted a blank prompt, did the system handle it gracefully?*

## Appendix C

Did image a1.jpg gave the response 'centipede'?

5 responses

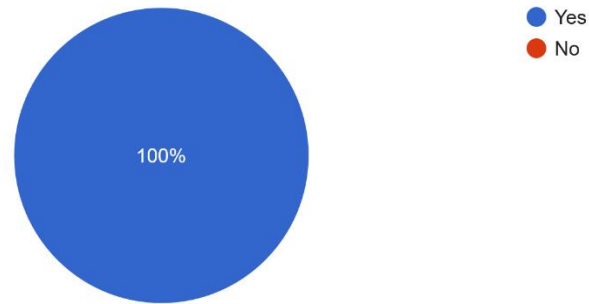


*Figure 7.45: Did image a1.jpg gave the response 'centipede'?*

Did image a2.jpg gave the response 'deer'?

5 responses

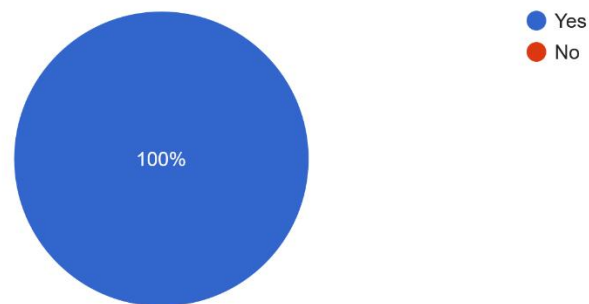


*Figure 7.46: Did image a2.jpg gave the response 'deer'?*

Did image a3.jpg gave the response 'hawk'?

5 responses

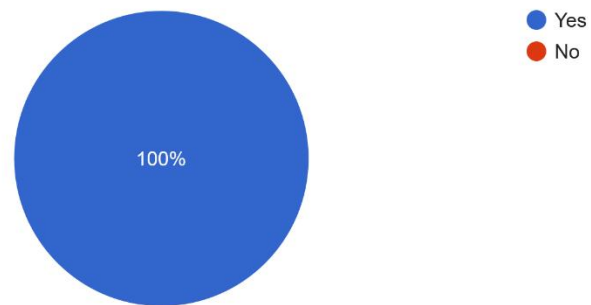


*Figure 7.47: Did image a3.jpg gave the response 'hawk'?*

Did image a4.jpg gave the response 'goat'?

5 responses

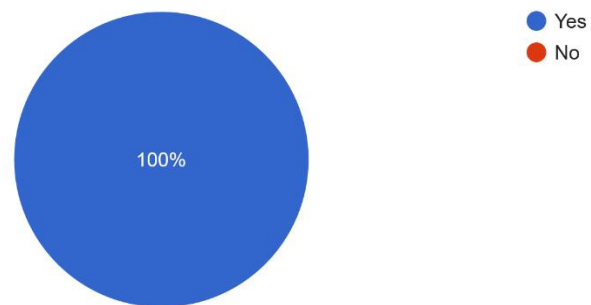


*Figure 7.48: Did image a4.jpg gave the response 'goat'?*

Did image a5.jpg gave the response 'turtle'?

5 responses

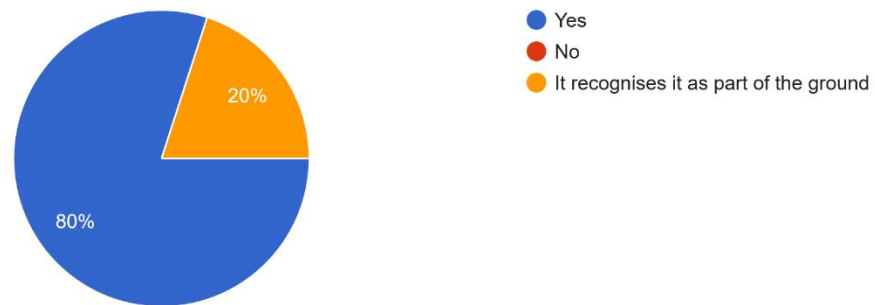


*Figure 7.49: Did image a5.jpg gave the response 'turtle'?*

Did image p1.jpg gave the response 'fig'?

5 responses

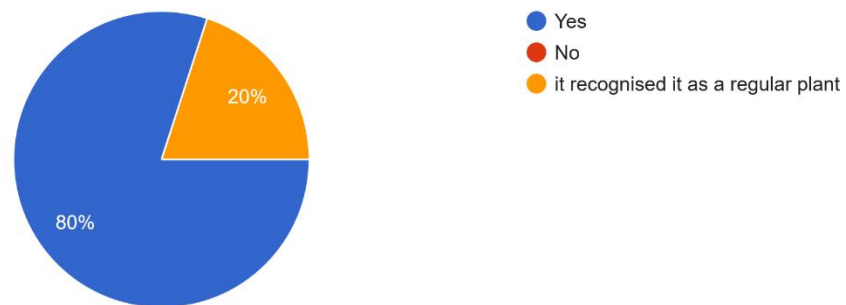


*Figure 7.50: Did image p1.jpg gave the response 'fig'?*

Did image p2.jpg gave the response 'lichen'?

5 responses

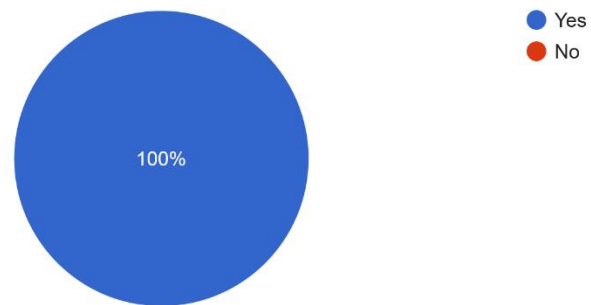


*Figure 7.51: Did image p2.jpg gave the response 'lichen'?*

Did image p3.jpg gave the response 'mushroom'?

5 responses

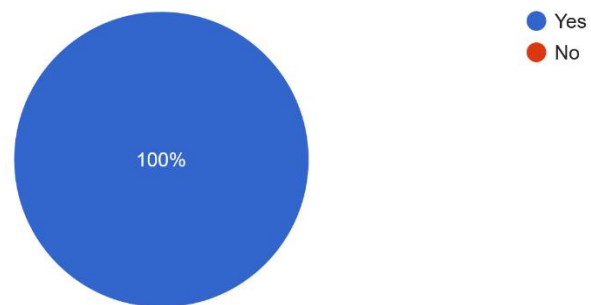


*Figure 7.52: Did image p3.jpg gave the response 'mushroom'?*

Did image p4.jpg gave the response 'lily'?

5 responses

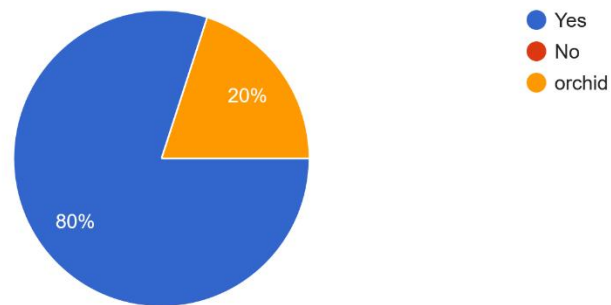


*Figure 7.53: Did image p4.jpg gave the response 'lily'?*

Did image p5.jpg gave the response 'red ginger'?

5 responses

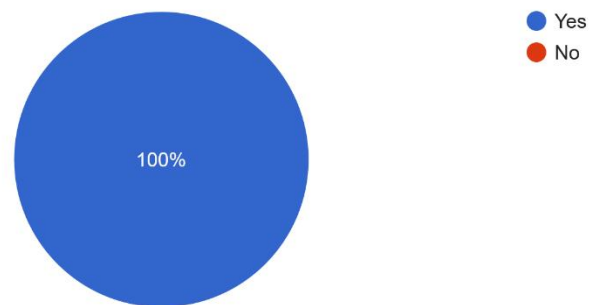


*Figure 7.54: Did image p5.jpg gave the response 'red ginger'?*