MINISTRY OF EDUCATION AND TRAINING

CAN THO UNIVERSITY
COLLEGE OF INFORMATION AND COMMUNICATION TECHNOLOGY
DEPARTMENT OF INFORMATION TECHNOLOGY

GRADUATION THESIS
BACHELOR OF ENGINEERING IN
INFORMATION TECHNOLOGY
(HIGH-QUALITY PROGRAM)

DEVELOPING A MUSHROOM IDENTIFICATION APPLICATION

Student: Nguyễn Quang Vinh
Student ID: B2105727
Class: 2021-2025 (K47)
Advisor: Dr. Bùi Võ Quốc Bảo

Can Tho, 12/2025

MINISTRY OF EDUCATION AND TRAINING

CAN THO UNIVERSITY
COLLEGE OF INFORMATION AND COMMUNICATION TECHNOLOGY
DEPARTMENT OF INFORMATION TECHNOLOGY

GRADUATION THESIS
BACHELOR OF ENGINEERING IN INFORMATION TECHNOLOGY
(HIGH-QUALITY PROGRAM)

DEVELOPING A MUSHROOM IDENTIFICATION APPLICATION

Student: Nguyễn Quang Vinh
Student ID: B2105727
Class: 2021-2025 (Cohort K47)
Advisor: Dr. Bùi Võ Quốc Bảo

Can Tho, 12/2025

ACKNOWLEDGEMENTS

First of all, I would like to express my deep gratitude to Professor Bui Vo Quoc Bao, lecturer at the School of Information and Communication Technology. As an advisor, he has spent a lot of time guiding and supporting me throughout the process of doing the thesis.

I also sincerely thank all the teachers at the School of Information and Communication Technology who have provided me with basic and specialized knowledge throughout the learning process, helping me to complete this thesis.

Finally, I would like to thank my family and friends, those who have always been there to cheer me up and encourage me, helping me overcome the pressures throughout the process of writing this thesis.

Although I have tried my best to complete the topic but due to the limitation of time and practical experience, the thesis cannot avoid many shortcomings. I look forward to receiving comments from the teachers to help my topic be more complete.

Can Tho, ……. 2025

Student

Nguyen Quang Vinh

ABSTRACT

Mushroom identification is challenging due to visual similarities between edible and toxic varieties, often leading to misidentification and serious health risks, particularly for beginners. This thesis develops FungiScan, a beginner friendly Android mobile application that combines real time image recognition with complementary tools to promote safe foraging.

The application uses a client-server architecture where a Flutter/Dart client handles user interactions, while a server-side Vision Transformer model, trained on a custom 80-class dataset of 16,000 images assembled from public sources to performs real-time classification, which achieves approximately 89% average accuracy on the test set, with per-class accuracy ranging from 65% to 100%. The application also offers keyword search via a bundled JSON dataset and an interactive forage map using the iNaturalist API.

In conclusion, FungiScan provides an accessible and reliable toolset that integrates machine learning with intuitive mobile design, encouraging safe mushroom identification and self-learning among novice foragers.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

No.	Acronym	Meaning
1	1D	One-dimension
2	2D	Two-dimension
3	AI	Artificial Intelligence
4	API	Application Programming Interface
5	ARM	Advanced RISC Machine
6	CLS	Classification Token
7	CNN	Convolutional Neural Network
8	CSV	Comma-Separated Values
9	CUDA	Compute Unified Device Architecture
10	GIS	Geographic Information System
11	GPU	Graphics Processing Unit
12	IDE	Integrated Development Environment
13	iOS	iPhone Operating System
14	JAX	Just After Execution
15	JSON	JavaScript Object Notation
16	LLM	Large Language Model
17	MLP	Multilayer Perceptron
18	NPL	Natural Language Processing
19	OSM	OpenStreetMap
20	SQL	Structured Query Language
21	TPU	Tensor Processing Unit
22	UI	User Interface
23	UML	Unified Modeling Language
24	URL	Uniform Resource Locator
25	ViT	Vision Transformer

CHAPTER 1: INTRODUCTION

1. PROBLEM DESCRIPTION

The identification of mushroom species has traditionally relied on methods such as field guides, visual comparison, and consultation with experts. While effective in many cases, these approaches often demand a considerable level of prior knowledge about the subject and can still lead to misclassification, especially for beginners. Many mushroom species display subtle similarities in features such as cap shape, color patterns and gill structure, making cases of misidentification common without experience. With thousands of documented species worldwide and numerous toxic varieties that closely resemble edible ones, misidentification continues to pose risks that range from mild discomfort to severe poisoning and even death.

With the growing interest in outdoor activities [1], hobbyist foraging has increased the demand for accessible tools to assist beginners. Modern smartphones combined with machine learning provide quick and intuitive identification based on photographic input. However, image-based recognition alone is not always sufficient, users might want to look up species by remembered characteristics or verifying what might be fruiting nearby in the current season, or cross reference community observations before deciding to harvest.

This thesis addresses these challenges by developing a comprehensive yet simple mobile solution that combines multiple approaches in a single, accessible application.

2. PURPOSE OF THE STUDY

The purpose of this thesis is to develop FungiScan, a beginner friendly system, designed primarily for beginner hobbyists seeking an approachable way to identify mushrooms.

FungiScan integrates three core functionalities:

- Real-time image-based identification powered by a server-side Vision Transformer model, delivering, confidence score, detailed information and a direct link to a trustworthy online resource.
- A keyword-searchable database using a bundled JSON dataset that allows users to explore mushroom species by physical characteristics, smell, habitat, toxicity, and other traits even without taking a photo.
- An interactive forage map using the iNaturalist API to display geolocated community observations of forageables and other wild species around the user’s location, with adjustable radius, month, limit and taxon filters.

Rather than replacing expert knowledge, FungiScan aims to offer initial guidance that helps users become more informed, avoid dangerous specimens, and pursue further learning when needed.

3. THESIS OBJECTIVES AND SCOPE

3.1. Research Objectives

- Develop FungiScan, an Android mobile application that enables safe and reliable mushroom identification for beginner hobbyists.
- Providing real-time image-based mushroom recognition using a server-side Vision Transformer model. Along with an on-device keyword search powered by a bundled JSON dataset and an interactive forage map using the iNaturalist API to display seasonal, geolocated community observations.
- Build a lightweight client-server architecture that delivers fast inference results.
- Create an intuitive, beginner-oriented user interface and experience that encourages safe foraging practices and self-learning.

3.2. Research Scope

- Development of the FungiScan Android application using the Flutter framework, targeting devices running Android 13.0 (API 33) and above.
- Implementation and deployment of a Vision Transformer inference model on a cloud server (Lightning.ai), with the server returning only the predicted class index and confidence score. Along with the keyword search feature and the interactive forage map powered by the iNaturalist public API.
- Design and integration of local storage (SQLite) for identification history and a bundled JSON dataset containing comprehensive, searchable mushroom information (descriptions, characteristics, safety tips and external links).

4. THESIS CONTENT

The thesis is organized around the complete development process of the FungiScan mobile application and the AI model. The main implementation phases include:

- **Requirement Analysis:**
  - Researching and collecting functional requirements for a mobile mushroom identification system and the server infrastructure supporting remote AI inference.

- Researching essential components of image-based species recognition workflows, keyword-based lookup, identification history management, and location-based foraging exploration.
- Researching ViT deep learning architectures, Flutter frameworks, public API, and lightweight client–server communication.

- **Design:**
  - Designing a use case diagram, system architecture diagram, and flowcharts describing user interactions with the main features.
  - Designing of the local SQLite storage schema and structure of the bundled JSON dataset.
  - Designing the architecture of the client-server interaction and integration with external links and API.
- **Implementation:**
  - Developing the Android application using the Flutter framework, including camera/gallery integration, permission handling, local SQLite storage, JSON parsing, network communication and other supporting packages.
  - Implementing and deploying a Vision Transformer model using PyTorch, hosted on Lightning.ai.
  - Integrating the iNaturalist API for the forage map feature and implementation of the keyword search using the bundled JSON dataset.
  - Conducting testing on a real device for both app functionality and API interaction to ensure accurate performance under real world conditions.
- **Technology Used:**
  - **Mobile Frontend:** Flutter/Dart and supporting packages.
  - **Backend & AI:** Python, PyTorch, FastAPI, LitServe, ViT model.
  - **Data Storage:** SQLite, bundled JSON.
  - **External APIs & URLs:** iNaturalist public API, external URLs.
  - **Deployment:** Lightning.ai.
  - **Design Tools:** PowerDesigner, StarUML, PowerPoint.
  - **Development Tools:** Android Studio, Lightning.ai Jupyter Notebook IDE, Kaggle.
  - **Testing Tools:** Physical Android device.

5. RELATED WORKS

Several mobile applications have been developed for mushroom identification on the Android platform, using machine learning to assist users in fungi recognition. Notable examples include *"Picture Mushroom – Mushroom ID"* by Next Vision Limited [2] and *"Mushroom Identification"* by Battery Stats Saver [3]. While these applications demonstrate practical utility in field settings, they often rely on heavy subscription-based models, which may impose a hefty monthly fee or daily identification limits. This can restrict access for casual users and beginner foragers. Additionally, many existing apps prioritize functionality over simplicity, making the interface less intuitive for beginners.

The FungiScan app addresses these gaps by offering a platform with a simple, user-friendly interface that allows quick mushroom identification and immediate feedback. Along with a keyword-searchable mushroom information that lets users look up species by descriptions and characteristics as well as an interactive forage map powered by the iNaturalist API showing recent community verified observations around the user's location. By combining these tools in a single, accessible package, FungiScan significantly lowers the barrier to safe mushroom identification and learning.

6. OUTLINE OF THE STUDY

This thesis is structured as follows: Chapter 1 presents the introduction, including the research problem, study purpose, objectives, scope, and related works. Chapter 2 establishes the theoretical basis. Chapter 3 details the system design. Chapter 4 covers the implementation and testing. Finally, Chapter 5 provides the conclusion and potential future directions.

CHAPTER 2: THEORETICAL BASIS

1. RELATED TECHNOLOGIES AND LIBRARIES

1.1. Flutter/Dart

Flutter is an open-source UI framework, developed by Google released in 2017. It enables cross-platform application development from a single codebase. It can compile directly to native ARM code for both iOS and Android. Its performance is achieved through the proprietary rendering engine Skia, which paints every UI component on the screen. The Dart programming language is built into Flutter to provide the language and runtimes that power Flutter apps. Together, this technology stack ensures high performance, consistent user experience, and accelerated development cycles.

1.2. SQLite & JSON

SQLite is a C-language library that implements a small, fast, self-contained, high-reliability and full-featured SQL database engine. SQLite is the most used database engine in the world [4]. It is serverless and requires zero configuration, making it an ideal choice for local data storage within mobile applications.

JSON is a lightweight, open standard data interchange format that uses human readable text to represent structured data as name-value pairs and arrays. Widely adopted for its simplicity and language independence, it has become standard for data exchange between web, mobile applications and servers, as well as for local configuration and static data storage in applications.

1.3. OpenStreetMap

OpenStreetMap (OSM) [5] is a collaborative, open-source mapping project that creates and maintains a free, editable global map dataset, encompassing roads, trails, points of interest, and infrastructure, contributed by a worldwide community of volunteers using tools like GPS devices, aerial imagery, and field surveys. Launched in 2004, OSM has grown into a vital resource for GIS professionals, developers, and enthusiasts, powering thousands of downstream projects without proprietary restrictions.

1.4. PyTorch

PyTorch [6] is an open-source deep learning framework known for its flexibility, speed, and Python native design, featuring dynamic computation graphs that enable rapid prototyping and intuitive model development. Widely adopted in

both academia and industry, it excels in computer vision, NLP, and generative tasks through a rich ecosystem of libraries, strong GPU acceleration, and seamless support for distributed training and production deployment.

1.5. Transformers (HuggingFace)

A transformers library [7] by Hugging Face provides thousands of pretrained models and unified, user-friendly APIs for fine-tuning and inference across NLP, vision, audio, and multimodal tasks. It offers extensive model, built-in tokenizers, and seamless PyTorch, TensorFlow and JAX interoperability, making it the standard for rapid prototyping and deployment of transformer-based models.

1.6. CUDA

CUDA [8] is NVIDIA's parallel computing platform and API that enables general purpose computing on GPUs. It dramatically accelerates deep learning workloads, especially matrix operations and convolutions by exploiting the massive parallelism of NVIDIA GPUs, significantly reducing training and inference time compared to CPU only execution.

1.7. Albumentations

Albumentations [9] is a Python library for image augmentations, widely used in computer vision tasks. It offers a collection of augmentation techniques, optimized for efficiency, which are important for effectively increasing the size and diversity of training datasets.

1.8. Matplotlib

Matplotlib [10] is a Python library for creating static, animated, and interactive visualizations. It is often used to generate curves, visualize sample images before and after augmentation, plot confusion matrices, and produce publication quality figures for performance analysis and reporting.

1.9. Lightning AI Studios

Lightning AI Studios [11] is a cloud platform built for AI development and deployment, providing instant GPUs or TPUs, persistent workspaces, and quick production endpoints. Its inference APIs are powered by LitServe [12], a high-performance open source serving engine that delivers auto batching, auto scaling, request streaming, and OpenAPI documentation for PyTorch models with minimal boilerplate on top of FastAPI.

1.10. FastAPI

FastAPI [13] is a modern, high-performance Python web framework for building APIs with automatic OpenAPI documentation and validation using Python type hints. It delivers fast execution thanks to asynchronous support and Starlette/UVicorn underpinnings. It is used under the hood by LitServe to provide type-safe, asynchronous endpoints, validation, and asynchronous request handling.

2. DEEP LEARNING MODEL

2.1. Deep Learning

Deep learning is a subfield of machine learning that draws inspiration from biological neural systems. It uses multiple layers of artificial neural networks to automatically learn meaningful patterns extracted from raw data. With each layer capturing increasingly abstract features of the input. The data flows from the input layer through multiple hidden layers, where nonlinear transformations are applied. Before reaching the output layer to generate the final prediction. The network's depth, often ranging from tens to hundreds of layers or more, enables it to learn complex relationships that simpler models cannot. These models are typically trained on large datasets using gradient-based optimization and backpropagation, allowing the network to iteratively adjust its internal parameters and improve performance.

With increase in GPU computational powers since the early 2010s, the availability of larger datasets, and breakthroughs to solve problems such as vanishing gradients, deep learning achieved a state-of-the-art performance across image classification, natural language processing, speech analysis, and many other tasks, with Convolutional Neural Networks (CNNs) and Vision Transformers (ViTs) becoming two of its most commonly adopted architectures:

Convolutional Neural Networks (CNNs), introduced in the late 1980s with the LeNet architecture [14], are specialized deep neural networks designed primarily for processing grid-like data such as images, medical scans, audio waveforms and more.

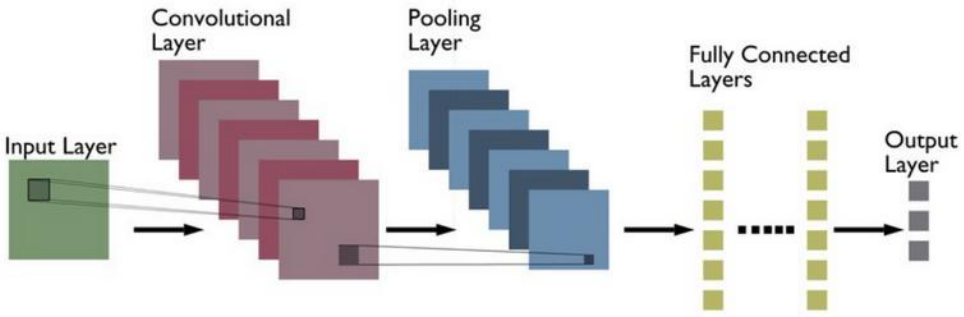

Figure 1. *Average CNN structures [15]*

Vision Transformers (ViTs), introduced by in 2020 in a paper “An Image is Worth 16×16 Words” [16], adapt the transformer architecture originally developed for natural language processing to vision tasks by dividing images into patches and processing them as sequences, achieving state-of-the-art performance in image classification and related computer vision applications.

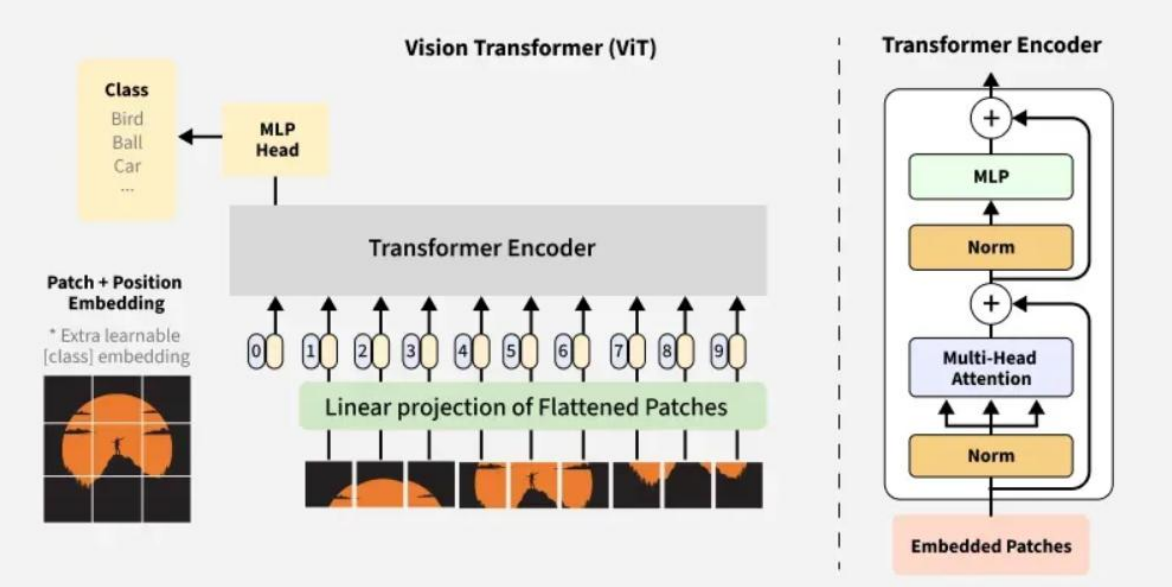

Figure 2. *Average ViT structures [17]*

2.2. Vision Transformers

Vision Transformers (ViT) adapt the transformer architecture that was originally developed for natural language processing, for image classification. An input image is divided into fixed-size non-overlapping patches, typically 16×16 pixels, each flattened and linearly projected into a vector to form patch embeddings. Positional encodings are added to preserve spatial information, converting the 2D image into a 1D sequence of tokens.

A special CLS token is prepended to this sequence. The tokens then pass through multiple transformer encoder layers containing multi-head self-attention and feed-forward networks, stabilized by residual connections and layer normalization. The self-attention mechanism enables every patch to attend to all others simultaneously, capturing long-range dependencies and global context more effectively than traditional convolutional networks.

After the final encoder layer, the CLS token’s embedding aggregates information from the entire image and is fed into a small MLP head with softmax activation to produce the class probabilities. This attention-based approach has

achieved high performance on many image classification benchmarks when trained on sufficiently large datasets.

2.3. Model Evaluation Metrics

True Positive (TP) refers to cases where the model correctly classifies a mushroom image as belonging to class i when that is its actual label, while False Positive (FP) occurs when the model incorrectly assigns an image to class i despite its true label being different. False Negative (FN) represents instances where the model fails to identify an image as class i even though it truly belongs to that class, and True Negative (TN) describes cases where the model correctly identifies that an image does not belong to class i when its actual label is indeed not class i.

Key classification metrics:

1. **Precision:** For a given class, precision measures the proportion of correct positive predictions:

$$\text{Precision}_i = \frac{TP_i}{TP_i + FP_i}$$

2. **Recall:** Also called sensitivity or True Positive Rate, measures the proportion of actual positives that were correctly identified:

$$\text{Recall}_i = \frac{TP_i}{TP_i + FN_i}$$

3. **F1 Score**: Also called harmonic mean of Precision and Recall, particularly useful when dealing with imbalanced classes:

$$\text{F1}_i = 2 \times \frac{\text{Precision}_i \times \text{Recall}_i}{\text{Precision}_i + \text{Recall}_i}$$

4. **Macro-averaged metrics:** Accounting for class imbalance by weighting each class by its support. All classes are treated equally:

$$\text{Macro-Precision} = \frac{1}{N}\sum_{i=1}^{N} \text{Precision}_i,$$

$$\text{Macro-Recall} = \frac{1}{N}\sum_{i=1}^{N} \text{Recall}_i,$$

$$\text{Macro-F1} = \frac{1}{N}\sum_{i=1}^{N} \text{F1}_i$$

Additional evaluation curves:

- Macro-Averaged ROC Curve and its AUC measure the relationship between the true positive rate and false positive rate across different thresholds.
- The Macro F1 Score Curve illustrates how the F1-score changes at various confidence levels, aiding in selecting an operating point that maximizes overall performance.

CHAPTER 3: SYSTEM DESIGN

1. SYSTEM ARCHITECTURE OVERVIEW

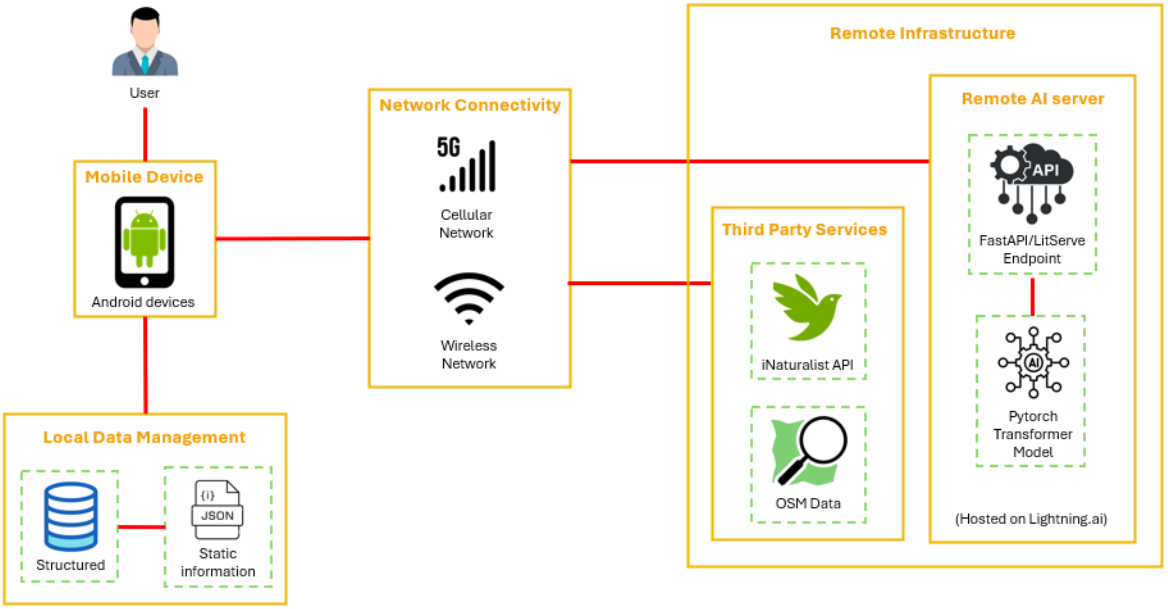

Figure 3. *System Architecture overview diagram*

The application uses a cloud architecture where the client connects to a remote PyTorch Transformer model through a FastAPI/LitServe endpoint hosted on Lightning.AI, via Wireless network for all real time inference. Locally, an SQLite database stores previously inferred results for instant retrieval and offline access, while a static JSON file serves dual purposes of providing data for inferred results and powering an on-device search feature that allows users to browse data of non-inferred items without persisting them in the database. The forage map feature operates independently by directly querying the public iNaturalist API and OpenStreetMap data (via CARTO Basemap) over the network, retrieving data based on user location, radius, month, and taxon filters, and displaying them on an interactive map.

2. USE CASE DIAGRAM

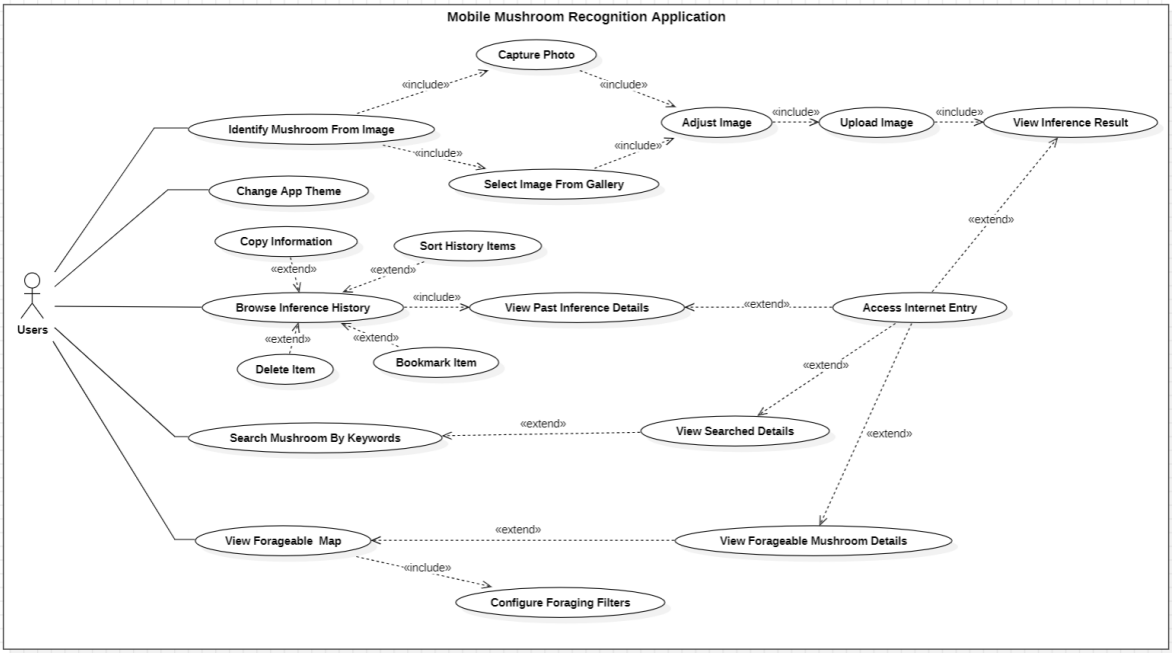

Figure 4. *Use case diagram*

FungiScan consists of these following use cases accessible via a bottom navigation bar:

- Identify Mushroom from Image: capture or select photo, then adjust image (crop, zoom and rotate), then upload image to server for inference and finally view the result with related information and automatically saved to history.
- Change app theme.
- Browse Identification History: view, sort, bookmark, delete or copy content of past inference results and view corresponding Internet entry.
- Search Mushroom by Keyword: text search using the bundled JSON dataset by keywords then view the details.
- View Forageable of various species including mushrooms on an interactive map through iNaturalist API via adjustable filters, then view marker details, upon which user can access the corresponding iNaturalist entry.

3. MAIN FUNCTION OF THE SYSTEM

3.1 Mushroom Identification from Image

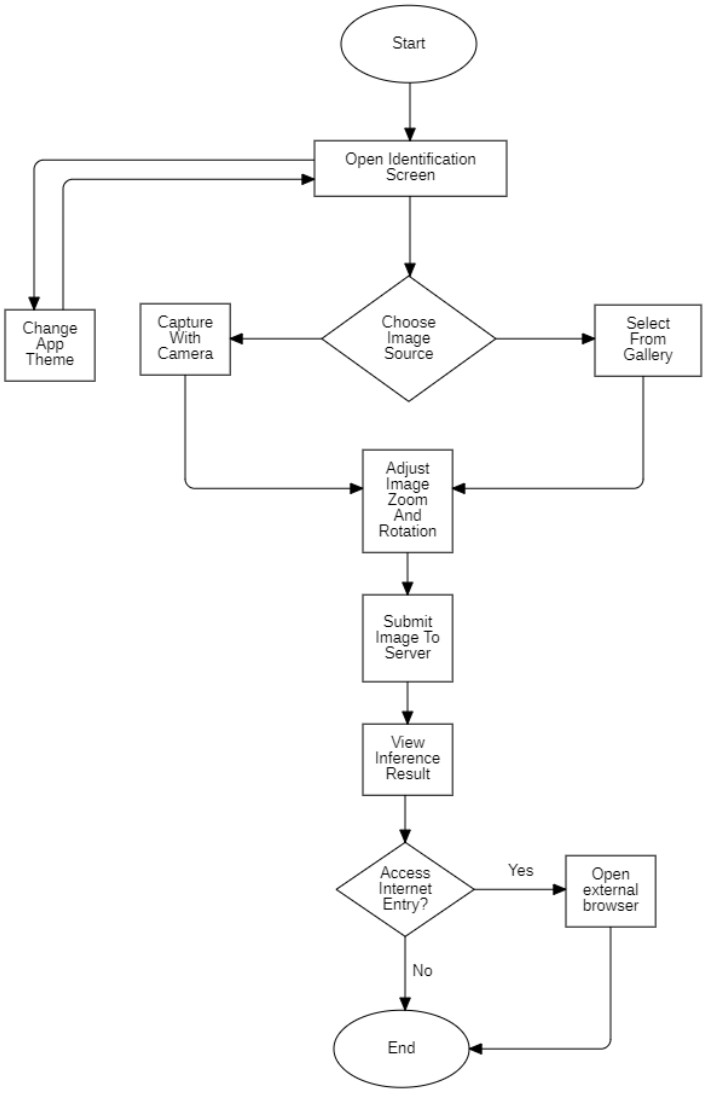

***Figure 5**. Mushroom Identification feature flowchart*

Upon launching FungiScan, the user immediately lands on the Mushroom Identification screen. From here, the user can either change the app theme or choose to capture a new photo via the camera or select an existing image from the gallery. The selected image is then displayed on an adjustment screen where the user can adjust image zoom, crop and rotation to center the mushroom clearly. After confirming with “OK”, the image is sent to the remote server, which runs the Vision Transformer model and returns only the predicted class index and confidence score. The app instantly maps this index to the corresponding entry in the bundled JSON dataset, displaying the full information of the identified species. Finally, the app automatically saves the complete record to local SQLite history. From the result

screen, the user can either return to the identification screen or tap the link icon to open the specific article in an external browser.

3.2. Identification History

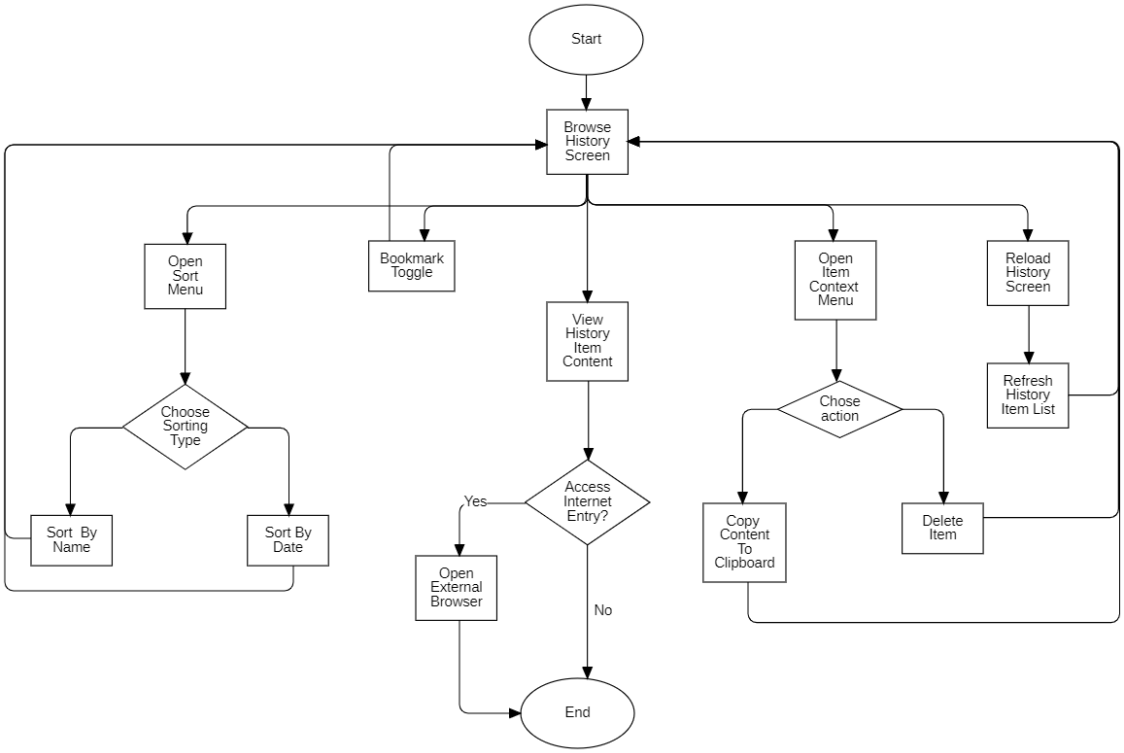

Figure 6. *Identification History feature flowchart*

Users can switch to the history screen via the bottom bar, this screen all past identification results as a scrollable list of cards and are stored locally in SQLite for full offline access. Users can sort the list by name or date and refresh the list using the action buttons on the top bar. Users can also interact with the history item by bookmarking the item or opening the context menu of said item to copy its content to the clipboard or delete it. Finally, users can view the content of the item by tapping on the item, where they can return to the history screen or the corresponding external reference page.

3.3. Mushroom Search

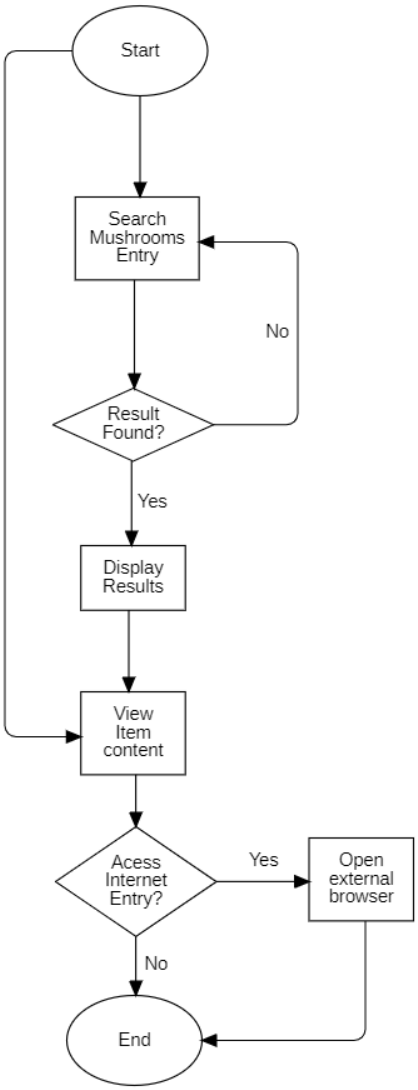

Figure 7. *Mushroom Search feature flowchart*

The Search screen provides a fast keyword lookup powered by the bundled JSON dataset. Upon navigating to the screen, ten mushroom items are immediately displayed as samples. If the desired species is among them, the user can tap it directly to view full details. Otherwise, the user enters any keywords, for example: cap color, smell, habitat, scientific or common name into the search bar; the list instantly filters in real time as characters are typed. Matching results appear as cards, tapping any card opens the complete species details. If no species matches the query, a “No results found.” message is shown. Like all other detail screens, users can optionally open the external reference page.

3.4. Forage map

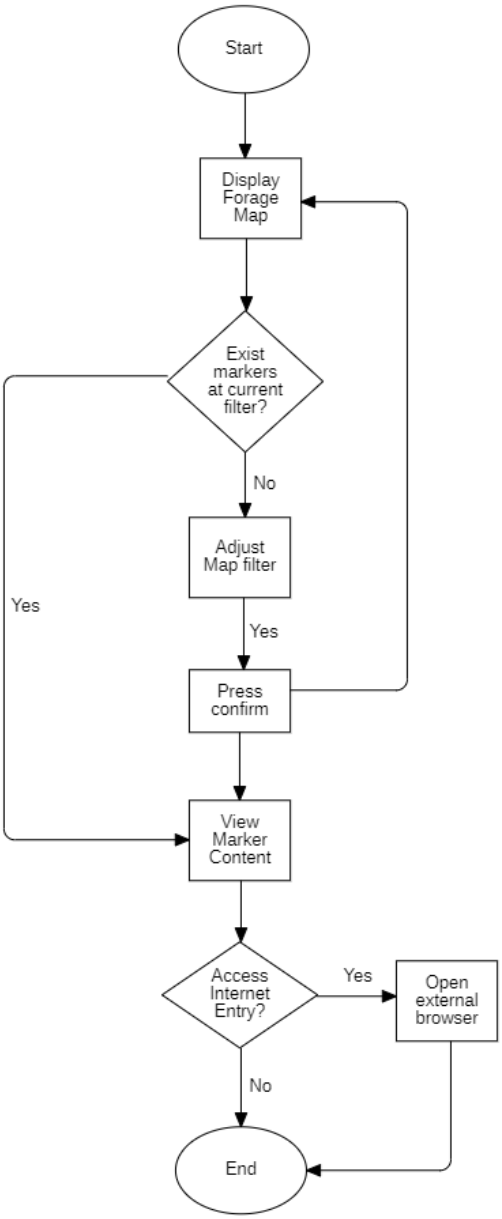

Figure 8. *Forage map feature flowchart*

The Forage Map screen is an online only feature that displays an interactive map centered on the user's current location. Upon opening, the map automatically applies the previously used filter settings or defaults filter settings, to show nearby observations sourced from the iNaturalist public API. If observations exist within range, a red marker appears immediately for each observation, tapping any marker opens a bottom sheet with the actual photo, upload date, common/scientific name, and a direct link to the iNaturalist observation entry. When no observations are found, users can modify the radius, month, observation limit and taxon scope (mushrooms only or all organisms) via a filter button on the top bar, triggering a refresh of markers.

4. LOCAL DATABASE DESIGN

4.1. SQLite table structure

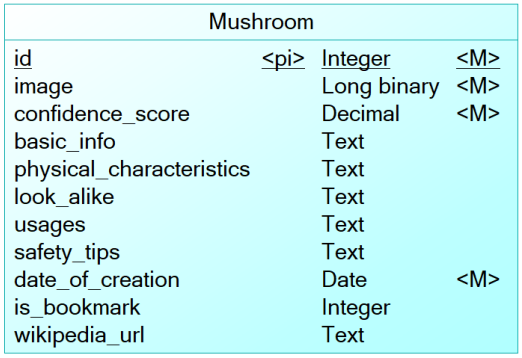

Figure 9. *Database schema*

FungiScan uses a single SQLite table named mushrooms to store inference results locally. After a successful inference, the app receives the predicted class key, for example "amethyst_chanterelle", a looks up of the corresponding entry in the mushroom_info.json file, copies all textual information and the cropped image used for inference got inserted into a self-contained record of the database.

Table 1. *Mushroom*

No	Field name	Data type	Description
1	id	ObjectID	Entry ID, Primary key, auto-increment
2	image	Long Binary	Raw image bytes of the cropped/uploaded photo.
3	confidence_score	Real	Model confidence (0.0 – 1.0)
4	basic_info	String	JSON string of the entire basic_info object (common name, scientific name, edibility, toxicity level, habitat) from the bundled mushroom_info.json
5	physical_characteristics	String	Detailed physical traits
6	look_alike	String	Common look-alikes and warnings
7	usages	String	Culinary/medicinal uses
8	safety_tips	String	Toxicity and handling advice
9	date_of_creation	Date	ISO-8601 timestamp
10	is_bookmark	Integer	Bookmark flag
11	wikipedia_url	String	Direct link to the external reference page (could be Wikipedia or other sources)

The mushrooms table serves to store inferred mushroom information and includes fields such as:

- id: This is the primary key of type ObjectID.
- image has data type Long Binary.
- confidence_score has type Real.
- basic_info, physical_characteristics, look_alike, usages, safety_tips and wikipedia_url have data type String.
- date_of_creation has data type Date.
- is_bookmark has data type Integer.

4.2. JSON format for mushroom information

The application contains a static JSON file in the assets folder. It is used as a source of mushroom information and is used for the search feature and to populate inference results.

The JSON is structured as a top-level object whose keys are unique, URL friendly identifiers, for example: "amethyst_chanterelle", "fly_agaric", etc. Each key maps to a comprehensive object containing all required information for that species.

```
{
  "amethyst_chanterelle": {
    "basic_info": { "common_name": "...", "scientific_name": "...", ... },
    "physical_characteristics": "…",
    "look_alike": "…",
    "usages": "…",
    "safety_tips": "…",
    "search_metadata": {
      "common_name": "Amethyst Chanterelle",
      "keywords": ["amethyst", "chanterelle", "violet", "false gills", ...],
      "image": "https://upload.wikimedia.org/...",
      "wikipedia_url": "https://en.wikipedia.org/wiki/Cantharellus_amethysteus"
    }
  },
  ...
}
```

Figure 10. *JSON file structure*

CHAPTER 4: IMPLEMENTATION

1. MODEL EVALUATION

1.1 Dataset

1.1.1. Data Collection

No image sample was self-collected during this thesis for the custom dataset. The dataset was entirely assembled from publicly available sources. The majority were sourced from iNaturalist observations, supplemented by images from the public Kaggle mushroom dataset [16], Wild Food UK [17], and a small number from various foraging blogs.

1.1.2. Dataset preparation

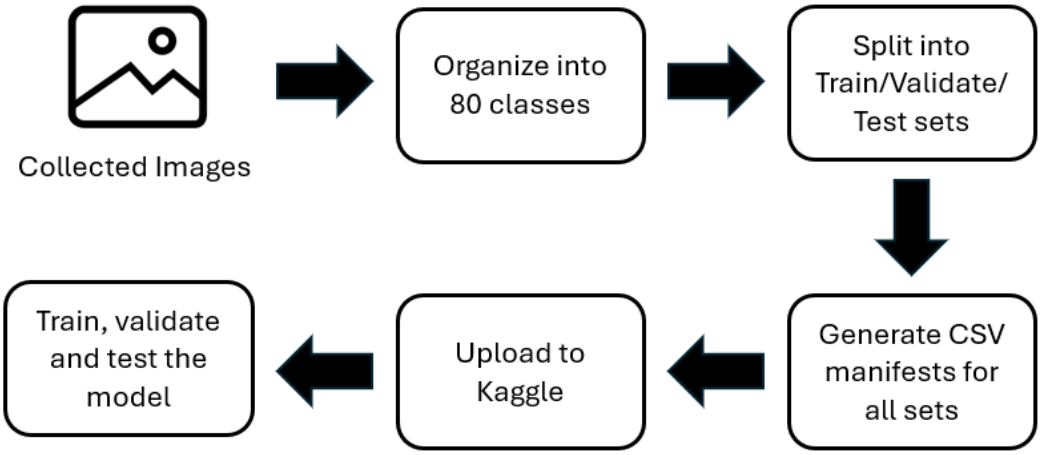

Figure 11. *Dataset preparation process*

Collected images were first organized into 80 folders representing 80 classes with exactly 200 images for each class. A custom Python script was used to randomly split the original dataset into 3 sets of train set, validate set and test set with the ratio of 80-10-10. Another python script is then used to generate three CSV manifest files containing relative file paths and class labels of each image for efficient loading with PyTorch's DataLoader. The dataset, including the split folders and CSV files, was then uploaded to Kaggle as a private dataset. During training, augmentation is applied to the train set, while validate and test sets only undergo resizing and normalization.

1.1.3. Data Augmentation

Augmentation was applied just before training using the Albumentations library to improve model generalization and robustness to real world field conditions. The train set used HorizontalFlip, VerticalFlip, Rotate, RandomBrightnessContrast,

HueSaturationValue, GaussNoise, GaussianBlur, CoarseDropout, and RandomGamma, followed by Resize, Normalize, and ToTensorV2. While the validate and test sets used Resize, Normalize, and ToTensorV2.

1.1.4. Dataset partitioning

Table 2. *Dataset partitioning*

Set	Percentage	Image per class	Total images	Purpose
Train	80%	160	12800	Training the model with data augmentation
Validate	10%	20	1600	Monitoring training results
Test	10%	20	1600	Final performance evaluation
Total	100%	200	16000	

1.1.5. Model Hyper-parameters

Table 3. *Model Hyper-parameters*

Variables	Value
Model	google/vit-base-patch16-224-in21k
Input Size	224x224
Batch Size	64
Learning rate	3e-5 (ViT) and 1e-3 (classifier)
Decay	1e-4
Epoch	32
Scheduler	CosineAnnealingLR with T_max=10
Label smoothing	0.1
Loss function	CrossEntropyLoss
Num classes	80

1.2. Model Evaluation results

Table 4. *Evaluation results*

Model	Macro-Averaged Precision	Macro-Averaged Recall	Macro-Averaged F1 Score	Macro-Averaged AUC
ViT	0.89	0.88	0.88	0.998
ResNet-50	0.85	0.84	0.84	0.997
EfficientNetB4	0.84	0.83	0.83	0.996
MobileNetV3	0.83	0.82	0.82	0.995

The ViT model outperforms all three CNN-based models across the macro-averaged precision, recall, F1 score, and AUC, indicating better overall classification performance on the mushroom dataset.

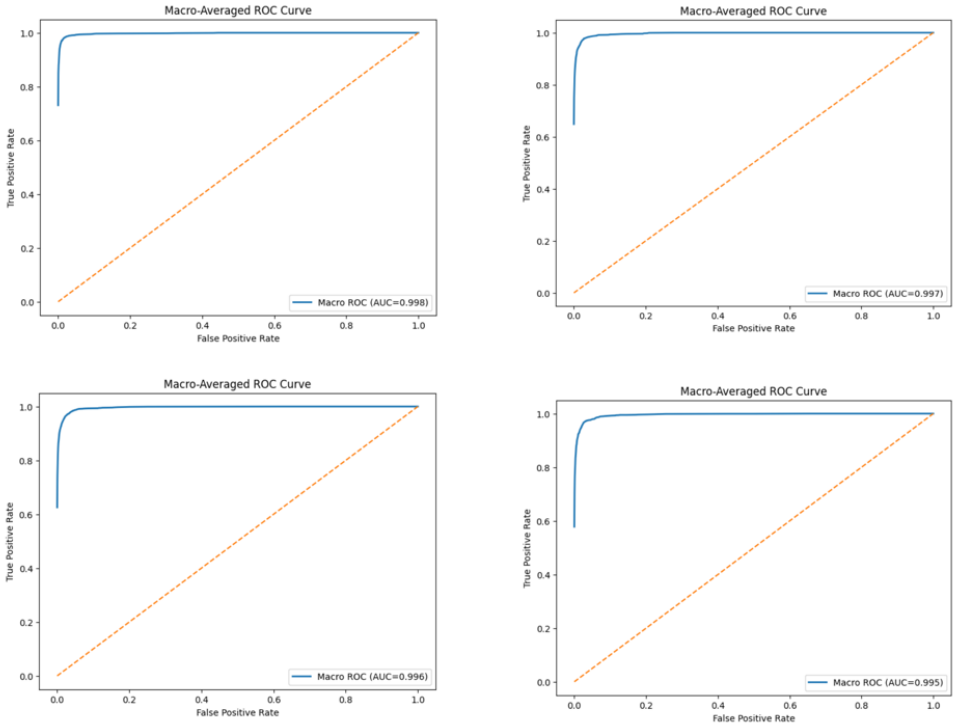

Figure 12. *Marco-Averaged ROC Curves of ViT (top-left), ResNet-50 (top-right), EfficientNetB4 (bottom-left) and MobileNetV3 (bottom-right)*

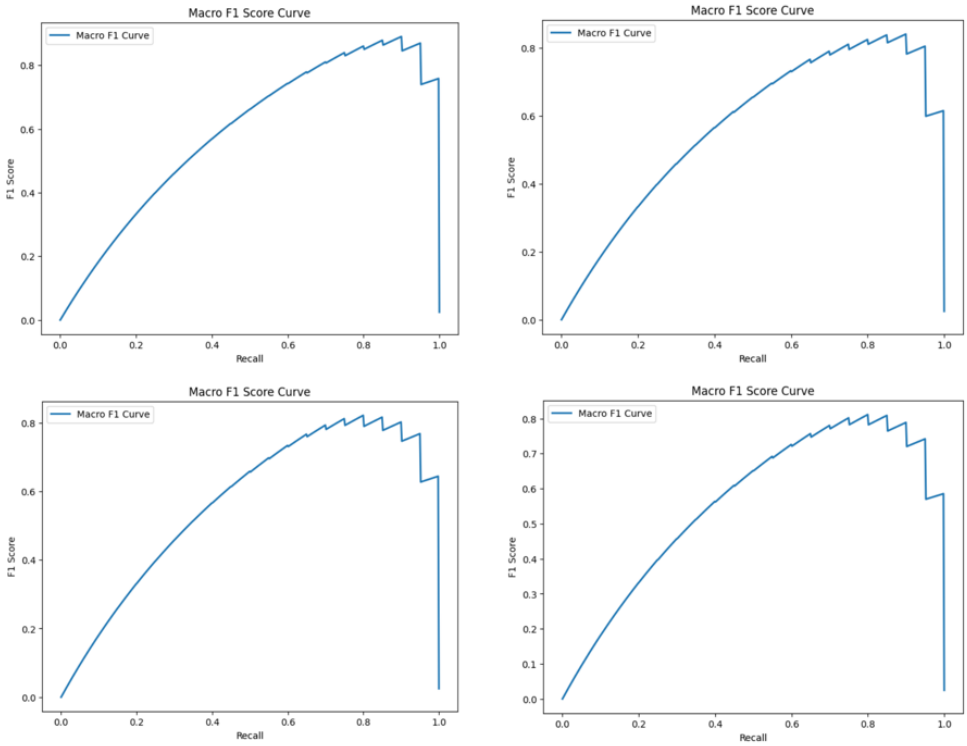

Figure 13. *Marco F1 Score Curves of ViT (top-left), ResNet-50 (top-right), EfficientNetB4 (bottom-left) and MobileNetV3 (bottom-right)*

2. MOBILE APP UI/UX OVERVIEW

2.1. Navigation bar

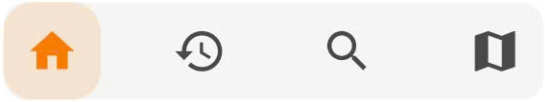

Figure 14. *Navigation bar*

The application has a fixed bottom navigation bar with icons depicting the four main screens of the application, namely, Mushroom Identification (default/home), History, Search, and Forage Map. The navigation bar allows access to all core functions of the app. The currently active tab is highlighted with the primary color to orientate the user.

2.2. Mushroom Identifier Screen (Home Screen)

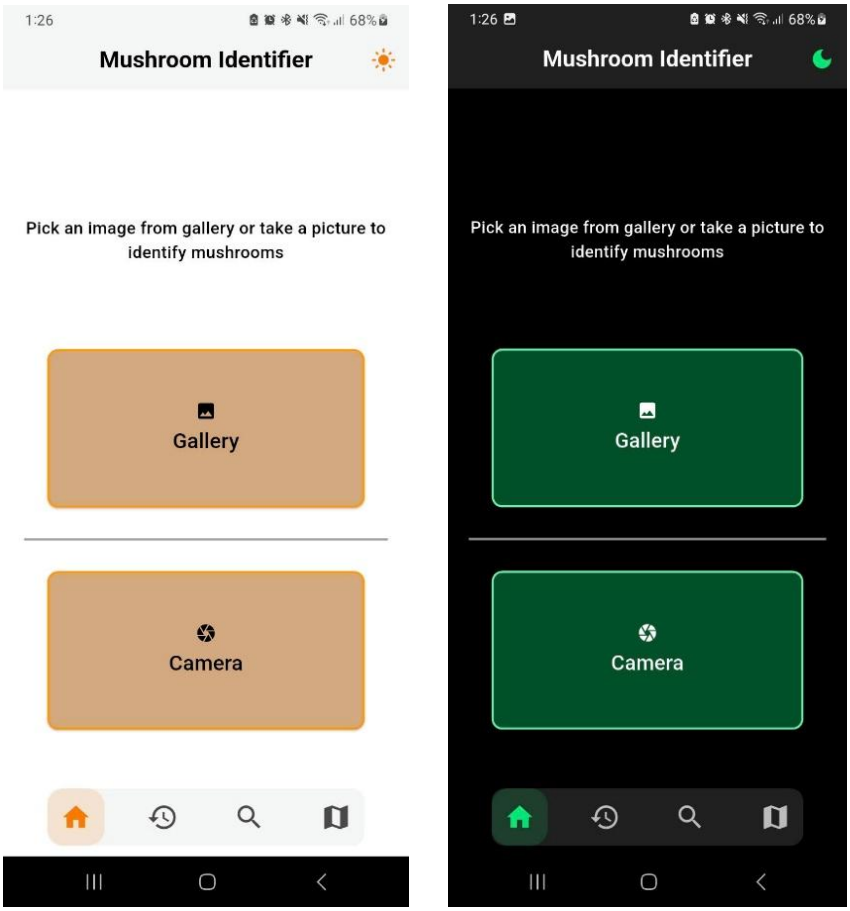

Figure 15. *Mushroom identification screen in light mode (left) and dark mode (right)*

The Mushroom Identification screen is the default page when users launch the app. A sun/moon icon on the right corner of the app bar allows for switching between light and dark themes. Centered on the screen are two buttons, namely the “From Gallery” button for selecting an existing photo and the “Take Photo” button for opening the device camera. This layout ensures that beginners can start an identification in one tap without any distraction.

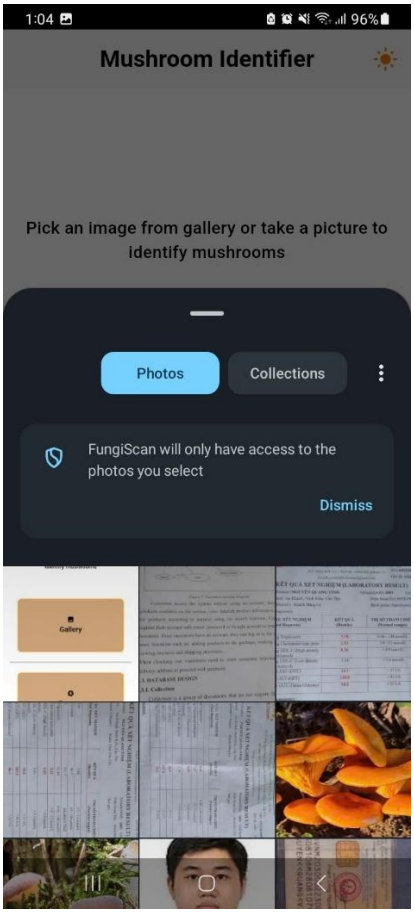

Figure 16. *Gallery picker*

After tapping on the “From Gallery” button, the gallery picker is presented with a standard system interface for choosing a photo from the device. After selection, the user is directed to the Image Editor screen to crop, rotate, or zoom the mushroom into focus before submission.

2.3. Image Editor Screens

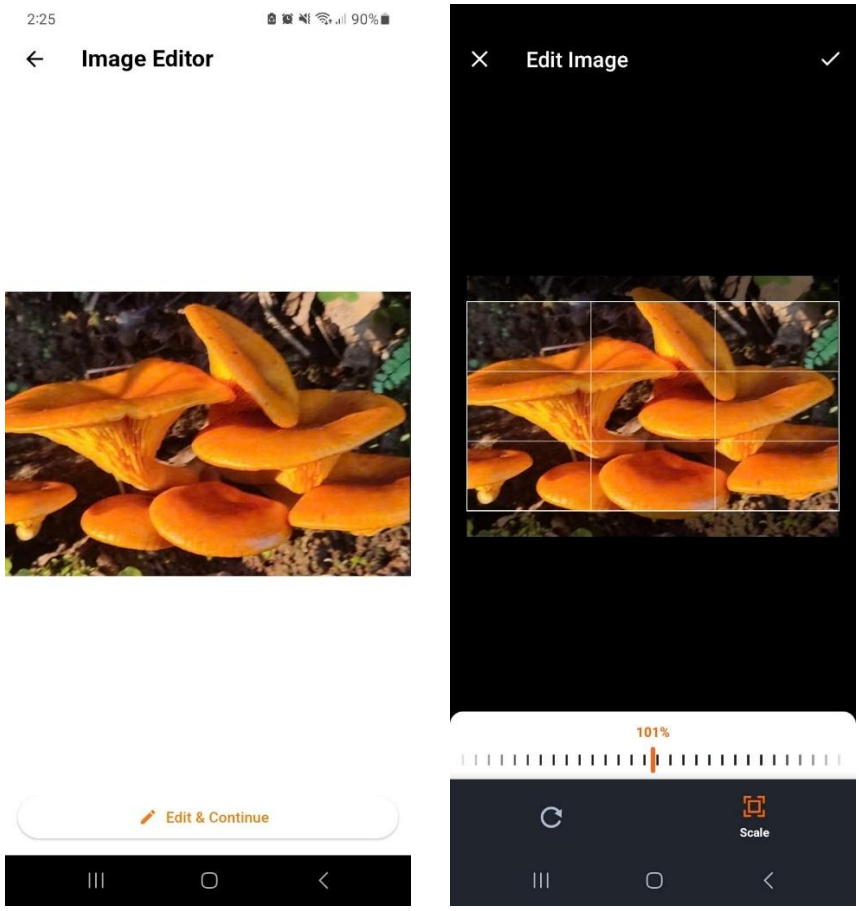

Figure 17. *Image Editor Screens*

After selecting or capturing a photo, the user is taken to the Image Editor screen. The user is first presented with a preview screen to review the captured/selected photo quickly. After pressing the “Edit & Continue” button the user is then taken to the actual editor screen that displays the photo with the options to crop, zoom, and rotate to ensure the mushroom is centered and clearly visible. Once satisfied, the user taps the checkmark to submit the adjusted image to the remote server for inference.

2.4. Inference Progress and Result Display

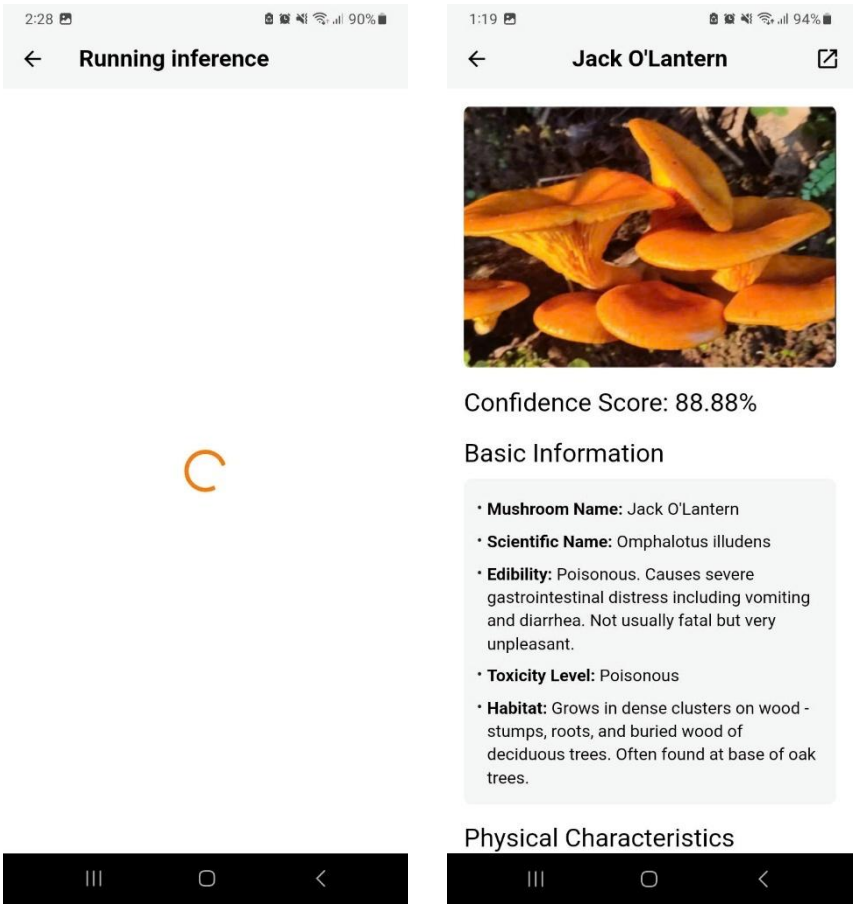

Figure 18. *Inference in progress (left) and detailed result screen (right)*

The inference process consists of two sequential screens shown side-by-side in figure above:

- **Left - Running Inference**: A screen displaying only a centered progress indicator and the title “Running inference” while the image is being uploaded and processed by the server.
- **Right - Result Screen**: Upon receiving the prediction, the app instantly presents the full result where the uploaded photo is at the top, confidence score in large text, followed by sections for Basic Information, Physical Characteristics, Look alike, Usages, and Safety Tips. An external link icon are placed in the app bar for quick access.

2.5. History Screen

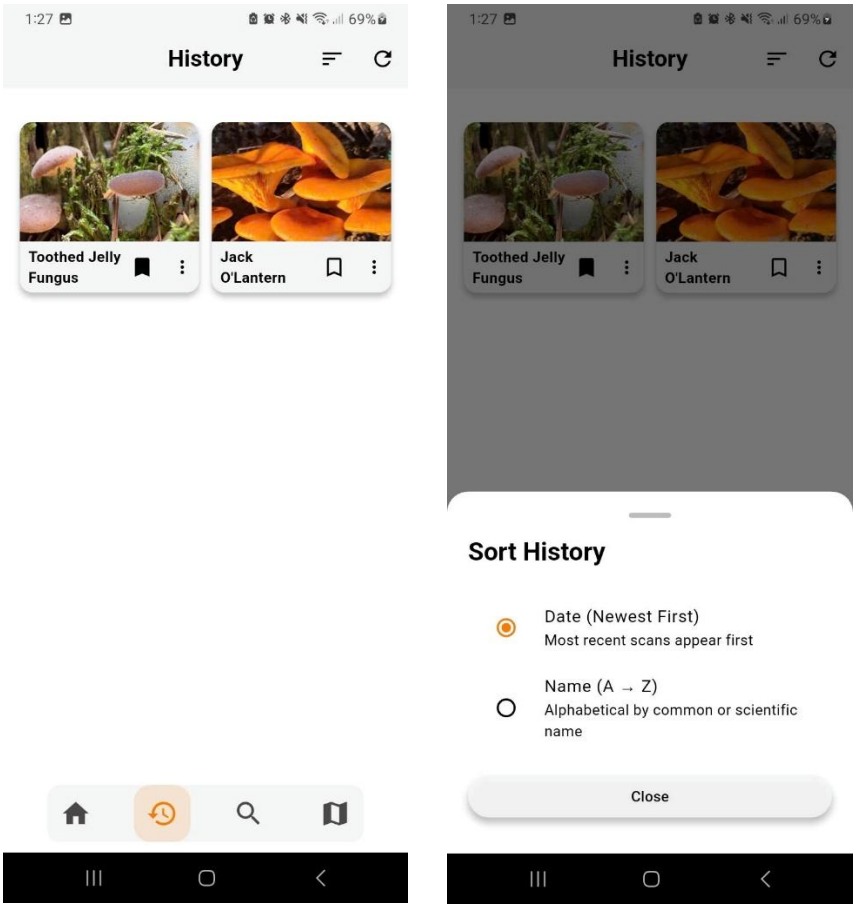

Figure 19. *Default History Screen with overflow menu open (left) and History Screen with Sorting option bottom sheet (right)*

The History screen displays all previous identifications as a grid of cards, each showing the mushroom thumbnail, common name, bookmark icon, and overflow menu. Two action buttons are placed in the top app bar, namely the sorting button for opening a bottom sheet offering sorting options (date or name), while refresh button reloads the list from local storage. Tapping the overflow menu provides options to copy details of or delete an item card.

2.6. History Item Content

Figure 20. *History Item Content Screen*

Tapping any card in the History screen opens the full result view, which is identical to the fresh inference result screen. It displays the original photo, confidence score, and the same information sections. An external link icon remains in the app bar, allowing the user to share the result or open the reference page at any time.

2.7. Search Screen

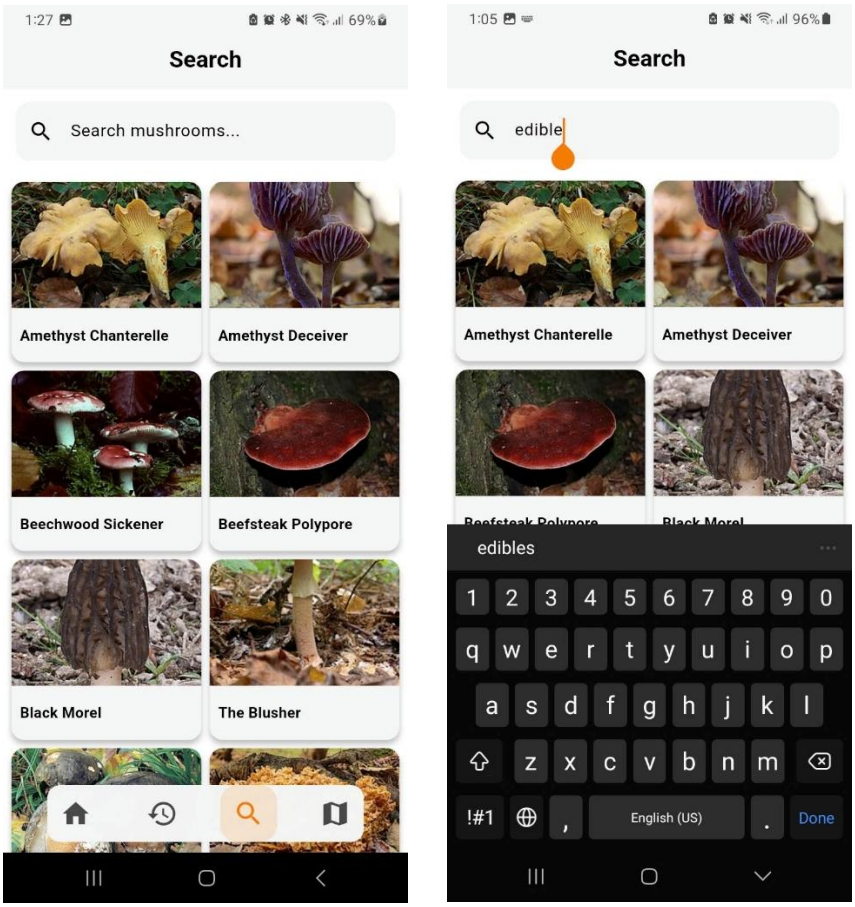

Figure 21. *Default Search Screen (left) and Search Screen with keyword (right)*

The Search screen offers a keyword-based lookup method from the bundled json file. Upon opening, a prominent search bar is displayed at the top with a placeholder “Search mushrooms…”. As the user types, for example: “edible”, results instantly filter and appear as a scrollable grid of cards showing the mushroom photo and common name. When no texts are entered, 10 species are shown by default. Tapping any card opens the same detailed result view used elsewhere in the app.

2.8. Search Result Content

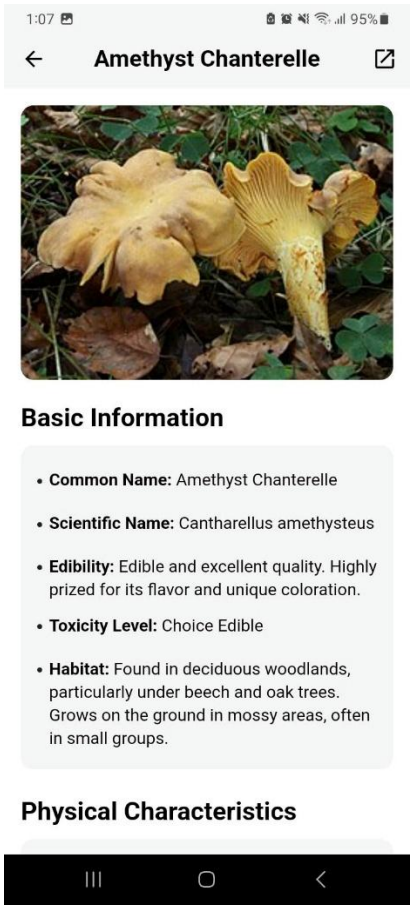

Figure 22. *Search Result Content Screen*

When a user taps any card in the Search screen, the app opens the detailed result view directly from the bundled JSON dataset. The layout is identical to the inference result screen and history item content screen, but the confidence score is intentionally omitted because no inference was performed. A share icon and external link icon remain available in the app bar.

2.9. Forage Map Screen

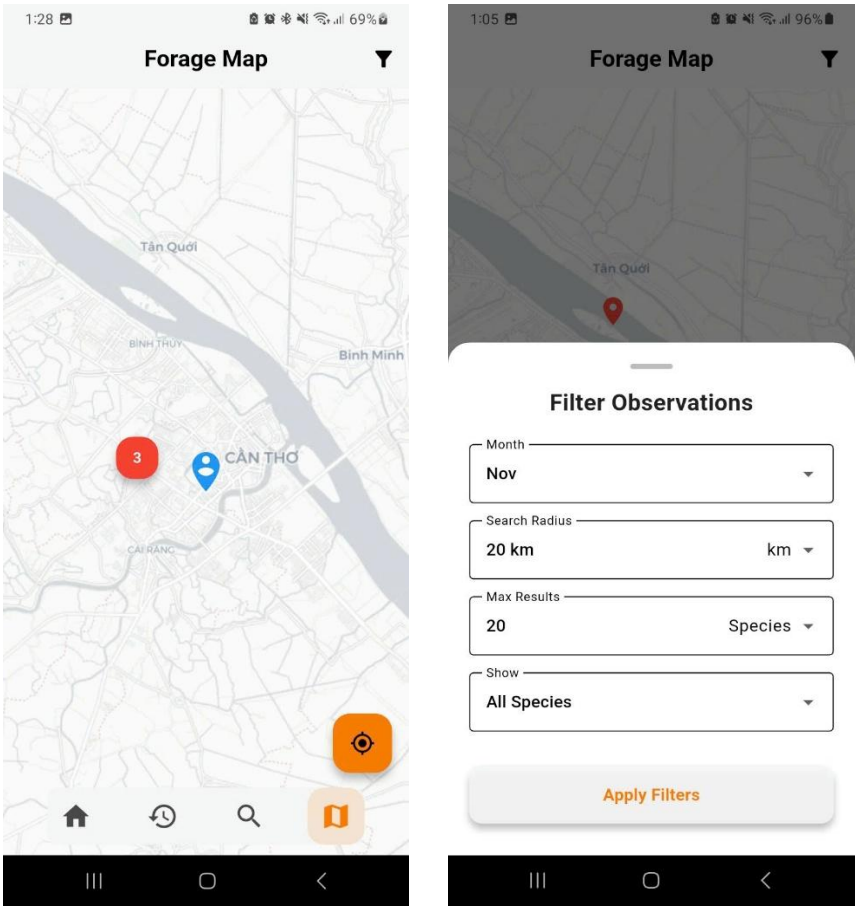

Figure 23. *Default Forage Map Screen (left) and Forage Map Screen with Filter bottom Sheet (right)*

The Forage Map screen integrates real time iNaturalist observation data with an interactive map from OpenStreetMap (via CARTO Basemap) to help users locate nearby forageables and observations. Upon opening, the map is centered on the user's current location, marked by a blue pin. Red markers represent verified iNaturalist observations of either fungi or all other species, depending on the applied filter, within the selected radius. An orange floating action button enables instant centering on the user's current position. The user can access the filter bottom sheet through an action button on the top bar.

Observation data is retrieved dynamically from iNaturalist public API using the user's current latitude and longitude, selected search radius, month filter, result limit, and taxon category. The API returns observations in JSON format. The application loops through each valid observation to extract its coordinates from the geojson field and creates a red pin marker at that location. These markers are added to a list that is rendered onto the map, displaying individual red pins for isolated

observations and clustered red markers with observation counts for densely grouped areas.

Changes are applied instantly upon tapping the “Apply Filters” button, triggering a new API request and subsequent map update. This feature enables foragers to discover seasonal hotspots, plan collection trips effectively, or simply explore local wildlife and surroundings.

2.10. Marker Description

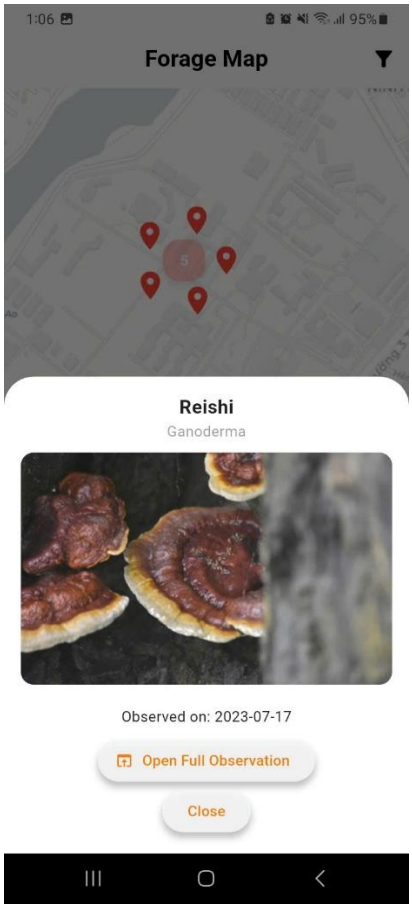

Figure 24. *Marker Description bottom sheet*

Tapping any red marker on the Forage Map opens a bottom sheet displaying the observed species’ common name, scientific name, a community summited photo, the observation date and a “Open Full Observation” button to redirect user to the iNaturalist entry of said observation on the device’s browser, which can further provide access to additional photos, community identifications, and precise coordinates.

3. APP EVALUATION

3.1. Testing Objectives

The testing phase of the FungiScan application is to ensure that the product meets the functional, performance, usability, and reliability requirements defined in previous chapters:

1. **Validate Functional Requirements**: Confirm that all core features operate exactly as specified:
    - Accurate image capture/upload, adjustment, server inference, and result display.
    - Correct functionality of the keyword search and identification history.
    - Correct integration and interaction with the iNaturalist public API.
    - Accurate retrieval and persistence of identification history and bookmarks.
    - Instantaneous and persistent theme switching
2. **Validate Non-Functional Requirements**: Assess performance and responsiveness on real devices:
    - Inference response time must remain acceptable, under 8 seconds on average using 4G or Wi-Fi.
    - App launch time, screen navigation, and list scrolling must feel instantaneous.
3. **Validate Offline Capabilities:** Assess the functionality of offline features:
    - Viewing complete identification history with images and information.
    - Proper restoration of bookmarks and sort preferences.
4. **Evaluate Usability and User Experience:** Confirm that the interface is intuitive and beginner friendly:
    - New users must be able to perform an identification workflow without guidance.
    - All interactive elements must be easily discoverable and provide appropriate feedback.
    - Error messages and empty states must be clear and helpful.
5. **Assess Security and Privacy:** Ensure the application is secured and private:
    - No unnecessary permissions are requested aside from gallery, camera and location.
    - All external links open to the intended sites on the system browser.

3.2. Test Scenario

3.2.1. Functional Testing

Table 5. *Functional Test Scenarios*

No.	Test Scenario	Expected Result	Date Tested
1	Launch application	App opens directly to Mushroom Identification screen, no login required	20/11/2025
2	Take photos using camera	Camera opens, photo is captured and redirected to Image Editor screen	20/11/2025
3	Select photo from gallery	System gallery picker opens, selected image and redirected to Image Editor screen	20/11/2025
4	Crop, rotate, and zoom image in adjustment screen	Image is correctly adjusted and “OK” button sends it to server	20/11/2025
5	Successful inference	Progress screen to result screen with photo, confidence score, and full details	20/11/2025
6	Automatic saving to history after inference	Results are saved and appear immediately in History tab with and thumbnail	20/11/2025
7	View identification history	All previous results displayed correctly with images and information	20/11/2025
8	Bookmark toggles for history items	Bookmark icon toggles correctly, state persists after app restart	20/11/2025
9	Delete history item	Item removed permanently	20/11/2025
10	Sort history by date and by name	List correctly reordered after selecting sort option	20/11/2025
11	Keyword search	Filtering items works instantly, default 10 items shown when empty	20/11/2025
12	Open species detail from search result	Full detail screen, without confidence score, displayed correctly	20/11/2025
13	Forage Map loads observations	Map centered on current user location and red markers appear for nearby observations	20/11/2025

14	User location tracking	Blue user location marker updates on the map after the device moves every 10 meters	20/11/2025
15	Map refresh after significant movement	iNaturalist observations reload and markers refresh after the device moves every 200 meters from the last fetch location	20/11/2025
16	Apply radius, month, result limit and taxon filters	Markers refresh correctly according to selected filters	20/11/2025
17	Tap on marker to view observation bottom sheet	Photo, name, date, and "Open Full Observation" button work correctly	20/11/2025
18	Open external links	System browser opens to correct website	20/11/2025
19	Switch between light and dark theme	Theme changes instantly and persists across screens and after restarts	20/11/2025

3.2.2. Non-Functional Testing

Table 6. *Non-Functional Test Scenarios*

No.	Test Scenario	Expected Result	Date Tested
1	First identification in under 30 seconds	From launch to result displayed in under 30 seconds	20/11/2025
2	Average inference time on 4G or Wi-Fi	Total time under 8 seconds	20/11/2025
3	App launch time	App startup under 5 seconds	20/11/2025
4	Scrolling performance in History and Search	Smooth scrolling.	20/11/2025

3.2.3. Error Handling and Security Testing

Table 7. *Error Handling and Security Test Scenarios*

No.	Test Scenario	Expected Result	Date Tested
1	Inference with no internet connection	Show clear error message with return option	20/11/2025
2	Search screen with no internet	Can still access the mushroom information but thumbnail image display default error image	20/11/2025
3	Forage Map with no internet	Shows Show clear error message with retry option	20/11/2025

4	Server temporarily unreachable	Show clear error message with return option	20/11/2025
5	Deny camera or gallery permission	Shows error message dialogue with options to either cancel or open system settings	20/11/2025
6	Deny location permission	Shows error message dialogue with options to either cancel or open system settings	20/11/2025
7	Verify no unnecessary permissions	Only Camera, Gallery, and Location requested	20/11/2025

3.2.4. Testing Environment

All tests were performed on a real physical Android device to ensure the results reflect actual field usage conditions by the targeted users. No emulators were used.

- **Device used:** Samsung Galaxy A12
- **Android version:** Android 13 (API level 33)
- **Hardware specifications:**

Table 8. *Hardware specifications*

Component	Specification
Processor	MediaTek MT6765 Helio P35 (12 nm) octa-core
CPU	Exynos 850
GPU	Mali-G52
RAM	6 GB
Internal storage	128 GB
Display	6.5 inches LCD, 720 × 1600 pixels
Rear camera	12 MP (wide) + 5 MP (ultrawide)
Battery	5 000 mAh
Network tested	Wi-Fi and 4G

3.3. Test Results

3.3.1. Mushroom Identifier Function

Table 9. *Mushroom Identifier Function test results*

No.	Test Scenario	Expected Result	Date Tested	Result
1	Launch application	App opens in under 5 seconds directly to Mushroom Identification screen, no login required	20/11/2025	Pass

2	Take photos using camera	Camera opens, photo is captured and redirected to Image Editor screen	20/11/2025	Pass
3	Select photo from gallery	System gallery picker opens, selected image and redirected to Image Editor screen	20/11/2025	Pass
4	Crop, rotate, and zoom image in adjustment screen	Image is correctly adjusted and “OK” button sends it to server	20/11/2025	Pass
5	Successful inference	Progress screen to result screen with photo, confidence score, and full details with the total time under 8 seconds	20/11/2025	Pass
6	Open external links	System browser opens to correct website	20/11/2025	Pass
7	Automatic saving to history after inference	Results are saved and appear immediately in History tab with and thumbnail	20/11/2025	Pass
8	Switch between light and dark theme	Theme changes instantly and persists across screens and after restarts	20/11/2025	Pass
9	Average inference time	Under 8 seconds	20/11/2025	Pass
10	First identification in under 30 seconds	From launch to result displayed in under 30 seconds	20/11/2025	Pass
11	Verify no unnecessary permissions	Only Camera and Gallery Location requested	20/11/2025	Pass
12	Inference with no internet connection	Show clear error message with return option	20/11/2025	Pass
13	Server temporarily unreachable	Show clear error message with return option	20/11/2025	Pass
14	Deny camera or gallery permission	Shows error message dialogue with options to either cancel or open system settings	20/11/2025	Pass
15	Force close app during inference	Shows error message dialogue with options to either cancel or open system settings	20/11/2025	Pass

3.3.2. History Function

Table 10. *History Function test results*

No.	Test Scenario	Expected Result	Date Tested	Result
1	View identification history	All previous results displayed correctly with images and information	20/11/2025	Pass
2	Bookmark toggles for history items	Bookmark icon toggles correctly, state persists after app restart	20/11/2025	Pass
3	Delete history item	Item removed permanently	20/11/2025	Pass
4	Sort history by date and by name	List correctly reordered after selecting sort option	20/11/2025	Pass
5	Open external links	System browser opens to correct website	20/11/2025	Pass
6	Scrolling performance in History	Smooth scrolling	20/11/2025	Pass

3.3.3. Search Function

Table 11. *Search Function test results*

No.	Test Scenario	Expected Result	Date Tested	Result
1	Keyword search	Filtering items works instantly, default 10 items shown when empty	20/11/2025	Pass
2	Open species detail from search result	Full detail screen, without confidence score, displayed correctly	20/11/2025	Pass
3	Open external links	System browser opens to correct website	20/11/2025	Pass
4	Scrolling performance in	Smooth scrolling	20/11/2025	Pass
5	Search screen with no internet	Can still access the mushroom information but thumbnail image display default error image	20/11/2025	Pass

3.3.4. Forage Map Function

Table 12. *Forage Map Function test results*

No.	Test Scenario	Expected Result	Date Tested	Result
1	Forage Map loads observations	Map centered on current user location and red markers appear for nearby observations	20/11/2025	Pass
2	Apply radius, month, result limit and taxon filters	Markers refresh correctly according to selected filters	20/11/2025	Pass
3	Tap on marker to view observation bottom sheet	Photo, name, date, and “Open Full Observation” button work correctly	20/11/2025	Pass
4	Open iNaturalist link from marker	Opens correct observation entry in system browser	20/11/2025	Pass
5	User location tracking	Blue user location marker updates on the map after the device moves every 10 meters	20/11/2025	Pass
6	Map refresh after significant movement	iNaturalist observations reload and markers refresh after the device moves every 200 meters from the last fetch location	20/11/2025	Pass
7	Verify no unnecessary permissions	Only Location requested	20/11/2025	Pass
8	Forage Map with no internet	Shows Show clear error message with retry option	20/11/2025	Pass
9	Deny location permission	Shows error message dialogue with options to either cancel or open system settings	20/11/2025	Pass

CHAPTER 5: CONCLUSION AND DEVELOPMENT DIRECTION

1. CONCLUSION

This thesis successfully developed FungiScan a beginner friendly Android mobile application that serves as a tool for safe mushroom identification. By integrating complementary features, the application addresses the real-world problems faced by beginner foragers:

- A real-time image-based identification powered by a server-side Vision Transformer model, with results locally stored after each inference.
- A keyword-based search feature based on a bundled JSON dataset.
- An interactive forage map using INaturalist API to display local, geolocated community observations with adjustable filters.

The project demonstrates a practical integration of machine learning, mobile development, cloud-based inference, and public APIs, contributing a valuable educational and safety tool to the mycology and foraging community.

2. DEVELOPMENT DIRECTION

Due to time and computational resources constraints, I could not implement a large-scale data collection, extended model training, and additional feature development and refinement of current features for the application. Nevertheless, the current system provides a solid and scalable foundation. Thus, I propose some future development directions:

- Expand the recognition scope beyond mushrooms to include other wild edibles (berries, herbs, roots, etc.) either through multi-task models or separate specialized classifiers.
- Implement a larger ViT variant to achieve a more reliable performance.
- Collect a larger and higher-quality dataset as well as refine data augmentation strategies to further mitigate overfitting dataset to further improve model accuracy.
- Upgrade the current keyword-based search feature to a chatbot interface powered by a small LLM that understands natural language queries.
- Add user contributed observation to iNaturalist feature to enrich the forage map with local data.

REFERENCE

[1]. M. Rombach and D. L. Dean, “Exploring Key Factors Driving Urban Foraging Behavior in Garden and Non-Garden Locations,” Foods, vol. 12, no. 5, p. 1032, Feb. 2023. [Online]. Available: https://doi.org/10.3390/foods12051032

[2]. Next Vision Limited, "Picture Mushroom - Mushroom ID," Google Play. Nov. 2025. [Online]. Available: https://play.google.com/store/apps/details?id=com.glority.picturemushroom&hl=en-US&pli=1.

[3]. Milandroid, "Mushroomizer,". Nov. 2025. [Online]. Available: https://repairbatterylife.com/mushroomizer/.

[4]. SQLite Consortium, 2025. [Online]. Available: https://sqlite.org/index.html

[5]. OpenStreetMap Foundation, "About OpenStreetMap," Nov. 2025. [Online]. Available: https://www.openstreetmap.org/about.

[6]. PyTorch Foundation, "PyTorch," Nov. 2025. [Online]. Available: https://pytorch.org/projects/pytorch/.

[7]. Hugging Face, “Transformers Documentation,” Nov. 2025. [Online]. Available: https://huggingface.co/docs/transformers/index.

[8]. NVIDIA, “CUDA Toolkit Documentation,” Nov. 2025. [Online]. Available: https://docs.nvidia.com/cuda/.

[9]. A. Buslaev, V. Iglovikov, E. Khvedchenya, A. Parinov, M. Druzhinin, and A. Kalinin, "Albumentations: Fast and Flexible Image Augmentations," Information, vol. 11, no. 2, p. 125, Feb. 2020. [Online]. Available: https://pypi.org/project/albumentations/

[10]. Matplotlib Development Team, "Matplotlib — Visualization with Python," Nov. 2025, [Online]. Available: https://matplotlib.org/.

[11]. Lightning AI, “Lightning AI Getting Started,” Nov. 2025. [Online]. Available: https://lightning.ai/docs/overview/getting-started

[12]. Lightning AI, “LitServe Home,” Nov. 2025. [Online]. Available: https://lightning.ai/docs/litserve/home

[13]. T. Ramírez, “FastAPI — Documentation,” Nov. 2025. [Online]. Available: https://fastapi.tiangolo.com/.

[14]. Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” Proc. IEEE, vol. 86, no. 11, pp. 2278–2324, Nov. 1998. [Online]. Available: https://ieeexplore.ieee.org/document/726791

[15]. A Kumar, “Different Types of CNN Architectures Explained: Examples”, Nov. 4, 2023. [Online]. Available: https://vitalflux.com/different-types-of-cnn-architectures-explained-examples/

[16]. A. Dosovitskiy et al., "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale,", Oct. 22, 2020. [Online]. Available: https://arxiv.org/abs/2010.11929

[17]. GeeksForGeeks, "Vision Transformer (ViT) Architecture”, Jul. 23, 2025. [Online]. Available: https://www.geeksforgeeks.org/deep-learning/vision-transformer-vit-architecture/

[18]. D. Onishchenko. (2023) Mushrooms images classification 215 [Dataset]. Kaggle. Sep. 2025. [Online]. Available: https://www.kaggle.com/datasets/daniilonishchenko/mushrooms-images-classification-215.

[19]. Wild Food UK, Wild Food UK - Foraging trips and courses, UK. Sep. 2025. [Online]. Available: https://www.wildfooduk.com/.