SUBJECT SPECIFIC FLASHCARD READER

A PROJECT REPORT

for

U23AM491 – MACHINE LEARNING

Submitted by

SANJEEVITHA C S 722823243123

in partial fulfillment for the award of the

degree of

BACHELOR OF TECHNOLOGY

in

ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

SRI ESHWAR COLLEGE OF ENGINEERING

(An Autonomous Institution, Affiliated to Anna University - Chennai - 600 025)

NOVEMBER / DECEMBER 2025

SUBJECT SPECIFIC FLASHCARD READER

A PROJECT REPORT

for

U23AM491 – MACHINE LEARNING

Submitted by

SANJEEVITHA C S **722823243123**

in partial fulfillment for the award of the

degree of

BACHELOR OF TECHNOLOGY

in

ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

SRI ESHWAR COLLEGE OF ENGINEERING

(An Autonomous Institution, Affiliated to Anna University - Chennai - 600 025)

NOVEMBER / DECEMBER 2025

DECLARATION

I declare that the work entitled **"Subject Specific Flashcard Reader"** is submitted in partial fulfillment of the requirement for the award of the degree in B. Tech. (AI&DS)., Anna University Chennai, is a record of the my own work carried out by me during the academic year 2025-2026 under the supervision and guidance of **Dr. G. Sathish Kumar,** Associate Professor, Department of Artificial Intelligence and Data Science, Sri Eshwar College of Engineering, Coimbatore. The extent and source of information are derived from the existing literature and have been indicated through the dissertation at the appropriate places. The matter embodied in this work is original and has not been submitted for the award of any other degree or diploma, either in this or any other University.

Student Name	**Register Number**	**Signature**
Sanjeevitha C S	722823243123	

I certify that the declaration made above by the candidate is true.

Dr. G. Sathish Kumar
Associate Professor
Department of AI&DS

BONAFIDE CERTIFICATE

Certified that this project report **"Subject Specific Flashcard Reader"** is the bonafide work of "**SANJEEVITHA C S (722823243123)"** who carried out the project work under my supervision.

SIGNATURE

Dr. M. MOHAMMED MUSTAFA
Head of the Department
Associate Professor & Head
Dept. of Artificial Intelligence and Data Science
Sri Eshwar College of Engineering, Coimbatore

SIGNATURE

Dr. G. SATHISH KUMAR
Supervisor
Associate Professor
Dept. of Artificial Intelligence and Data Science
Sri Eshwar College of Engineering, Coimbatore

Submitted for the Project viva-voce examination held on _____

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

I thank the Almighty for giving me an opportunity to study in a prestigious institution and grace to complete this project successfully.

I also express my gratitude to our Chairperson **Mr. R. Mohanram** and Director **Mr. R. Rajaram,** Sri Eshwar College of Engineering for their endeavor to provide all the facilities we require and the interest they showed in the welfare of the students.

I wish to express my sincere thanks to **Dr. Sudha Mohanram,** the **Principal & Joint Managing Trustee** of our college for providing the necessary infrastructure facilities.

I express my sincere gratitude to **Dr. M. Mohammed Mustafa** the **Head of the Department,** Artificial Intelligence and Data Science**,** for his guidance and providing valuable suggestion and all necessary facilities at the right time for doing this project.

I express my gratitude and sincere thanks to my guide **Dr. G. Sathish Kumar, Associate Professor,** Dept. of Artificial Intelligence and Data Science, for his valuable suggestions and constant encouragement for successful completion of this project.

I would also extend my hearty thanks to my family members and all the friends who encouraged me to do this project successfully.

TABLE OF CONTENTS

ABSTRACT

The Flashcard Reader is an interactive, camera-based learning system designed to enhance the way students study and recall information from physical flashcards. Traditional flashcards require learners to manually look up explanations, images, or additional context, which can be time-consuming and interrupts the learning flow. This project addresses that challenge by using Optical Character Recognition (OCR) to automatically detect the title or keyword written on any flashcard when viewed through a device's camera. Once detected, the system instantly retrieves corresponding information—including explanations, fun facts, images, and videos—from a structured CSV dataset.

The application is developed using a Flask backend that processes incoming image frames, performs real-time OCR using Tesseract, and serves relevant media and textual content to the client. The frontend is built with HTML, CSS, and JavaScript, enabling smooth camera streaming, frame capture, stability detection, and dynamic content presentation. To further improve user engagement, the system integrates Text-to-Speech (TTS), allowing explanations and fun facts to be narrated automatically. This makes the system highly accessible and more effective for visual and auditory learners.

The Flashcard Reader is lightweight, mobile-friendly, and supports immediate feedback, thereby reducing the effort required for students to search for information manually. Its design supports extensibility through CSV updates, allowing educators or learners to expand the flashcard database without modifying the code. Overall, the project demonstrates an efficient combination of computer vision, real-time processing, and web technologies to create an intelligent educational tool that enhances interactive learning and knowledge retention.

LIST OF FIGURES

LIST OF ABBREVIATIONS

OCR	Optical Character Recognition (Used to detect text from camera frames)
CSV	Comma-Separated Values (Dataset storing flashcard keywords, explanations, images, and video links)
TTS	Text-To-Speech (Converts explanation and fun fact into audio output)
UI	User Interface (The webpage layout displaying detected text, media, and details)
HTML	HyperText Markup Language (Creates the structure of the web application interface)
CSS	Cascading Style Sheets (Used for styling and visual design of the application)
API	Application Programming Interface (Communication between frontend and Flask backend routes like /ocr, /lookup, /tts)

CHAPTER 1

INTRODUCTION

Learning through flashcards has been one of the most effective study techniques for students because it promotes active recall, repetition, and quick revision. However, traditional flashcards have a major limitation: students often need to look up additional information, images, examples, or explanations from external sources, which disrupts the flow of learning. In modern digital learning environments, there is a growing need for tools that make this process faster, more interactive, and more accessible.

The Flashcard Reader project addresses this need by transforming the way students interact with physical flashcards. Instead of manually searching for information, the system uses camera-based Optical Character Recognition (OCR) to automatically read the flashcard title when the user points their device's camera at it. Once detected, the application instantly retrieves detailed explanations, fun facts, images, and videos from a pre-defined dataset. This provides learners with a comprehensive understanding of the concept without requiring any manual effort.

This project is implemented as a web-based application using a combination of modern technologies. The frontend, built with HTML, CSS, and JavaScript, manages real-time camera streaming, frame capture, and user interface updates. The backend is developed using the Flask framework in Python, which handles server-side OCR processing, CSV-based database lookup, media retrieval, and text-to-speech (TTS) audio generation. The system also incorporates dynamic features such as stability checking to ensure accurate text detection, automatic narration of explanations, and fallback search options through Google Images or YouTube when a match is not found.

The Flashcard Reader provides a practical and user-friendly solution for students, educators, and self-learners. It removes the dependency on manual browsing, enables multimodal learning (visual + auditory), and offers a scalable structure where new flashcards can be added easily by updating the CSV file. With its ability to combine computer vision, web technologies, and educational content delivery, the project demonstrates an innovative approach to enhancing real-world learning experiences.

CHAPTER 2

LITERATURE REVIEW

A research paper titled “A Hybrid Approach to Detect and Identify Text in Picture” by S. S. Sannakki et al. (2024) presents a combined OCR framework that integrates classical image processing with modern deep-learning–based text extraction techniques. The study demonstrates that hybrid OCR models significantly improve recognition accuracy under challenging conditions such as noise, varying fonts, shadows, and poor lighting. These findings directly relate to the current project, as the Flashcard Reader also relies on real-time OCR to detect flashcard titles from live camera feeds. The paper highlights the importance of preprocessing steps—including grayscale conversion, thresholding, and noise reduction—to enhance OCR performance. This reinforces the preprocessing pipeline used in this project to ensure stable keyword detection across multiple frames.

Another study, “Innovative Wearable Ring Device: Enhancing Accessibility for Visually Impaired Individuals through OCR and TTS Integration with Raspberry Pi” (SSRN Preprint, 2024), focuses on the integration of OCR and Text-to-Speech (TTS) technologies in assistive devices. The wearable ring captures text through a miniature camera, processes it using an OCR engine, and immediately converts the extracted text into speech output. The research demonstrates the effectiveness of combining visual text recognition with audio feedback, particularly for accessibility and learning support. This strongly aligns with the Flashcard Reader project, where detected flashcard keywords are not only displayed with explanations and images but also narrated using TTS. The study further emphasizes the need for lightweight systems, fast processing, and user-friendly interaction—principles that shape the design of the current project.

Together, these two studies provide a solid foundation for developing a real-time OCR and TTS–based educational tool. Their insights validate the technical approach of the Flashcard Reader and support its goal of enhancing learning through automation, accessibility, and multimodal content delivery.

CHAPTER 3

PROBLEM DEFINITION

3.1 EXISTING SYSTEM

In the existing system, students depend on physical flashcards that contain only brief titles or keywords. To understand the concept fully, they must manually search textbooks or browse the internet for explanations, images, or related videos. This slows down learning, creates interruptions, and makes the study process less efficient. Traditional OCR apps can extract text from an image, but they do not automatically link the detected text to meaningful educational content.

Some desktop-based OCR tools, including earlier versions of this project, can read text using a webcam but offer limited interactivity. They do not provide real-time stability detection, structured explanations, multimedia display, or text-to-speech feedback. As a result, existing systems are fragmented, require manual effort, and do not offer a seamless experience that connects physical flashcards with instant digital learning support.

3.2 DISADVANTAGES OF EXISTING SYSTEM

- Traditional flashcards require manual searching for explanations, slowing down the learning process.
- Basic OCR tools often produce inaccurate or unstable text detection.
- No mechanism exists for real-time keyword stability or confirmation.
- Existing systems lack integrated multimedia support such as images and videos.
- No built-in text-to-speech feature, making learning less interactive and less accessible.

3.3 PROPOSED SYSTEM

The proposed Flashcard Reader system introduces a fully automated, real-time learning tool that bridges the gap between physical flashcards and digital educational content. Instead of manually searching for explanations, the user simply points their device camera at a flashcard, and the system uses Optical Character Recognition (OCR) to detect the keyword written on it. A stability-checking mechanism verifies that the detected text is clear and consistent across several frames, ensuring accurate recognition even in varying lighting or hand-movement conditions.

Once the keyword is confirmed, the system instantly retrieves related information from a structured CSV dataset containing categories, explanations, fun facts, images, and video links. The Flask backend serves this data to the frontend, which dynamically displays the results in a clean and user-friendly interface. Additionally, the system integrates Text-to-Speech (TTS) to read the explanation aloud, supporting both visual and auditory learners. If a matching flashcard is not found, the system provides fallback options with direct Google Images and YouTube search links. This combination of OCR, dynamic content retrieval, multimedia support, and TTS makes the proposed system a smart, accessible, and interactive educational assistant.

3.4 ADVANTAGES OF PROPOSED SYSTEM

- Provides automated and highly accurate flashcard text detection using real-time OCR and stability checks.
- Instantly retrieves structured learning content such as explanations, fun facts, images, and videos from the dataset.
- Enhances learning with multimedia elements and text-to-speech, supporting both visual and auditory learners.
- Reduces manual effort by automating detection, lookup, display, and audio narration.
- Offers a fast, interactive, and user-friendly learning experience with minimal user input.

CHAPTER 4

SYSTEM STUDY

4.1 Feasibility Study

A feasibility study evaluates the practicality and effectiveness of the proposed Subject Specific Flashcard Reader. It examines whether the project can be successfully developed, deployed, and maintained using the available tools, technologies, and resources. This study ensures that the system is economically affordable, technically achievable, and socially beneficial for its intended users. By assessing these factors, the feasibility study confirms that the system offers a viable and efficient solution for enhancing real-time learning through automated text detection, multimedia content retrieval, and text-to-speech support.

4.1.1 Economical Feasibility

The system is highly economical because it relies entirely on free and open-source tools such as Flask, OpenCV, Tesseract OCR, and gTTS. No additional hardware investments are required beyond a standard smartphone or laptop camera, making it cost-efficient for students, teachers, and institutions. The overall development and maintenance cost is minimal, making the system financially viable.

4.1.2 Technical Feasibility

The proposed system is technically feasible as it uses widely supported technologies and straightforward integration. The backend uses Python and Flask, which are easy to implement and maintain. Tesseract OCR provides accurate text extraction, while JavaScript handles real-time camera capture smoothly. The system runs efficiently on normal hardware and supports both mobile and desktop browsers, ensuring reliable performance without complex infrastructure.

4.1.3 Social Feasibility

The system is socially acceptable because it enhances the learning process by reducing manual effort and providing instant access to explanations, images, and audio descriptions. It supports different learning styles—visual, auditory, and self-paced—and improves accessibility through TTS. Since the system is simple to use and does not require technical expertise, it is well-suited for students, educators, and general learners.

CHAPTER 5

SYSTEM REQUIREMENTS SPECIFICATION

5.1 Hardware Requirements

The hardware requirements define the minimum and recommended configurations needed to run the Flashcard Reader system smoothly.

Minimum Requirements:

- Processor: Intel Core i3 or equivalent
- RAM: 4 GB
- Hard Disk: 250 GB
- Camera: Built-in webcam or external USB camera (720p)
- Display: Standard 13 - 15.6” Monitor
- Network: Basic internet connection.

Recommended Requirements:

- Processor: Intel Core i5/i7 or AMD equivalent
- RAM: 8 GB or higher
- Hard Disk: 256 - 500 GB SSD
- Camera: 1080p webcam
- Display: Full HD Monitor
- Network: High-speed broadband connection

5.2 Software Requirements

The software requirements define the operating systems, programming tools, and frameworks needed to build and run the Flashcard Reader system.

- **Operating System:** Windows 10 / 11, Linux(Ubuntu), or macOS
- **Programming Language:** Python 3.8+
- **Backend Framework:** Flask
- **Frontend:** HTML, CSS, JavaScript
- **Required Libraries:** OpenCV, pytesseract, pandas, numpy, gTTS
- **Additional Tools:** Tesseract-OCR installation, Git, VS Code/PyCharm

5.3 Software Description

The software used in the Flashcard Reader project is primarily open-source, offering flexibility and low cost. Below is a brief overview of the major software components and their roles in the system.

- **Python 3.x:** Core development language used for the backend (Flask), image processing, CSV handling, and TTS orchestration. Python's rich ecosystem simplifies integration of OCR and media-serving components.
- **Flask:** A lightweight web framework that exposes REST endpoints (/ocr, /lookup, /image, /video, /tts) for frontend interaction. Flask handles frame uploads, runs OCR, serves media files (including HTTP Range support for video), and streams generated TTS audio.
- **OpenCV (cv2):** Performs image preprocessing (grayscale conversion, Gaussian blur, thresholding) and any optional image transformations to improve OCR accuracy. Used on frames received from the browser before invoking Tesseract.
- **Tesseract OCR & pytesseract:** Tesseract (the OCR engine) performs character recognition on preprocessed images. pytesseract is the Python wrapper used in the backend. Proper installation of the Tesseract binary on the host machine is required.
- **Pandas & NumPy:** Used for loading and manipulating the CSV flashcard dataset (flashcards2.csv). Pandas enables quick lookup, normalization, and hot-reload of the dataset into memory.
- **gTTS (Google Text-to-Speech):** Generates spoken narration (MP3) for detected explanations and fun facts. The backend produces an audio stream that the frontend plays back. pyttsx3 is listed as an optional offline alternative if internet-free TTS is needed.
- **Frontend (HTML/CSS/JavaScript):** Implements camera access (getUserMedia), frame capture and downscaling, stability checking logic, and dynamic UI updates. JavaScript fetches /ocr and /lookup endpoints and embeds returned images/videos or YouTube if local media is unavailable.

CHAPTER 6

SYSTEM DESIGN

6.1 System Architecture

The system architecture of the proposed Flashcard Reader integrates camera input, OCR processing, flashcard lookup, multimedia display, and text-to-speech into a simple and efficient workflow. The system works across five main layers: Camera Input, OCR Processing, Data Lookup, Media Delivery, and User Interaction..

1. **Camera Input Layer:**
   The browser uses getUserMedia to capture live video frames from the user's camera and periodically sends them to the backend.

2. **OCR Processing Layer:**
   The backend preprocesses the frame using OpenCV and extracts text using Tesseract OCR to identify the flashcard keyword.

3. **Keyword Stability Layer:**
   The frontend checks if the same keyword appears across multiple frames to ensure accurate detection before triggering lookup.

4. **Data Lookup Layer:**
   The backend searches the keyword in the CSV file and retrieves the explanation, fun fact, image path, or video link.

5. **Media & Interaction Layer:**
   The frontend displays the retrieved content and plays audio via gTTS. If not found, Google Images and YouTube links are provided.

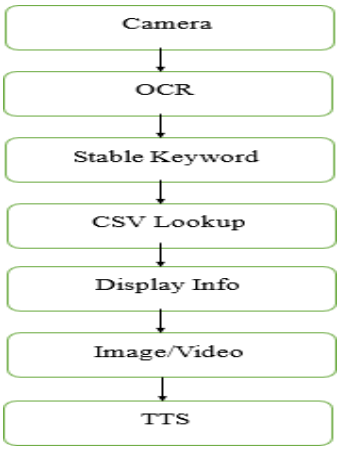

FIGURE 6.1 System Architecture

6.2 Data Flow Diagram (DFD)

The Data Flow Diagram for the Flashcard Reader shows how data moves between the user, the browser (frontend), the Flask backend, and the CSV flashcard database.

DFD Overview

1. **User → Browser (Camera Frames):**
   User shows a flashcard to the camera; browser captures video frames.
2. **Browser → Backend OCR (Image Data):**
   Selected frames are sent to /ocr for text extraction.
3. **Backend OCR → Browser (Keyword):**
   OCR returns the detected text; browser checks stability.
4. **Browser → Lookup Process (Stable Keyword):**
   Once stable, browser sends keyword to /lookup.
5. **Lookup Process → CSV (Fetch Data):**
   Backend searches the CSV for matching flashcard details.
6. **Backend → Browser (Flashcard Info):**
   Explanation, fun fact, image/video links are returned.
7. **Browser → User (Display + Audio):**
   UI displays results and plays TTS audio via /tts.

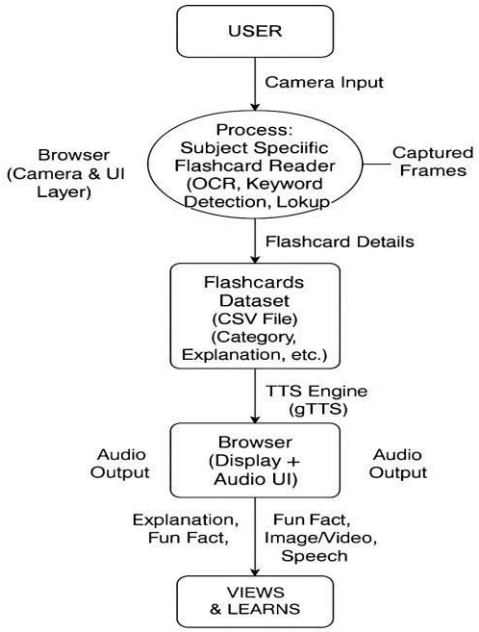

FIGURE 6.2 Data Flow Diagram

CHAPTER 7

SYSTEM IMPLEMENTATION

7.1 Module Description

The Flashcard Reader system is divided into several functional modules that work together to perform real-time flashcard detection, content retrieval, and multimedia output. Each module is responsible for a specific task to ensure smooth operation.

7.1.1 Camera Capture Module

This module captures live video from the device's camera using the browser's getUserMedia API. It continuously streams frames to the frontend, where selected frames are sent to the backend for OCR. The module ensures real-time performance and handles camera start/stop actions.

7.1.2 Image Preprocessing & OCR Module

This module runs on the backend using OpenCV and Tesseract OCR. It converts the received frame into grayscale, removes noise, and applies thresholding to improve text clarity. The processed image is then passed to Tesseract, which extracts the text written on the flashcard. The output is cleaned, normalized, and converted into a keyword.

7.1.3 Keyword Stability Detection Module

Since OCR output may vary frame-to-frame, this module ensures accuracy by comparing keywords across multiple frames. Only when the same keyword is detected consistently (stable), it is sent for lookup. This prevents false triggers caused by motion blur, lighting, or partial visibility.

7.1.4 Flashcard Lookup Module

This module searches the detected keyword inside the flashcards2.csv dataset. It retrieves all matching details such as category, explanation, fun fact, image path, and video link. It returns the results as a JSON response. If the keyword is not found, it prepares fallback Google and YouTube search links.

7.1.5 Media Handling Module

This module manages the retrieval and display of multimedia content.

- If an image exists in the dataset, it is served through the /image endpoint.
- If a video file is available, it is delivered through the /video endpoint with Range support.
- If no local media exists, YouTube embedding or external search links are provided.
- The module also handles missing files gracefully by switching to fallback options.
- It optimizes loading to ensure smooth playback and proper UI display on any device.

The module ensures smooth playback and proper UI display.

7.1.6 Text-to-Speech (TTS) Module

This module uses gTTS to convert the flashcard's explanation and fun fact into speech. The backend generates an MP3 audio stream via the /tts endpoint, and the frontend automatically plays it. This feature improves accessibility, supports auditory learners, and helps users understand the content without reading.

7.1.7 Frontend Display & Interaction Module

This module updates the user interface with the detected keyword, category, explanation, fun fact, and image/video content. It provides visual feedback through stability indicators, category tags, and media previews. The interface is responsive, user-friendly, and allows quick access to external resources like Google Images and YouTube, ensuring a smooth learning experience.

CHAPTER 8

SYSTEM TESTING AND MAINTENANCE

8.1 UNIT TESTING

1. **Camera Capture Module:**
   Verified that the browser successfully accessed the webcam, captured live video frames, and transmitted selected frames to the backend without delays or frame corruption.
2. **OCR & Preprocessing Module:**
   Tested grayscale conversion, noise removal, and thresholding. Ensured Tesseract OCR correctly extracted text from different flashcard images under varying lighting and angles.
3. **Keyword Stability Module:**
   Checked that the frontend accurately counted repeated OCR outputs and triggered lookup only when the keyword remained stable across multiple frames.
4. **Flashcard Lookup Module:**
   Ensured correct keyword matching in the flashcards2.csv dataset. Verified retrieval of category, explanation, fun fact, image path, and video link without mismatches.
5. **Media Handling Module:**
   Tested image loading, video streaming via /video, YouTube embed fallback, and correct handling of missing media files.
6. **Text-to-Speech (TTS) Module:**
   Validated that gTTS generated audio correctly, returned MP3 output, and played it smoothly in the browser.
7. **Frontend Display Module:**
   Checked that explanation, fun fact, image, and video were displayed properly. Ensured UI updated instantly upon keyword recognition.

8.2 INTEGRATION TESTING

After unit tests, all modules were integrated to verify smooth communication:

- Browser sent frames to the OCR backend accurately.
- Stable OCR output triggered correct keyword lookup and media/TTS processing.
- Frontend displayed content and audio smoothly.

This testing ensured that all interconnected components worked together as a unified system.

8.3 FUNCTIONAL TESTING

Functional tests verified whether the system performed as expected from a user's point of view:

- Camera started and stopped correctly.
- Flashcard titles were detected accurately using OCR.
- Stable keywords triggered lookups without false positives.
- Explanations, fun facts, and media were correctly displayed.
- Audio was generated and played without delay.
- External search links (Google Images, YouTube) opened properly.

These tests confirmed that all user-level functions operated smoothly.

8.4 SYSTEM TESTING

The entire system was tested as a complete working unit:

- Real-time workflow: Camera → OCR → Stability → Lookup → Display → TTS .
- OCR accuracy was tested under various environments (brightness, distance, motion).
- System performance was evaluated with large CSV datasets.
- Response time, audio playback speed, and UI rendering were validated.

This ensured overall reliability, accuracy, and responsiveness of the full application.

8.5 WHITE BOX TESTING

Internal system logic and algorithm flow were evaluated

- Verified OCR preprocessing logic and text-cleaning rules.
- Checked stability calculation and correct reset conditions.
- Ensured CSV parsing, path resolution, and JSON responses were accurate.
- Tested TTS generation, media routing, and video Range streaming logic.

White box testing confirmed that internal logic and code execution paths were correct.

8.6 BLACK BOX TESTING

Black box testing validated system behavior based solely on inputs and outputs:

- Tested flashcards with different fonts, sizes, and designs.
- Verified keyword extraction matched actual flashcard titles.
- Checked correct display of explanations, images, videos, and audio.
- Confirmed fallback links appeared for unknown keywords.

This confirmed the system provided accurate, reliable, and user-friendly outputs consistently.

CHAPTER 9

FUTURE ENHANCEMENT AND CONCLUSION

The Subject-Specific Flashcard Reader provides an intelligent and interactive learning solution by combining real-time OCR, structured flashcard lookup, multimedia display, and text-to-speech narration. While the system effectively detects flashcard titles and provides instant explanations, images, videos, and audio output, there are several opportunities for future improvement. The system can be enhanced by integrating more advanced deep learning–based OCR models such as CRNN or Transformer OCR, which would improve accuracy under challenging lighting conditions, angled text, or handwritten flashcards. Support for multiple languages can also be added to make the system useful for a wider range of learners across different regions.

Further enhancements could include developing a dedicated mobile application with offline OCR and offline TTS capabilities, allowing students to use the flashcard reader without internet connectivity. The flashcard dataset can be expanded dynamically, or integrated with external APIs like Wikipedia or Google Knowledge Graph to automatically fetch richer explanations. Features such as spaced repetition, performance tracking, and gamification can turn the system into a complete smart-learning platform. Moving the system to cloud-based storage would also improve scalability, enable collaborative usage, and allow institutions to manage and update flashcard content centrally.

Overall, this project demonstrates how OCR, real-time processing, and multimedia content delivery can transform traditional learning methods into a more engaging and efficient experience. By eliminating manual searching and providing instant, accurate information through visual and audio channels, the Flashcard Reader enhances comprehension and accessibility for students. With future advancements in OCR accuracy, dataset expansion, and mobile integration, the system has strong potential to evolve into a powerful intelligent learning assistant for diverse educational environments.

APPENDIX

A.1 SOURCE CODE

App.py

```
import os
import io
import json
import base64
import tempfile
from urllib.parse import quote_plus
from flask import Flask, render_template, request, jsonify, send_file
import cv2
import numpy as np
import pytesseract
import pandas as pd
from gtts import gTTS
BASE_DIR = os.path.dirname(os.path.abspath(__file__))
CSV_PATH = os.path.join(BASE_DIR, 'flashcards2.csv')
TESSERACT_ENV = os.environ.get('TESSERACT_CMD')
if TESSERACT_ENV and os.path.exists(TESSERACT_ENV):
pytesseract.pytesseract.tesseract_cmd = TESSERACT_ENV
else:
common_paths = [
r"C:\\Program Files\\Tesseract-OCR\\tesseract.exe",
r"C:\\Program Files (x86)\\Tesseract-OCR\\tesseract.exe",]
for p in common_paths:
if os.path.exists(p):
pytesseract.pytesseract.tesseract_cmd = p
break
# Load CSV into memory with hot-reload support
if not os.path.exists(CSV_PATH):
raise FileNotFoundError(f"CSV not found at {CSV_PATH}")
```

```
required_cols = ['category', 'keyword', 'explanation', 'fun_fact', 'image_path', 'video_link']
records_by_keyword = {}
CSV_MTIME = None
def _load_csv_into_memory():
global records_by_keyword
df_local = pd.read_csv(CSV_PATH)
for c in required_cols:
if c not in df_local.columns:
raise ValueError(f"Missing required column '{c}' in CSV")
rbk = {}
for _, row in df_local.iterrows():
key = str(row['keyword']).strip().upper()
rbk[key] = {
'category': str(row['category']).strip(),
'keyword': key,
'explanation': str(row['explanation']).strip(),
'fun_fact': str(row['fun_fact']).strip(),
'image_path': str(row['image_path']).strip() if not pd.isna(row['image_path']) else '',
'video_link': str(row['video_link']).strip() if not pd.isna(row['video_link']) else '',
}
records_by_keyword = rbk
def refresh_data_if_changed():
global CSV_MTIME
try:
mtime = os.path.getmtime(CSV_PATH)
except OSError:
return
if CSV_MTIME is None or mtime != CSV_MTIME:
_load_csv_into_memory()
CSV_MTIME = mtime
refresh_data_if_changed()
app = Flask(__name__)
def _resolve_media_path(path_str: str) -> str:
```

"""Resolve a media path that may be absolute or relative to BASE_DIR.

Also normalize slashes and strip extraneous quotes."""

if not path_str:

return ''

s = str(path_str).strip().strip('"').strip("'")

s = s.replace('/', os.sep)

s = os.path.normpath(s)

if os.path.isabs(s):

return s

return os.path.normpath(os.path.join(BASE_DIR, s))

def preprocess_for_ocr(frame_bgr: np.ndarray) -> np.ndarray:

gray = cv2.cvtColor(frame_bgr, cv2.COLOR_BGR2GRAY)

Light blur (fast) to reduce noise

gray = cv2.GaussianBlur(gray, (3, 3), 0)

Fast global threshold with Otsu

_, th = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)

return th

def ocr_image_from_bytes(image_bytes: bytes) -> str:

np_arr = np.frombuffer(image_bytes, np.uint8)

img = cv2.imdecode(np_arr, cv2.IMREAD_COLOR)

if img is None:

return ''

proc = preprocess_for_ocr(img)

Tesseract config: single line of text is expected (faster)

config = '--psm 7'

text = pytesseract.image_to_string(proc, config=config)

text = text.strip()

Normalize to uppercase single line keyword candidate

text = text.replace('\n', ' ').strip().upper()

Keep only letters, numbers, space, hyphen, and apostrophe

cleaned = []

for ch in text:

if ch.isalnum() or ch in [' ', '-', '\'']:

```
cleaned.append(ch)
text = ''.join(cleaned)
tokens = [t for t in text.split(' ') if t]
if not tokens:
return ''
best = ' '.join(tokens[:4])
if len(tokens) > 4:
candidates = []
for L in range(1, min(4, len(tokens)) + 1):
for i in range(0, len(tokens) - L + 1):
s = ' '.join(tokens[i:i+L])
candidates.append(s)
best = max(candidates, key=len)
return best
@app.route('/')
def index():
return render_template('index.html')
@app.route('/ocr', methods=['POST'])
def ocr_endpoint():
refresh_data_if_changed()
if 'frame' not in request.files:
return jsonify({'text': ''})
f = request.files['frame']
image_bytes = f.read()
text = ocr_image_from_bytes(image_bytes)
return jsonify({'text': text})
@app.route('/lookup', methods=['GET'])
def lookup():
refresh_data_if_changed()
q = request.args.get('q', '').strip().upper()
if not q:
return jsonify({'found': False})
rec = records_by_keyword.get(q)
```

```
if rec:
image_path = _resolve_media_path(rec['image_path']) if rec['image_path'] else ''
image_exists = bool(image_path) and os.path.exists(image_path)
# Support both YouTube links and local video file paths
video_link_raw = rec['video_link'] if rec['video_link'] else ''
resolved_video_path = _resolve_media_path(video_link_raw) if video_link_raw else ''
is_local_video = bool(resolved_video_path) and os.path.exists(resolved_video_path)
return jsonify({
'found': True,
'category': rec['category'],
'keyword': rec['keyword'],
'explanation': rec['explanation'],
'fun_fact': rec['fun_fact'],
'image_available': image_exists,
'image_url': f"/image?q={quote_plus(rec['keyword'])}" if image_exists else '',
# If local video exists, expose via /video; else, pass through original link for YouTube embedding
'video_link': video_link_raw if (video_link_raw and not is_local_video) else '',
'video_available': is_local_video,
'video_url': f"/video?q={quote_plus(rec['keyword'])}" if is_local_video else '',
'google_images': f"https://www.google.com/search?tbm=isch&q={quote_plus(rec['keyword'])}",
'youtube_search':
f"https://www.youtube.com/results?search_query={quote_plus(rec['keyword'])}",})
return jsonify({
'found': False,
'keyword': q,
'google_images': f"https://www.google.com/search?tbm=isch&q={quote_plus(q)}",
'youtube_search': f"https://www.youtube.com/results?search_query={quote_plus(q)}",
})
@app.route('/image')
def image_serve():
refresh_data_if_changed()
q = request.args.get('q', '').strip().upper()
if not q:
```

```
return ('', 404)
rec = records_by_keyword.get(q)
if not rec:
return ('', 404)
path = _resolve_media_path(rec.get('image_path', ''))
if not path or not os.path.exists(path):
return ('', 404)
return send_file(path)
@app.route('/video')
def video_serve():
from flask import request, Response
refresh_data_if_changed()
q = request.args.get('q', '').strip().upper()
if not q:
return ('', 404)
rec = records_by_keyword.get(q)
if not rec:
return ('', 404)
path = _resolve_media_path(rec.get('video_link', ''))
if not path or not os.path.exists(path):
return ('', 404)
file_size = os.path.getsize(path)
range_header = request.headers.get('Range', None)
if not range_header:
return send_file(path)
# Parse Range: bytes=start-end
bytes_unit, _, range_spec = range_header.partition('=')
if bytes_unit.strip().lower() != 'bytes' or '-' not in range_spec:
return send_file(path)
start_str, _, end_str = range_spec.partition('-')
try:
start = int(start_str) if start_str else 0
end = int(end_str) if end_str else file_size - 1
```

```
end = min(end, file_size - 1)
length = end - start + 1
with open(path, 'rb') as f:
f.seek(start)
data = f.read(length)
rv = Response(data, 206, mimetype='video/mp4', direct_passthrough=True)
rv.headers.add('Content-Range', f'bytes {start}-{end}/{file_size}')
rv.headers.add('Accept-Ranges', 'bytes')
rv.headers.add('Content-Length', str(length))
return rv
except Exception:
return send_file(path)
@app.route('/tts')
def tts():
text = request.args.get('text', '').strip()
if not text:
return ('', 400)
# Generate speech with gTTS and stream as mp3
try:
tts = gTTS(text=text, lang='en')
buf = io.BytesIO()
tts.write_to_fp(buf)
buf.seek(0)
return send_file(buf, mimetype='audio/mpeg', as_attachment=False, download_name='tts.mp3')
except Exception as e:
return ('', 500)
if __name__ == '__main__':
app.run(host='0.0.0.0', port=5000, debug=True)
```

A.2 SCREENSHOTS

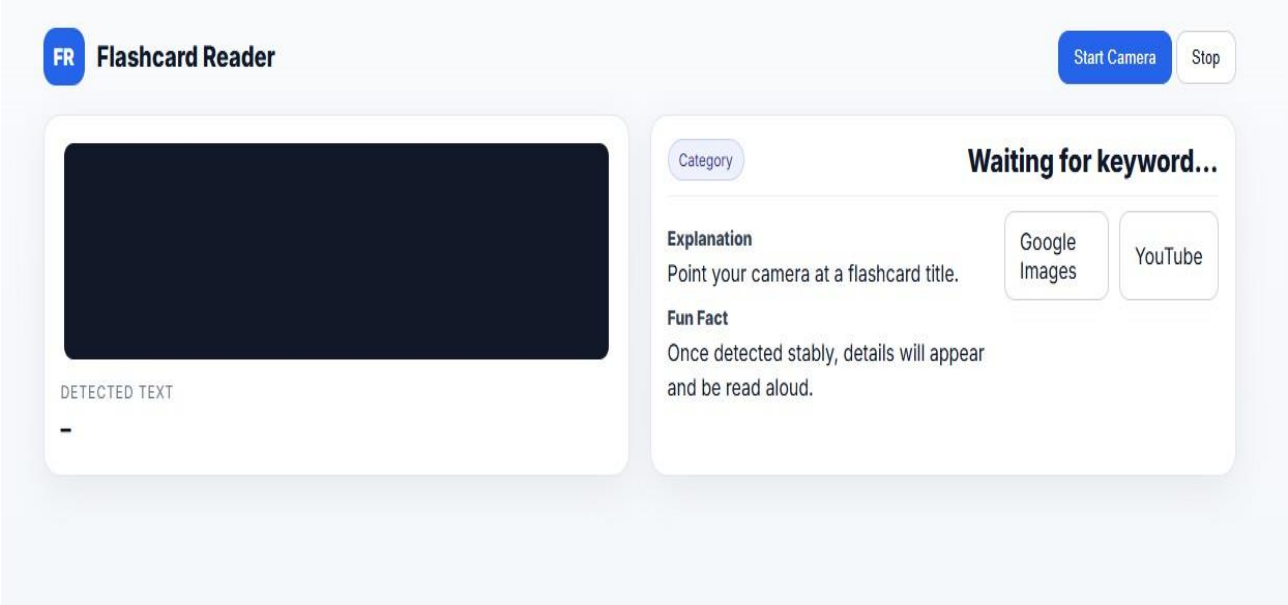

Fig: A.2.1 Home Screen

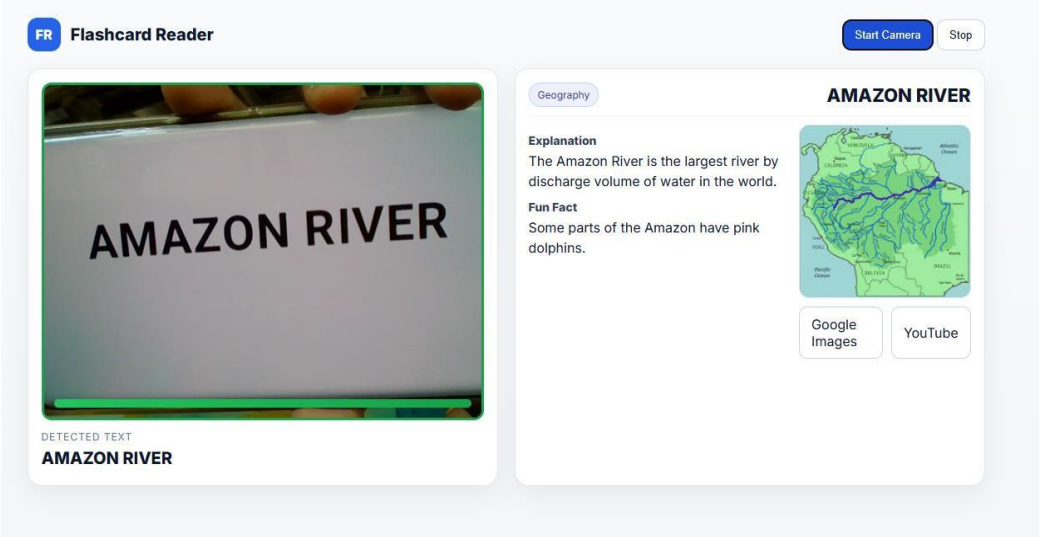

Fig : A.2.2 Output Screen

REFERENCES

1. Smith, R., “An Overview of the Tesseract OCR Engine,” Proceedings of the Ninth International Conference on Document Analysis and Recognition (ICDAR), pp. 629-633, 2007.
2. Badla, S., “Improving the Efficiency of Tesseract OCR Engine,” Master's Projects, 377, San Jose State University, 2014. DOI: 10.31979/etd.9v5v-9z9v
3. Patel, C., Patel, A., Patel, D., “Optical Character Recognition by Open Source OCR Tool Tesseract: A Case Study,” International Journal of Computer Applications, vol. 55, no. 10, pp. 50-56, 2012.
4. Mori, S., Suen, C. Y., Yamamoto, K., “Historical Review of OCR Research and Development,” Proceedings of the IEEE, vol. 80, no. 7, pp. 1029-1058, 1992.
5. Susanto, F. A., Beeh, Y. R., “Pemanfaatan Teknologi Optical Character Recognition (OCR) Untuk Mengenali Alfabet Yunani Berbasis Android,” Artikel Ilmiah, Universitas Kristen Satya Wacana, Salatiga, Indonesia, 2015.
6. Thorat, C., Bhat, A., Sawant, P., Bartakke, I., Shirsath, S., “A Detailed Review on Text Extraction Using Optical Character Recognition,” ICT Analysis and Applications (Lecture Notes in Networks and Systems), vol. 314, pp. 719-728, Springer, 2022.
7. Du, Y., Li, C., Guo, R., Yin, X., Liu, W., Zhou, J., Bai, Y., Yu, Z., Yang, Y., Dang, Q., Wang, H., “PP-OCR: A Practical Ultra Lightweight OCR System,” arXiv Preprint arXiv:2009.09941, 2020.
8. Vujović, Ž. Đ., “Classification Model Evaluation Metrics,” International Journal of Advanced Computer Science and Applications, vol. 12, no. 6, pp. 599-606, 2021.
9. Bissacco, A., Cummins, M., Netzer, Y., Neven, H., “PhotoOCR: Reading Text in Uncontrolled Conditions,” Proceedings of the IEEE International Conference on Computer Vision (ICCV), pp. 785–792, 2013.
10. Smeulders, A. W. M., Worring, M., Santini, S., Gupta, A., Jain, R., “Content-Based Image Retrieval at the End of the Early Years,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 22, no. 12, pp. 1349–1380, 2000.