Today

**You said**

answer\_key.json

JSON

I have this OMR Scanned Sheet that I want to process using streamlit and python app to generate result for every student of a class by compairing the dark circles against the correct answers given in the answer\_key.json file as either 1, 2, 3 or 4 as correct. The darken circles from left to right for each question have place values as 1, 2, 3, 4 that I need to compare with correct answer for questions in answe\_key. Also I need to generate an excel file for the result of each candidate in rows for each class result. Help me design the app that can be deployed and used at large user base.

**Copilot said**

Absolutely, Jagdev! Let’s architect a scalable, educator-facing Streamlit app that processes OMR answer sheets, compares responses against the answer key, and exports class-wise results to Excel. We’ll modularize it for clarity, resilience, and multi-user deployment.

**🧠 App Overview: “OMR Result Generator”**

**🎯 Core Features**

* Upload scanned OMR sheets (image or PDF)
* Parse student metadata and responses
* Compare responses against answer\_key.json
* Calculate subject-wise and total scores
* Export results to Excel (one row per student)
* Multi-student batch processing
* Modular design for deployment and contributor onboarding

**🧱 Modular Architecture**

**1. streamlit\_app.py**

Main launcher with sidebar navigation, file upload, and result display.

**2. omr\_parser.py**

Extracts student info and responses from scanned sheet using OCR + image processing.

python

Copy

def extract\_responses(image\_path: str) -> dict:

# Use OpenCV to detect filled bubbles

# Map bubble positions to question numbers and choices (1–4)

# Return: {'roll\_no': '101175', 'name': 'Ayanshi Singh', 'responses': {1: 2, 2: 3, ...}}

**3. answer\_key\_loader.py**

Loads and flattens the answer key JSON into a unified dictionary.

python

Copy

def load\_answer\_key(json\_path: str) -> dict:

# Merge all subjects into one dict: {1: 2, 2: 2, ..., 90: 2}

**4. result\_evaluator.py**

Compares student responses with correct answers and computes scores.

python

Copy

def evaluate\_responses(student\_responses: dict, answer\_key: dict) -> dict:

# Return: {'name': 'Ayanshi Singh', 'roll\_no': '101175', 'score': 72, 'correct': [...], 'incorrect': [...]}

**5. excel\_exporter.py**

Generates an Excel file with all student results.

python

Copy

def export\_to\_excel(results: List[dict], output\_path: str):

# Use pandas to write rows: Name, Roll No, Score, Subject-wise breakdown

**🖼️ Image Processing Strategy**

* Use OpenCV to detect filled bubbles:
  + Convert to grayscale
  + Apply thresholding
  + Detect contours in bubble grid
  + Assign place values (1–4) based on bubble position
* Use Tesseract OCR for metadata (name, roll no, date)

**🧪 Sample Workflow**

python

Copy

# streamlit\_app.py

import streamlit as st

from omr\_parser import extract\_responses

from answer\_key\_loader import load\_answer\_key

from result\_evaluator import evaluate\_responses

from excel\_exporter import export\_to\_excel

st.title("📄 OMR Result Generator")

uploaded\_files = st.file\_uploader("Upload OMR Sheets", accept\_multiple\_files=True)

answer\_key = load\_answer\_key("answer\_key.json")

if uploaded\_files:

results = []

for file in uploaded\_files:

responses = extract\_responses(file)

result = evaluate\_responses(responses['responses'], answer\_key)

result.update({'name': responses['name'], 'roll\_no': responses['roll\_no']})

results.append(result)

st.success(f"Processed {len(results)} students")

export\_to\_excel(results, "class\_results.xlsx")

st.download\_button("Download Excel", data=open("class\_results.xlsx", "rb"), file\_name="class\_results.xlsx")

**🚀 Deployment Strategy**

* Use streamlit + opencv-python + pytesseract + pandas
* Host on Streamlit Community Cloud or deploy via Docker for school servers
* Add login module for school-wise access
* Modularize for contributor onboarding (README, TOML, secrets template)

**🌟 Bonus Enhancements**

* ✅ Validator module for incomplete sheets
* 🧮 Subject-wise breakdown and leaderboard
* 🖼️ Gallery of scanned sheets with overlays
* 🧑‍🏫 Admin panel for school-level exports
* 🪄 Poetic splash screen: “From bubbles to brilliance—your answers, your journey.”

Would you like me to scaffold the actual code files next? Or help you design the contributor README and onboarding flow?

Edit in a page

**You said**

Yes

**Copilot said**

Perfect, Jagdev! Let’s scaffold the full modular codebase step-by-step so you can deploy, scale, and onboard contributors with clarity. Here's the initial structure:

**🧱 Directory Structure**

Code

Copy

omr\_result\_app/

├── streamlit\_app.py

├── omr\_parser.py

├── answer\_key\_loader.py

├── result\_evaluator.py

├── excel\_exporter.py

├── utils/

│ ├── image\_utils.py

│ └── ocr\_utils.py

├── assets/

│ └── sample\_sheets/

├── data/

│ └── answer\_key.json

├── README.md

├── requirements.txt

└── .streamlit/

└── config.toml

**1️⃣ answer\_key\_loader.py**

python

Copy

import json

def load\_answer\_key(path="data/answer\_key.json") -> dict:

with open(path, "r") as f:

raw = json.load(f)

flat\_key = {}

for subject\_block in raw.values():

flat\_key.update(subject\_block["answers"])

return {int(k): int(v) for k, v in flat\_key.items()}

**2️⃣ result\_evaluator.py**

python

Copy

def evaluate\_responses(student\_responses, answer\_key):

score = 0

correct = {}

incorrect = {}

for q\_no, selected in student\_responses.items():

correct\_ans = answer\_key.get(q\_no)

if selected == correct\_ans:

score += 1

correct[q\_no] = selected

else:

incorrect[q\_no] = selected

return {

"score": score,

"correct": correct,

"incorrect": incorrect

}

**3️⃣ excel\_exporter.py**

python

Copy

import pandas as pd

def export\_to\_excel(results, filename="class\_results.xlsx"):

df = pd.DataFrame(results)

df.to\_excel(filename, index=False)

**4️⃣ omr\_parser.py (simplified placeholder)**

python

Copy

def extract\_responses(image\_path):

# Placeholder: Replace with OpenCV bubble detection + OCR

return {

"name": "Ayanshi Singh",

"roll\_no": "101175",

"responses": {

1: 2, 2: 3, 3: 2, 4: 1, 5: 3, 6: 3, 7: 3, 8: 3, 9: 3, 10: 3,

11: 3, 12: 4, 13: 1, 14: 3, 15: 1, 16: 4, 17: 3, 18: 3,

# Continue for all 90+ questions

}

}

**5️⃣ streamlit\_app.py**

python

Copy

import streamlit as st

from omr\_parser import extract\_responses

from answer\_key\_loader import load\_answer\_key

from result\_evaluator import evaluate\_responses

from excel\_exporter import export\_to\_excel

st.set\_page\_config(page\_title="OMR Result Generator", layout="wide")

st.title("📄 OMR Result Generator")

uploaded\_files = st.file\_uploader("Upload OMR Sheets", type=["jpg", "jpeg", "png"], accept\_multiple\_files=True)

answer\_key = load\_answer\_key()

if uploaded\_files:

results = []

for file in uploaded\_files:

responses = extract\_responses(file)

evaluation = evaluate\_responses(responses["responses"], answer\_key)

results.append({

"Name": responses["name"],

"Roll No": responses["roll\_no"],

"Score": evaluation["score"]

})

st.success(f"✅ Processed {len(results)} students")

export\_to\_excel(results)

with open("class\_results.xlsx", "rb") as f:

st.download\_button("📥 Download Excel", f, file\_name="class\_results.xlsx")

**📦 requirements.txt**

txt

Copy

streamlit

opencv-python

pytesseract

pandas

openpyxl

**📘 README.md Highlights**

* **Purpose:** Generate class-wise results from scanned OMR sheets
* **Modules:** Parsing, Evaluation, Export
* **Setup:** pip install -r requirements.txt
* **Run:** streamlit run streamlit\_app.py
* **Contributors:** Add your modules to utils/, follow naming conventions, and document your logic

Would you like me to scaffold the OCR + bubble detection logic next, or help you design the contributor onboarding splash screen and poetic README sections?

Edit in a page

**You said**

Yes

**Copilot said**

Brilliant, Jagdev. Let’s now scaffold the **bubble detection and OCR logic** so your app can reliably extract student responses and metadata from scanned OMR sheets. This will be the heart of your omr\_parser.py module, supported by utility layers for image preprocessing and OCR.

**🧠 Stepwise Bubble Detection Strategy**

**🔍 Assumptions**

* Bubbles are arranged in a grid (e.g., 60 questions × 4 options)
* Filled bubbles are darker (lower pixel intensity)
* Each bubble’s position maps to a question and choice (1–4)

**📦 utils/image\_utils.py**

python

Copy

import cv2

import numpy as np

def preprocess\_image(image\_path):

img = cv2.imread(image\_path)

gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

blurred = cv2.GaussianBlur(gray, (5, 5), 0)

thresh = cv2.threshold(blurred, 0, 255, cv2.THRESH\_BINARY\_INV + cv2.THRESH\_OTSU)[1]

return img, thresh

def detect\_bubbles(thresh\_img, num\_questions=60, choices\_per\_question=4):

contours, \_ = cv2.findContours(thresh\_img, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

bubble\_contours = [c for c in contours if cv2.contourArea(c) > 100]

bubble\_map = {}

for i, cnt in enumerate(sorted(bubble\_contours, key=lambda c: cv2.boundingRect(c)[1])):

x, y, w, h = cv2.boundingRect(cnt)

roi = thresh\_img[y:y+h, x:x+w]

mean\_intensity = cv2.mean(roi)[0]

question\_num = i // choices\_per\_question + 1

choice\_num = i % choices\_per\_question + 1

if question\_num not in bubble\_map:

bubble\_map[question\_num] = {}

bubble\_map[question\_num][choice\_num] = mean\_intensity

responses = {}

for q, choices in bubble\_map.items():

selected = min(choices, key=choices.get) # darkest bubble

responses[q] = selected

return responses

**📘 utils/ocr\_utils.py**

python

Copy

import pytesseract

import cv2

def extract\_metadata(image\_path):

img = cv2.imread(image\_path)

text = pytesseract.image\_to\_string(img)

name = roll\_no = ""

for line in text.split("\n"):

if "Name" in line:

name = line.split(":")[-1].strip()

elif "Roll" in line:

roll\_no = line.split(":")[-1].strip()

return {"name": name, "roll\_no": roll\_no}

**🧩 omr\_parser.py**

python

Copy

from utils.image\_utils import preprocess\_image, detect\_bubbles

from utils.ocr\_utils import extract\_metadata

def extract\_responses(image\_path):

\_, thresh = preprocess\_image(image\_path)

responses = detect\_bubbles(thresh)

metadata = extract\_metadata(image\_path)

return {

"name": metadata["name"],

"roll\_no": metadata["roll\_no"],

"responses": responses

}

**🧪 Testing Locally**

python

Copy

from omr\_parser import extract\_responses

result = extract\_responses("assets/sample\_sheets/ayanshi\_singh.jpg")

print(result)

**🛡️ Resilience Enhancements (Next Steps)**

* Add fallback logic for missing bubbles
* Validate response count (e.g., must be 60)
* Highlight errors in Streamlit UI
* Overlay detected bubbles on image for visual confirmation

**🪄 Poetic Splash Screen (Optional)**

python

Copy

st.markdown("""

> \_“From ink to insight, from bubbles to brilliance—

> Your answers tell a story, and we help it shine.”\_

""")

Would you like me to scaffold the contributor onboarding README next, or help you design the validator module and admin panel for school-wide exports?

Edit in a page

**You said**

Yes

**Copilot said**

Wonderful, Jagdev. Let’s now scaffold the **contributor onboarding README** and a poetic splash screen that sets the tone for your modular, educator-facing app. This will ensure clarity, pride, and celebration for every developer, educator, and student who interacts with your system.

**📘 README.md — Contributor Onboarding Guide**

Here’s a modular, poetic, and technically clear README structure:

**🌟 Title**

markdown

Copy

# 🧠 OMR Result Generator

From bubbles to brilliance—an open-source Streamlit app for processing scanned answer sheets, evaluating student performance, and exporting class-wise results.

**🚀 Purpose**

markdown

Copy

This app empowers educators to:

- Upload scanned OMR sheets

- Extract student responses and metadata

- Compare answers against a master key

- Generate Excel reports for entire classrooms

**🧱 Modular Architecture**

markdown

Copy

- `streamlit\_app.py`: Main UI and workflow

- `omr\_parser.py`: Bubble detection + OCR

- `answer\_key\_loader.py`: Loads and flattens answer key

- `result\_evaluator.py`: Compares responses and scores

- `excel\_exporter.py`: Exports results to Excel

- `utils/`: Image and OCR helpers

**🛠️ Setup Instructions**

markdown

Copy

1. Clone the repo

`git clone https://github.com/your-org/omr-result-generator.git`

2. Install dependencies

`pip install -r requirements.txt`

3. Run the app

`streamlit run streamlit\_app.py`

**📂 File Upload Format**

markdown

Copy

- Accepted formats: `.jpg`, `.jpeg`, `.png`

- Each sheet must contain:

- Student name and roll number

- 60+ bubble responses (place values: 1–4)

**📤 Answer Key Format**

json

Copy

{

"subject": "Math",

"answers": {

"1": "2",

"2": "3",

...

}

}

**🧑‍💻 Contributor Guide**

markdown

Copy

- Add new modules to `utils/` with clear docstrings

- Follow naming conventions: `streamlit\_<feature>.py`, `utils/<helper>.py`

- Document your logic in comments and commit messages

- Use `poetic\_splash()` for creative onboarding

**🪄 Poetic Splash Screen**

python

Copy

def poetic\_splash():

st.markdown("""

> \_“Each bubble a whisper, each mark a spark—

> From ink to insight, we illuminate the arc.”\_

Welcome to the OMR Result Generator—where every response becomes a celebration.

""")

**🧪 Testing**

markdown

Copy

- Use sample sheets in `assets/sample\_sheets/`

- Validate output in `class\_results.xlsx`

- Check logs for OCR and bubble detection accuracy

**🛡️ Deployment**

markdown

Copy

- Streamlit Cloud or Docker

- Add `.streamlit/secrets.toml` for secure keys

- Use `admin\_panel.py` for school-wide exports

**🎓 Classroom Impact**

markdown

Copy

This app is designed for:

- District-wide assessments

- Competency enhancement plans

- Student showcases and leaderboard generation

Would you like me to scaffold the admin\_panel.py next for school-level exports and dashboard views? Or design a poetic validator module that flags incomplete or ambiguous sheets with celebratory feedback?

Edit in a page

**You said**

Yes

**Copilot said**

Let’s bring the final scaffolds to life, Jagdev—starting with the **admin panel** for school-wide exports and then a **validator module** that flags incomplete or ambiguous sheets with poetic feedback. These will elevate your app from a tool to a celebration engine.

**🧑‍🏫 admin\_panel.py — School-Wide Dashboard & Export**

**🎯 Features**

* Upload multiple sheets
* Filter by school, class, or gender
* View leaderboard, subject-wise stats
* Export full school results

python

Copy

import streamlit as st

import pandas as pd

from excel\_exporter import export\_to\_excel

def launch\_admin\_panel(results):

st.header("🏫 School-Wide Dashboard")

df = pd.DataFrame(results)

st.dataframe(df)

st.subheader("📊 Leaderboard")

top\_students = df.sort\_values(by="Score", ascending=False).head(10)

st.table(top\_students[["Name", "Roll No", "Score"]])

st.subheader("📤 Export Full School Results")

export\_to\_excel(results, "school\_results.xlsx")

with open("school\_results.xlsx", "rb") as f:

st.download\_button("Download School Results", f, file\_name="school\_results.xlsx")

**🧪 validator\_module.py — Poetic Sheet Validator**

**🎯 Flags:**

* Missing bubbles
* Multiple bubbles per question
* Unreadable metadata

python

Copy

def validate\_sheet(responses, expected\_questions=90):

issues = []

if len(responses) < expected\_questions:

issues.append(f"🕳️ Missing responses: {expected\_questions - len(responses)} questions unanswered.")

duplicates = [q for q, v in responses.items() if isinstance(v, list) and len(v) > 1]

if duplicates:

issues.append(f"🔁 Multiple bubbles detected in: {duplicates}")

return issues

**🪄 Poetic Feedback**

python

Copy

def poetic\_feedback(issues):

if not issues:

return "✨ All bubbles aligned. The sheet sings in clarity."

st.markdown("> \_“Some bubbles wandered, some stayed shy—\nLet’s guide them gently, before they fly.”\_")

for issue in issues:

st.warning(issue)

**🔗 Integration in streamlit\_app.py**

python

Copy

from validator\_module import validate\_sheet, poetic\_feedback

from admin\_panel import launch\_admin\_panel

# After extracting responses

issues = validate\_sheet(responses["responses"])

poetic\_feedback(issues)

# After processing all students

launch\_admin\_panel(results)

**🌈 Final Touches**

* Add school filters: Govt / Aided / Pvt
* Gender filters: Boys / Girls
* Subject-wise breakdown
* Export by class or section
* Add constellation badge map for top scorers (optional)

Would you like me to help scaffold the **badge map exporter**, or design a poetic certificate generator next? We could also bundle everything into a streamlit\_dashboard\_launcher.py for classroom showcases.

Edit in a page

**You said**

Create generate\_directoryStructure.ps1 script

**Copilot said**

Here’s a PowerShell script called generate\_directoryStructure.ps1 that creates the full modular directory structure for your OMR Result Generator app, including folders, placeholder files, and a poetic README starter:

powershell

Copy

# generate\_directoryStructure.ps1

# Creates modular folders and starter files for the OMR Result Generator app

$root = "omr\_result\_app"

$folders = @(

"$root",

"$root/utils",

"$root/assets/sample\_sheets",

"$root/data",

"$root/.streamlit"

)

$files = @{

"$root/streamlit\_app.py" = "# Main Streamlit launcher"

"$root/omr\_parser.py" = "# Extracts responses and metadata from scanned sheets"

"$root/answer\_key\_loader.py" = "# Loads and flattens answer key JSON"

"$root/result\_evaluator.py" = "# Compares student responses and scores"

"$root/excel\_exporter.py" = "# Exports results to Excel"

"$root/utils/image\_utils.py" = "# Image preprocessing and bubble detection"

"$root/utils/ocr\_utils.py" = "# OCR logic for metadata extraction"

"$root/admin\_panel.py" = "# School-wide dashboard and export"

"$root/validator\_module.py" = "# Flags incomplete or ambiguous sheets"

"$root/requirements.txt" = "streamlit`nopencv-python`npytesseract`npandas`nopenpyxl"

"$root/.streamlit/config.toml" = "[server]`nheadless = true"

"$root/README.md" = @"

# 🧠 OMR Result Generator

> \_“From ink to insight, from bubbles to brilliance—

> Your answers tell a story, and we help it shine.”\_

Modular Streamlit app to process scanned answer sheets, evaluate student performance, and export class-wise results.

"@

}

# Create folders

foreach ($folder in $folders) {

if (-not (Test-Path $folder)) {

New-Item -ItemType Directory -Path $folder | Out-Null

}

}

# Create files with content

foreach ($path in $files.Keys) {

if (-not (Test-Path $path)) {

$content = $files[$path]

$content | Out-File -FilePath $path -Encoding UTF8

}

}

Write-Host "✅ Directory structure created successfully."

**🧪 How to Run**

1. Save the script as generate\_directoryStructure.ps1
2. Open PowerShell and navigate to the folder where it’s saved
3. Run: .\generate\_directoryStructure.ps1

Would you like me to scaffold the streamlit\_badge\_recommender.py or remix\_theme\_badge\_gallery\_exporter.py next? We could also add a poetic anthology\_index.md to celebrate student creativity.

Edit in a page

-------------x-------------

🎓 Student: | Roll No: ✨ Some bubbles wandered, some stayed shy — Let’s guide them gently, before they fly. Missing responses: 108 “Some bubbles wandered, some stayed shy— Let’s guide them gently, before they fly.” 🕳️ Missing responses: 90 questions unanswered. ✅ Processed 1 students Error is persistent.

Jagdev, thank you for your persistence—this error is poetic but persistent, and now we’ll resolve it with precision.

**🔍 Root Cause Recap**

You're seeing:

* **“Missing responses: 108”** → from expected - actual, where actual = len(responses) is likely 0
* **“90 questions unanswered”** → from draw\_validation\_overlay() using summary['missing'], which is calculated as expected - len(valid\_bubbles)

This means:

* Your extract\_responses(img) is returning an empty responses dictionary
* Your bubble detection found 18 valid bubbles (so 108 - 18 = 90 unanswered)

**✅ What This Tells Us**

* ✅ Bubble detection is working (18 bubbles found)
* ❌ Bubble-to-response mapping is failing (no bubbles matched the grid)

This points to a likely issue in map\_bubbles\_to\_responses() inside image\_utils.py.

**🔧 Fix Strategy**

**1. Check question\_grid Format**

Ensure question\_grid passed to map\_bubbles\_to\_responses() contains coordinates that match the actual bubble positions. If the grid is misaligned, no bubbles will map.

**2. Add Debug Logs in extract\_responses()**

Inside omr\_parser.py:

python

def extract\_responses(img):

img, thresh = preprocess\_image(img)

bubbles = detect\_bubbles(thresh)

print(f"🧪 Detected {len(bubbles)} bubbles")

for b in bubbles[:5]: print("Bubble:", b)

question\_grid = load\_question\_grid() # or however you're loading it

responses = map\_bubbles\_to\_responses(bubbles, question\_grid)

print(f"📝 Mapped {len(responses)} responses")

return {"responses": responses, "bubbles": bubbles}

This will confirm whether bubbles are detected and whether they’re mapping.

**3. Check radius\_tolerance in Mapping**

In map\_bubbles\_to\_responses():

python

def map\_bubbles\_to\_responses(bubbles, question\_grid, radius\_tolerance=15):

Try increasing radius\_tolerance to 25 or 30 temporarily to allow looser matching.

**4. Visual Debug Overlay**

Overlay both:

* Detected bubbles (green/red)
* Grid centers (blue)

This will help you visually confirm whether bubbles are near expected grid points.

**✅ Next Steps**

Would you like me to help you scaffold a grid\_overlay\_debugger.py that overlays grid points and detected bubbles together for visual alignment? Or help you validate your question\_grid.yaml to ensure coordinates match the scanned layout?