Presentation

DEWY-THE AI SKIN ASSISTANT

ROLES

Sruthi -Frontend (Quiz UI & Image Upload)
Siddartha- Frontend (Dashboard, Output Display)
Varshit- CNN Model for Image Classification
Haryashw -CNN Integration & Prediction API
Rithwika- Quiz Model (Random Forest)
Abhigna -Training Quiz Model, Mapping Logic
Charitha Reddy -Flask Backend ,API Integration & ML models
Antara- Dataset Curation & JSON Recommendations

PROBLEM STATEMENT

- Around 4 in every 5 people suffers from a skin condition, yet access to timely dermatological care is limited due to high costs, specialist shortages, and location barriers. Many rely on self-diagnosis or delayed treatment, making conditions worse. Current tools focus only on images and ignore personal factors like age, skin type, or allergies. There’s a clear need for a smart, affordable solution that provides early, personalized skin health support.

OUR SOLUTION!!(Objectives)

DEWY-THE AI SKIN ASSISTANT

To address this, we propose DEWY – The AI Skin Assistant, a web-based application that:

- Allows users to upload an image of their affected skin area.
- Takes quiz input (age, skin type, allergies, activity, etc.) for deeper context.
- Provides condition-specific recommendations:
  - Treatments and medications
  - Skincare product suggestions
  - Dietary and lifestyle advice
- Supports ingredient analysis of skincare products for safety and effectiveness.
- Generates a complete, easy-to-understand report for the user.

Software Engineering Approach!?

WATERFALL MODEL😊

- Requirement Analysis – Defined problem and user needs.
- System Design – Designed architecture and selected tools.
- Implementation – Developed using Python (Flask), HTML/CSS/JS.
- Testing – Manual testing of models, form inputs, and API.
- Deployment (Local) – Runs on localhost with test datasets.
- Documentation – Prepared SRS, SDD, and final reports.

Requirements Gathering

Functional:

- Upload image for disease prediction.
- Submit quiz for personalized recommendation.
- Show product, treatment, and diet suggestions.

Non-functional:

- User-friendly UI.
- Fast response time.
- Lightweight local Hosting

Project Architecture

Architecture Pattern Used: MVC (Model–View–Controller)

Model:

- Contains the machine learning models:
- CNN model for analyzing skin images
- SLM (Small Language Model) for processing quiz/text inputs
- Recommendation logic for treatment, products, diet, etc.

View:

- The user interface developed using HTML, CSS, and JavaScript:
- Home page, quiz form, image upload form
- Result display and skincare analysis output

Controller:

- The backend logic using Flask (Python):
- Handles API requests
- Connects user inputs from the View to the Models
- Sends predictions and recommendations back to the frontend

CNN Model for Image Prediction

- CNN (Convolutional Neural Network) is used to detect skin diseases from uploaded images.
- It has multiple hidden layers:
- Convolutional layers extract features (like skin texture, bumps, color).
- Pooling layers reduce image size and focus on key features.
- ReLU activation adds non-linearity, improving learning.
- Fully Connected Layers make final predictions after feature extraction.
- Softmax output gives probabilities for diseases (e.g., acne, eczema, etc.).
- Model trained on labeled image dataset using cross-entropy loss and Adam optimizer and used accuracy metrics.
- Final model predicts disease from the image and is integrated with the backend.

SLM Model for Quiz Analysis

- SLM is used to predict skin diseases from user quiz responses (age, allergies, duration, activity level, etc.).
- The model follows these steps:
- Data Preprocessing: Text answers are encoded into numerical format using Label Encoding for both features and target variable (disease).
- Model Training: A Random Forest Classifier is trained using the processed data to map quiz answers to skin disease predictions.
- Evaluation: The data is split into training and testing sets to evaluate the model's performance.
- Model Storage: The trained model, along with label encoders for feature and target encoding, is saved using pickle.
- The final model is integrated with the backend and predicts diseases based on quiz inputs when an image is unavailable or unclear.

Methodology and Implementation

Data Collection

- Image Dataset: Collected labeled images for five skin conditions: acne, herpes, panu, rosacea, and eczema.
- Quiz Dataset: Gathered data from a comprehensive quiz covering user age, allergies, skin issue duration, physical activity, etc. and used PYTHON SCRIPT to generate CSV

Data Preprocessing

- Image Preprocessing: Images are resized and normalized for CNN-based analysis.
- Quiz Data: Textual quiz responses are encoded using Label Encoding for use in the SLM.

Model Development

- CNN (Image Model): Trained on image data to detect skin diseases based on visual features.
- SLM (Text Model): Trained on quiz data to predict diseases based on user responses like age, allergies, and lifestyle.

Methodology and Implementation

Integration

- Both models (CNN and SLM) are integrated into a unified system that can predict skin diseases using either images, quiz inputs, or both.

Backend Development

- Built a backend using Flask to handle user inputs, process image data, and provide disease predictions via both models.

Deployment

- The system is deployed with an intuitive frontend for users to upload images and fill out quizzes, with the backend providing real-time disease predictions and treatment recommendations.

FLASK BACKEND AND API🤯

- Built using Flask, a Python-based micro web framework.
- Handles routes such as /, /predict-text, /predict-image.
- Integrates AI models for text and image prediction.
- JSON-based API communication between frontend and backend.
- Handles form submissions and sends responses dynamically.
- No external database used; login/signup forms are static.
- app.py acts as the central controller in MVC architecture.

Achievements 💃

- Successfully integrated CNN and SLM models into a working web app.
- Designed an intuitive UI with image upload and text-based quiz.
- Generated disease predictions with treatment/product suggestions.
- Created a working backend API structure using Flask.
- Offered a multi-modal approach (image + text) to improve diagnosis accuracy.

Future Plans 🤔

- Add real-time camera capture using HTML5 or MediaPipe.
- Implement Firebase/Supabase database for storing user accounts and results.
- Include user progress tracking and reports.
- Host on Render/Heroku or other cloud platforms for live demo.
- Build mobile responsiveness and accessibility.
- Enable voice-based interaction and multi-language support.

Problems Faced😭

- ⚠ GitHub file limit: .pkl and .keras model files >25MB couldn’t be pushed.
- Solved by sharing via external link.
- ⚠ Login/signup forms are static due to no backend database.
- ⚠ File upload only — no real-time camera input for images.
- ⚠ Integration complexity of combining image + text predictions.
- ⚠ Initial bugs in Flask routes and data parsing.
- ⚠ Model accuracy tuning took multiple iterations and preprocessing.

Struggles Behind the Scenes🤕

We considered implementing user authentication using Firebase, but faced technical limitations. Our HTML frontend uses multi-page rendering with Flask, which doesn't fully support Firebase's async authentication flow. Also, OTP systems require domain verification, and in development environments, Firebase's phone authentication fails without proper setup. To securely handle login sessions, we would need JWT tokens or Flask session handling, which conflicted with our existing ML model routing. So to avoid introducing unstable behavior, we chose to leave authentication as a frontend design component for now.

We didn't use a live camera feature due to technical and security reasons. getUserMedia() requires HTTPS and doesn't work well in development setups. Also, the integration with Flask needed custom logic for handling captured image blobs, and ensuring it worked across devices would require extra libraries. Since our image model was already working with uploaded files, we chose the simpler and safer method of file upload to keep the workflow stable.

Thank You