Project Title: AI Retina Guard (Next-Gen Ophthalmic Diagnostics)

1. Project Overview

AI Retina Guard is a full-stack medical diagnostic application designed to detect retinal diseases from fundus scan images. It utilizes a custom-built **Convolutional Neural Network (CNN)** for image classification and integrates **Google's Gemini 1.5 Flash LLM** to act as a context-aware medical assistant.

2. Deep Learning Model Architecture (The "Hidden" Core)

The core intelligence is a custom PyTorch CNN model designed specifically for feature extraction from retinal images.

- **Model Type:** Custom Convolutional Neural Network (CNN)
- **Input Resolution:** 256 x 256 pixels (RGB channels)
- **Total Learnable Layers:** 5 Layers (3 Convolutional Blocks + 2 Fully Connected Layers)
- **Output Classes (4):** Cataract, Diabetic Retinopathy, Glaucoma, Normal.

Detailed Architecture Breakdown:

1. **Feature Extraction (Convolutional Blocks):**
    - **Layer 1:** `Conv2d` (3 → 16 filters, Kernel: 3, Stride: 2) + `BatchNorm` + `LeakyReLU` + `MaxPool` (2x2).
    - **Layer 2:** `Conv2d` (16 → 32 filters, Kernel: 3, Stride: 2) + `BatchNorm` + `LeakyReLU` + `MaxPool` (2x2).
    - **Layer 3:** `Conv2d` (32 → 64 filters, Kernel: 3, Stride: 2) + `BatchNorm` + `LeakyReLU` + `MaxPool` (2x2).
    - *Technical Note:* The use of `Stride=2` in convolutions *plus* MaxPool results in aggressive downsampling to capture high-level features while reducing computational load.
2. **Classifier (Dense Layers):**
    - **Flattening:** The 3D feature maps are flattened into a 1D vector (Input size: 64 * 3 * 3 = 576 neurons).
    - **Hidden Layer:** `Linear` (576 → 128 neurons) + `ReLU` Activation.
    - **Output Layer:** `Linear` (128 → 4 neurons).
3. **Regularization & Optimization:**
    - **Batch Normalization:** Applied after every convolution to stabilize learning.
    - **Dropout (0.2):** Applied before dense layers to prevent overfitting (randomly disabling 20% of neurons during training).
    - **Activation:** Uses `LeakyReLU` for feature extraction (prevents "dead neurons") and standard `ReLU` for the classifier.

3. Backend Workflow (Python/Flask)

The backend acts as the bridge between the user interface, the CNN model, and the Gemini API.

- **Image Preprocessing:**
  - Incoming images are converted to RGB (stripping alpha channels).
  - Transformed into PyTorch Tensors.
  - Resized strictly to **256x256** to match the neural network's input requirement.
- **Inference Logic:**
  - The model runs in `eval()` mode on the available device (CUDA/GPU or CPU).
  - **Softmax Function:** The raw model outputs (logits) are passed through a Softmax function to convert them into probabilities (0% to 100%).
  - **Confidence Calculation:** The highest probability is extracted and returned as the "Confidence Score."
- **Context-Aware Chatbot:**
  - The system uses **Prompt Engineering**. It doesn't just send your message to Google; it wraps your message in a system prompt: *"You are an expert Ophthalmologist... The user is currently viewing a result for: [Current Disease Context]"*.
  - This ensures the AI gives advice relevant to the specific disease detected in the image.

4. Frontend Technology (The Interface)

The interface is built with a focus on modern UI/UX principles, specifically **Glassmorphism**.

- **Tech Stack:** HTML5, CSS3, Vanilla JavaScript (No heavy frameworks like React/Angular).
- **Visual Engine:**
  - **CSS Variables (`:root`)**: Handles dynamic theming (Light/Dark mode).
  - **Glassmorphism**: Uses `backdrop-filter: blur()` and semi-transparent backgrounds (`rgba`) to create a frosted glass effect.
  - **Animations**: CSS Keyframes (`@keyframes`) are used for the floating background orbs and the fade-in entry effects.
- **State Management:**
  - JavaScript variables track the `currentContext` (the detected disease), allowing the chatbot to "remember" what the user is looking at without requiring a database.

5. Deployment Specifications

- **Containerization:** Designed to run via Docker (as seen in previous code fragments).
- **Port:** Default port `7860` (Standard for Hugging Face Spaces), falling back to 5000 for local dev.
- **Environment Security:** Uses `python-dotenv` to load the Google API Key securely, preventing it from being hardcoded in the source.