**1. What We Have Now**

- Our current system can detect brain tumors from MRI scans using AI (ResNet18 model).
- Users upload images on a React web app; the Flask backend predicts the tumor type, gives a basic explanation (caption), and shows a Grad-CAM heatmap for visual proof.

---

**2. What We're Adding Next**

**A. Fracture Detection**

- We will train a YOLO model to detect bone fractures in X-ray images.
- The system will work similarly: user uploads image, backend detects fracture, and displays results.

**B. Smarter Captions with GenAI**

- Instead of fixed captions, we'll use advanced AI (LLMs) to write clear, personalized medical explanations for both tumor and fracture results.

**C. Severity Assessment**

- The AI will also rate how serious the tumor or fracture is, and give simple follow-up advice.

**D. RAG-based Report Summaries & Chat**

- Users can upload medical reports (PDFs); our system will summarize them and answer questions by finding info in those PDFs.

**E. Chatbot with Image Memory**

- Users can chat with an AI assistant about their images and reports.
- The bot remembers previous uploads and can compare old and new results.

**F. Proactive Clinical Assistant (Agentic AI)**

- The assistant will not just answer but also suggest next steps:
  - For example, if a high-risk tumor is found, it will offer to show clinical guidelines or help schedule a doctor's visit.
  - If no fracture is found, but the user has pain, it may suggest uploading a follow-up image later.

### 1. Fracture Detection via YOLO

- **Technical Detail**:
  - We will train a YOLO model (You Only Look Once) on a fracture X-ray dataset.
  - The Flask backend will load this new model file (e.g. yolo_fracture.pth) and add a new API endpoint (like /fracture/predict).
  - The frontend will let the user upload an X-ray, send it to the backend, and display bounding boxes showing detected fractures.

---

### 2. LLM-Based Captioning for MRI & Fracture

- **Technical Detail**:
  - Instead of showing fixed captions, we'll use a Large Language Model (LLM) like GPT or Llama.
  - This model will take results (tumor/fracture type, image region) and generate a custom, easy-to-understand medical report for each scan.
  - The backend will add an endpoint (e.g. /caption/generate) that sends data to the LLM and returns the generated text.

---

### 3. Severity Assessment with LLM

- **Technical Detail**:
  - After detection, the AI will rate how serious the tumor or fracture is.
  - The backend will use LLM prompts based on model results to give a severity score ("low", "moderate", "high") and explain why.
  - The frontend will show this score alongside the results.

---

### 4. RAG-Based Clinical Report Summaries & Chat

- **Technical Detail**:
  - Users can upload PDFs of medical reports.
  - The backend will use Retrieval-Augmented Generation (RAG): it finds relevant info from the report using embeddings and returns LLM-generated summaries or answers.

    - A new API (e.g. /rag/query) will handle these requests, and the frontend will show answers with document citations.

---

**5. Chatbot with Image Memory**

- **Technical Detail**:
    - Users chat with an AI assistant about their scans.
    - The backend keeps session memory: it can recall previous images and results, so users can ask things like “compare this to my last scan.”
    - The frontend will have a chat interface that connects to an endpoint like /chat/message.

---

**6. Proactive Agentic Clinical Assistant**

- **Technical Detail**:
    - The AI agent monitors results and user profile.
    - If a high-risk result is found, it can automatically suggest next steps (like “view guidelines,” “schedule appointment,” or “upload old scans for comparison”).
    - This logic runs in the backend and triggers prompts in the frontend for user action—always requiring user consent before taking any step.

---

**7. Multi-Model Backend Structure**

- **Technical Detail**:
    - Each model (tumor, fracture, captions) will have its own Python file and Flask endpoint.
    - For example, model_mri.pth for tumor, yolo_fracture.pth for fracture.
    - The backend will load these models separately and route user requests to the right endpoint, making it easy to add new analysis types in the future.

**1. Frontend (React)**

**User Experience**

- **Login & Profile:**
    - User signs in (see Login.tsx), profile info managed (see AuthContext.tsx and Profile.tsx).
    - Can edit details, upload profile picture, and store medical history for personalized analysis.
- **Image Upload & Analysis:**
    - User uploads an MRI or X-ray on ImageUpload.tsx.
    - The page shows image preview, processing animation, and then results (prediction, explanation, Grad-CAM image).
- **Result Display:**
    - Displays predicted label (tumor/fracture type), AI-generated explanation, and visual Grad-CAM (for MRI) or bounding boxes (for fractures).
- **Interactive Chat & RAG Features:**
    - User can chat about their scan or upload medical PDFs for instant summaries and Q&A.

---

**2. Backend (Flask)**

**Core API Structure**

- **Modular Endpoints:**
    - /predict (MRI): Uses ResNet18, returns tumor label, caption, Grad-CAM.
    - /fracture/predict: Will use YOLO, return fracture type and bounding boxes.
    - /caption/generate: Uses LLM to make a readable report/caption based on findings.
    - /severity/score: LLM rates seriousness (low/high) and gives advice.
    - /rag/query: For document-based answers using clinical PDFs.
    - /chat/message: Smart assistant, keeps chat session and responds based on current and previous scans.

    - /agent/suggest: Agentic AI logic, proactively suggests actions based on findings.

**Model Handling**

- **Multiple model files:**
    - Each task (MRI, fracture) has its own .pth model loaded once at server start.
    - Model registry or config file points Flask to correct weights for each endpoint.
    - Easy to add new models: just update registry/config and add a new endpoint.

---

**3. Medical AI Models**

**A. MRI Tumor Detection**

- **Model:** ResNet18, fine-tuned for 4 classes (glioma, meningioma, pituitary, notumor).
- **Explainability:** Grad-CAM highlights suspicious regions, shown to user.

**B. Fracture Detection**

- **Model:** YOLOv8 (or latest), trained on bone fracture X-rays.
- **Output:** Bounding boxes on the image, fracture type, confidence score.

**C. Future Models**

- **KAN, other CNNs:** For experiments, or to improve accuracy.

---

**4. GenAI Modules**

**A. Caption Generation (LLM)**

- **LLM Model:** GPT-4o, Llama2, or similar (hosted via API or local inference).
- **Input:** Structured findings (tumor/fracture type, region, severity, patient history).
- **Output:**
    - Patient-friendly text: “Your MRI shows a small glioma in the left temporal lobe. No signs of spread.”
    - Clinician summary: “Glioma, left temporal, 2.3cm, no edema.”

- **Integration:** Flask endpoint /caption/generate calls the LLM, returns result to frontend.

**B. Severity Assessment**

- **LLM uses rules:**
  - Reads size, location, displacement, confidence, patient history.
  - Outputs severity (e.g., “High risk due to location near motor cortex”) and advice.
- **Endpoint:** /severity/score

**C. RAG (Retrieval-Augmented Generation)**

- **PDF Upload:** User uploads clinical reports/guidelines.
- **How it works:**
  - Backend splits PDF into chunks, creates embeddings, stores in FAISS (vector DB).
  - When user asks a question, backend finds relevant chunks and LLM answers only using info from those chunks.
  - Citations: Each answer links to the exact PDF page/section.
- **Endpoints:** /rag/ingest, /rag/query

**D. Chatbot with Memory**

- **LLM-powered conversation:**
  - Backend stores session data (previous images, findings, chat history).
  - When user chats, LLM gets current findings and session history for smarter replies.
- **Example:** “Last scan showed a meningioma. This one looks normal. Want to compare images?”
- **Endpoint:** /chat/message

**E. Agentic Clinical Assistant**

- **Logic:**
  - Monitors predictions and user profile.
  - Proactively suggests actions (not just answers questions).
- **Examples:**

- “High-grade glioma detected. Would you like to view treatment guidelines or schedule a specialist visit?”
- “No fracture found, but your profile says ongoing pain. Should we set a reminder to check again in two weeks?”

- **Endpoint:** /agent/suggest
- **Opt-in:** User must accept before actions (privacy-first).

---

**5. Backend Tech Stack**

- **Flask:** Serves all AI endpoints, handles API requests from frontend.
- **PyTorch:** Loads and runs ResNet/YOLO models for predictions.
- **FAISS:** Fast vector search for RAG.
- **Session store:** SQLite or Redis for chat and user history.
- **Joblib:** Loads label encoders and other pickled utilities.

---

**6. Frontend Tech Stack**

- **ReactJS:** SPA for all pages.
- **Tailwind CSS:** For beautiful, responsive design.
- **React Dropzone:** For image upload UI.
- **LocalStorage:** For storing user/session data.
- **REST API calls:** To Flask endpoints for predictions, captions, chat, etc.

---

**7. How All Features Work Together (Flow)**

1. **User logs in, completes profile.**
2. **Uploads an MRI or fracture X-ray.**
3. **Frontend sends image to backend.**
4. **Backend runs the correct model (ResNet18 or YOLO), gets prediction.**
5. **Grad-CAM/boxes created for visual explanation.**
6. **Prediction sent to LLM for easy-to-read caption/report.**
7. **LLM also scores severity and gives advice.**

8. **User can chat about findings, compare with old scans, or upload PDFs for instant Q&A.**

9. **Agentic assistant monitors, and if something important is found, proactively suggests next steps (with user consent).**

10. **All results, chats, and history stored for future reference.**

---

### 8. Adding New Models (Technical Plan)

- Put each new model (e.g., model_fracture.pth) in a models/ folder.
- Register it in a config file, mapping task → model file.
- Add a new Flask endpoint (e.g., /fracture/predict) that loads and uses only that model.
- Frontend adds a new upload mode (MRI or fracture), and calls the right endpoint based on image type.
- Results handled in the same way—modular, easy to scale.

---

### 9. Deployment & Scaling

- **Backend:** Runs on cloud VM or college server, with CUDA for speed if available.
- **Frontend:** Hosted on Vercel/Netlify or college web server.
- **Security:** CORS enabled, authentication required, user data protected.
- **Models:** Loaded once at startup, fast inference (CPU or GPU).
- **RAG:** PDF data stored locally; embeddings indexed for instant Q&A.

---

### 10. Why This Stack?

- **Modular:** Easy to add new models and features.
- **Scalable:** Can handle more tasks (other diseases, more imaging types).
- **User-friendly:** Clear reports, chat, and proactive help.
- **AI-powered:** Combines classic deep learning (for detection) with cutting-edge GenAI (for captions, chat, RAG, agentic logic).

---

### 11. Summary Table (Feature → Tech Details)

| Feature | Tech Details |
|---|---|
| MRI Tumor Detection | ResNet18, Grad-CAM, Flask /predict endpoint |
| Fracture Detection | YOLOv8, Flask /fracture/predict endpoint |
| Caption Generation | LLM (GPT-4o/Llama2), Flask /caption/generate |
| Severity Assessment | LLM + prompt, Flask /severity/score |
| RAG-based Q&A | PDF chunking, SBERT/MiniLM embeddings, FAISS, LLM, Flask /rag/query |
| Chatbot | LLM, session/context store, Flask /chat/message |
| Agentic Assistant | Trigger logic + LLM, Flask /agent/suggest |
| Frontend | ReactJS, Tailwind, REST calls, pages: upload, results, chat, profile |
| Adding Models | Place new .pth file, update config, add endpoint, update frontend mode selector |