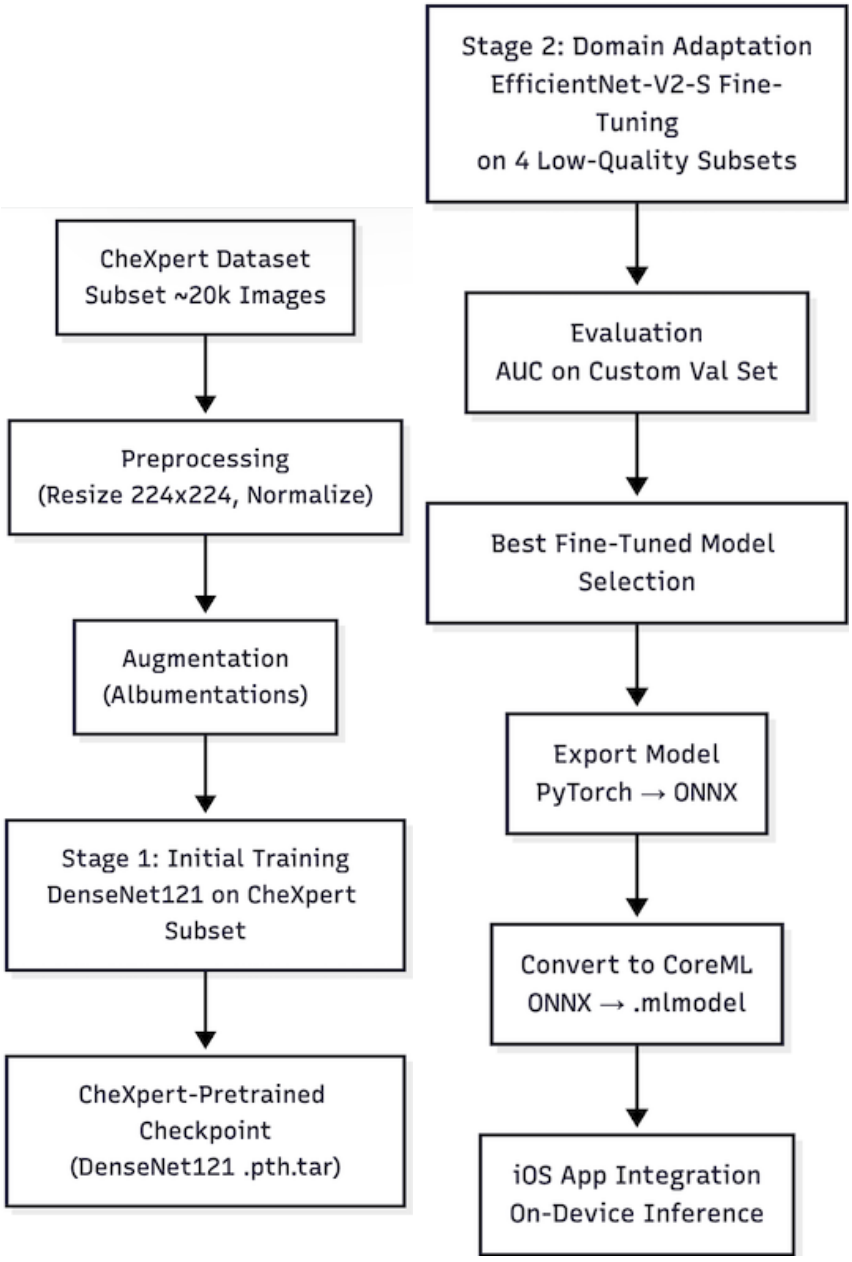
Title: Chest X-Ray Diagnosis App

Class Name: CSE 486 Computer Sci Capstone Project II
Team Name: Team Mayo
Team Members: Connor McMahon, Sanskar Srivastava
Sponsor Name: Dr. Imon Banerjee

Technical Documentation

1. Architecture Diagrams of the System

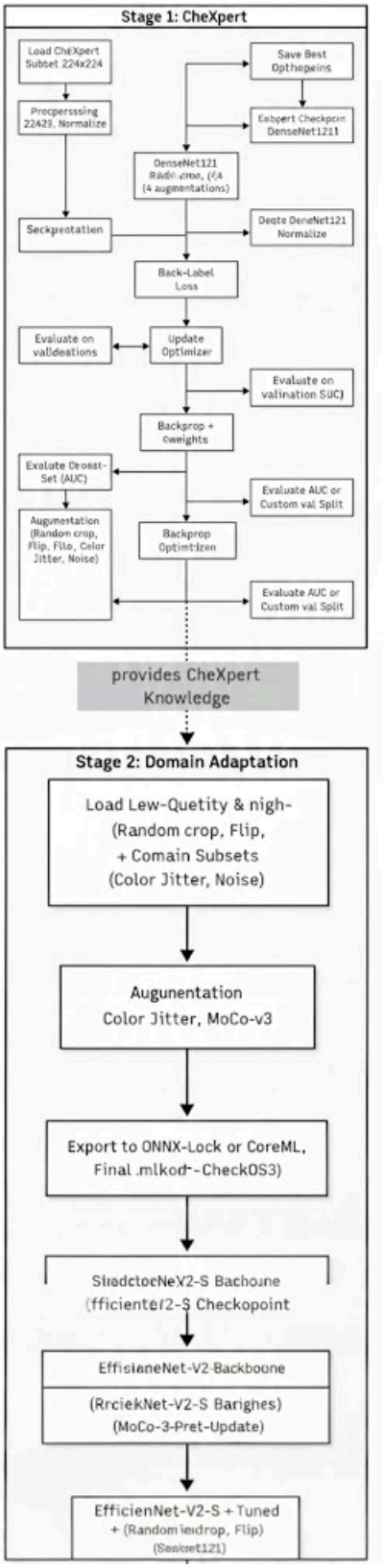
1.1 High-Level System Pipeline (Flowchart – Two-Stage Training)



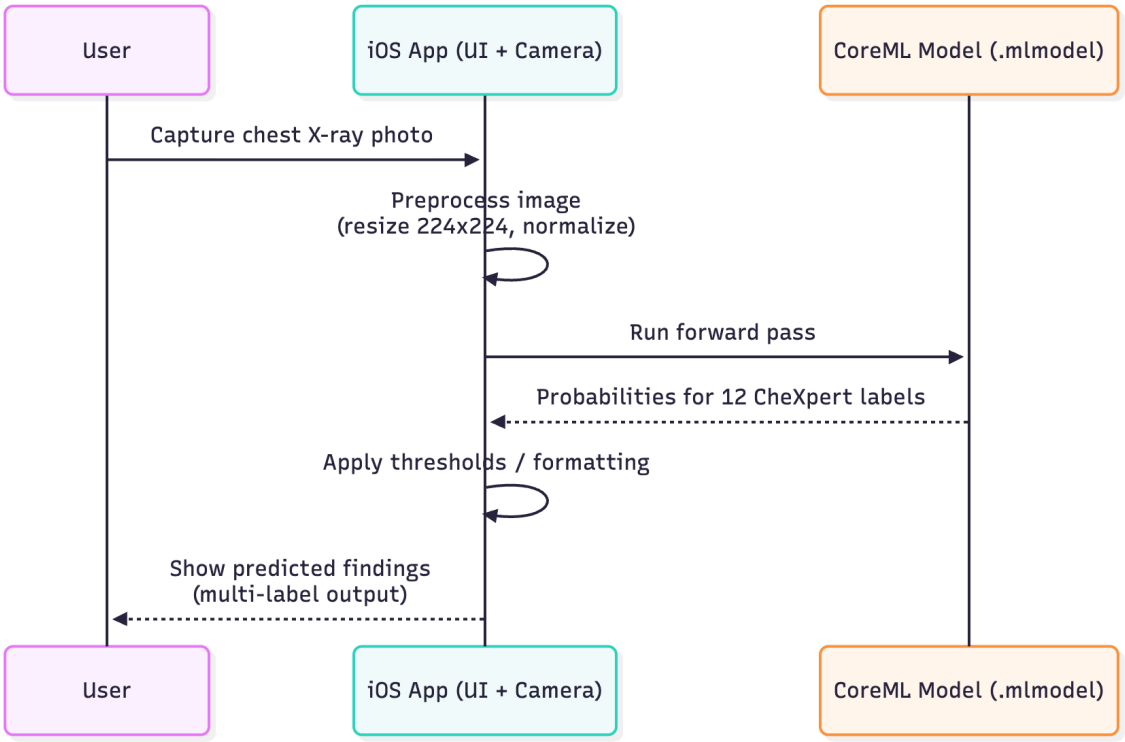
Stage 1: Train DenseNet121 on CheXpert subset → CheXpert-aware features.

Stage 2: Use a separate EfficientNet-V2-S model as the final backbone and fine-tune it for low-quality, phone-like images using the domain-adaptation notebook.

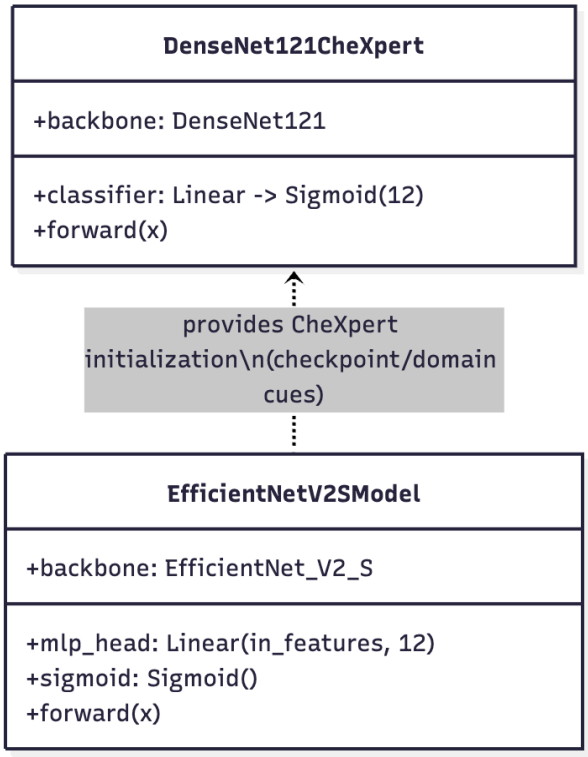
1.2 Training & Fine-Tuning Pipeline (Flowchart – Detailed)



1.3 On-Device Inference Flow (Sequence Diagram)



1.4 Model-Level Architecture (Class Diagram – Final Model: EffNetV2-S)



2. File Navigation Information (Corrected)

2.1 Dataset Directory Structure

Images are organized per label:

Atelectasis/
Cardiomegaly/
Consolidation/
Edema/
Enlarged_Cardiomediastinum/
Fracture/
Lung_Lesion/
Lung_Opacity/
No_Finding/
Pleural_Effusion/
Pleural_Other/
Pneumonia/
Pneumothorax/
Support_Devices/

This structure is assumed in your notebooks for loading the CheXpert subset.

2.2 Code Entry Points (Corrected)

- cheXpert_final.ipynb – Stage 1: CheXpert Pretraining
    - Defines DenseNet121:
        - Uses torchvision.models.densenet121(pretrained=True).
        - Replaces the classifier with nn.Linear(..., out_size) + Sigmoid() for 14 labels.
    - Trains on the ~20k CheXpert subset.
    - Saves a CheXpert-pretrained DenseNet121 checkpoint (.pth/.tar).
- finetuning.ipynb – Stage 2: Domain Adaptation (EfficientNet-V2-S)
    - Imports efficientnet_v2_s and EfficientNet_V2_S_Weights from torchvision.models.
    - Builds an EfficientNet-V2-S + MLP head model.

- Loads configuration like:
  - TRAIN_CSV, TEST_A_CSV, TEST_B_CSV
  - TRAIN_CROP_SIZE, VAL_RESIZE, VAL_CROP_SIZE
  - BATCH_SIZE = 12
- Uses an upstream checkpoint path (CheXpert-pretrained model) as an additional initialization cue.
- Fine-tunes on 4 domain subsets (white/yellow, closer/further).

2.3 Where a New Developer Should Start

1. Read cheXpert_final.ipynb
   - Understand how the CheXpert subset is loaded and how DenseNet121 is trained to become CheXpert-pretrained.
2. Read finetuning.ipynb
   - See how EfficientNet-V2-S is constructed and fine-tuned on low-quality subsets.
   - Check how the pretrained checkpoint is referenced.
3. Follow the export section in finetuning.ipynb to see ONNX → CoreML conversion.

3. Installation Process

3.1 Environment

- Environment manager: Conda
- Python: 3.11
- Hardware: GPU with CUDA (training done on SOL supercomputer, ASU)

Example:

conda create -n chexpert-env python=3.11

```
conda activate chexpert-env
pip install -r requirements.txt
```

Dependencies are specified in the requirements.txt file (available on GitHub).

3.2 Key Dependencies (From requirements.txt)

Core stack (subset):

- **Machine Learning**
    - torch==2.4.1
    - torchvision==0.19.1
    - albumentations==2.0.8
    - opencv-python, opencv-python-headless requirements
- **Conversion / Deployment**
    - onnxruntime-gpu
    - coremltools==9.0 requirements
- **Data / Analytics**
    - numpy, pandas, scikit-learn, matplotlib, seaborn

3.3 Hardware Requirements

- NVIDIA GPU with CUDA that matches the PyTorch build in requirements.txt.
- Enough GPU memory for:
    - Batch size: 12
    - Training crop: 320 (for finetuning)
- Training environment: SOL supercomputer at ASU.

4. Licenses

4.1 Project License

- No explicit license file → **N/A**

4.2 CheXpert Dataset License

- CheXpert is released by Stanford for personal, non-commercial research under dataset-specific terms.

5. Libraries (Source + Version)

5.1 Core Training & Inference

- **PyTorch ecosystem**
    - torch==2.4.1
    - torchvision==0.19.1
    - torchaudio==2.4.1 (not central to this project)
- **Model architectures**
    - **Stage 1:** torchvision.models.densenet121
    - **Stage 2:** torchvision.models.efficientnet_v2_s, EfficientNet_V2_S_Weights
- **Augmentations / Preprocessing**
    - albumentations==2.0.8
    - opencv-python, opencv-python-headless
- **Conversion / Deployment**
    - onnxruntime-gpu
    - coremltools==9.0
- **Data / Metrics / Visualization**

- numpy, pandas, scikit-learn, matplotlib, seaborn

6. API / SDK Details

- No external API: **on-device inference only**.
- **Runtime:** CoreML model embedded in an iOS app.

6.1 Model Interface

- **Input:**
  - Single image tensor shaped like (1, 3, 224, 224) after preprocessing which would be normalized
- **Output:**
  - Probability vector (0–1) for the CheXpert labels.
  - Thresholding, formatting, and display handled in the iOS UI layer.
  - GradCam Generated with overlay on image

7. Model Training Details

7.1 Dataset & Subsets

- **Base dataset:** CheXpert (large chest X-ray dataset). Dataset
- **Working subset:** ~20k high-quality images with 14 subclasses.
- **Domain subsets** (for simulating phone captures):
  - white_furtherout
  - white_closerin
  - yellow_furtherout
  - yellow_closerin

7.2 Preprocessing

- Resize images to **224×224**.
- Convert to float and normalize:
    - A.ToFloat(max_value=255.0)
    - A.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])

7.3 Data Augmentation

Training pipeline (Albumentations):

```
A.RandomResizedCrop(size=(TRAIN_CROP_SIZE, TRAIN_CROP_SIZE),
                    scale=(0.8, 1.0)),
A.HorizontalFlip(p=0.5),
A.RandomBrightnessContrast(
   brightness_limit=(0.08, 0.18),
   contrast_limit=(0.02, 0.08),
   p=0.3),
A.ColorJitter(brightness=0.05,
              contrast=0.05,
              saturation=0.05,
              hue=0.02,
              p=0.3),
A.GaussNoise(var_limit=(5.0, 20.0), mean=0.0, p=0.2),
A.ToFloat(max_value=255.0),
A.Normalize(mean=[0.5]*3, std=[0.5]*3)
```

Purpose:

- Adjusts scale and crop to handle framing differences (far/close radiograph photos).
- Adds flips and photometric noise to mimic phone camera variability.
- Regularizes the model to be robust to white/yellow background shifts.

7.4 Stage 1 – CheXpert Pretraining (DenseNet121)

- **Model:** DenseNet121 with modified classifier:
    - torchvision.models.densenet121(pretrained=True)
    - Last layer replaced by Linear(in_features, 14) + Sigmoid.
- **Objective:** Multi-label chest X-ray classification across 14 findings.
- **Training:** On ~20k curated CheXpert subset.
- **Output:** CheXpert-pretrained DenseNet121 checkpoint, used as the **checkpoint that would be later used for domain shift.**

7.5 Stage 2 – Domain Adaptation (EfficientNet-V2-S)

- **Backbone:** efficientnet_v2_s from torchvision.models.
- **Initialization:**
    - Uses EfficientNet-V2-S weights (ImageNet) plus information from the saved CheXpert checkpoint (as per your notebook/comments).
- **Head:**
    - MLP classifier mapping EfficientNet features → 14 sigmoid outputs.
- **Optimizer:** AdamW.
- **Loss:** Asymmetric Loss (ASL) for imbalanced multi-label data.
- **Epochs:** 40.
- **Batch size:** 12.

- **No freezing:** All layers of EfficientNet-V2-S are fine-tuned.
- **Training data:**
    - Uses both high-quality and low-quality subsets, with specific emphasis on:
        - subset_white_* and subset_yellow_* to simulate phone captures (different lighting/background distances).

7.6 Evaluation

- **Metric:** AUC (Area Under ROC Curve)
    - Likely per-label AUC and/or macro averages.
- **Validation:** Custom validation split derived from CheXpert

7.7 Deployment

- Final fine-tuned **EfficientNet-V2-S + MLP head** is:
    1. Saved as PyTorch weights.
    2. Exported to **ONNX**.
    3. Converted from ONNX to **CoreML .mlmodel** using coremltools.
    4. Embedded into an **iOS application** for **offline, on-device inference** (no API calls).

User Guide

1. Goals and Functionality (EPICS / User Stories)

EPIC 1: Capture High-Quality Chest X-Ray Images

- *I want the app to assist me as a user when I take a photo of a chest X-ray so that the final image is clear and helpful for diagnosis.*
- *As a user, I want the system to automatically assess the quality of the image and alert me if the X-ray is poorly framed or fuzzy.*

EPIC 2: Automated Quality Control

- *As a user, I would like the app to automatically identify the lung region so that the system is aware of where to perform quality checks.*
- *As a user, I want the app to utilize edge detection to assess the clarity and focus of my lung region and notify me when a retake is necessary.*

EPIC 3: On-Device X-Ray Analysis

- *As a user, I want this app to forecast its results without the use of the internet, as some areas may not have a network.*

EPIC 4: Visual Explainability

- *As a user*, I want to see a heatmap that highlights regions of interest for model predictions so I understand why the model made a decision.

EPIC 5: Multi-Label Disease Classification

- *As a user*, I want the results displayed clearly and visually alongside my X-ray.

2. Services and/or Features

2.1 Fully On-Device ML Pipeline

- No internet connection required.
- All models (segmentation, quality assessment, and disease prediction) run locally on the backend.

2.2 Real-Time Guided X-Ray Capture

- Integrated camera interface optimized for photographing radiographs.
- On-screen alignment and positioning suggestions.
- Immediate feedback on whether the image is:
    - Too blurry

- Out of frame
- Incorrectly oriented
- Missing lung regions

2.3 Automatic Image Quality Assessment

The system validates image quality before any diagnosis steps by:

1. **Lung Region Segmentation (Core ML segmentation model)**
    - Detects lung masks directly on-device.
    - Ensures the X-ray is correctly centered and captured.
2. **Sharpness Measurement**
    - Sobel edge detection for gradient-based sharpness analysis.
    - Laplacian variance detection to measure overall focus.
    - Threshold-based pass/fail system:
        - If the below sharpness threshold → user is prompted to retake.
        - If acceptable → continues to predictive analysis.

2.4 Diagnostic Features

2.4.1 Grad-CAM Heatmap Generation

- Uses the results of the model to generate a colored heatmap highlighting the most influential regions for the prediction
- The heatmap is **overlaid directly on the user's captured X-ray**.

2.4.2 Multi-Label Disease Classification

- After quality approval, the model returns predictions for **14 chest pathologies**:
    1. Atelectasis

2. Cardiomegaly
3. Consolidation
4. Edema
5. Enlarged Cardiomediastinum
6. Fracture
7. Lung Lesion
8. Lung Opacity
9. Pleural Effusion
10. Pleural Other
11. Pneumonia
12. Pneumothorax
13. Support Devices
14. No Finding

Each condition is displayed with:

- Probability score
- Grad-CAM visualization for relevant findings

2.5 Secure Offline Workflow

- Images remain on the user's device.
- No cloud transmission.
- No API calls or internet dependency.

3. Application Description

The application is a fully on-device chest X-ray analysis tool that combines computer vision, mobile machine learning, and an intuitive UI to help users capture and interpret radiographic images.

When the user takes a photo:

1. Lung segmentation is performed using a Core ML segmentation model to locate lung regions.
2. Sharpness and focus are evaluated inside the segmented lung region using Sobel filters and Laplacian variance.

3. If the image fails quality checks, the app instructs the user to retake the image, ensuring only diagnostically meaningful images are processed.

4. If the image passes the quality check, the user can generate:

    - Grad-CAM heatmaps

    - Multi-label disease predictions

5. The results and overlays are all displayed within the app.

The entire workflow is offline, self-contained, and optimized for real-time decision support.

4. Inputs

- App Installation

    - Users download and build the application directly from the Xcode project files.

    - Currently designed for iOS devices (CoreML-based).

- User Input

    - The user captures a picture of a chest X-ray using the in-app camera.

    - The image must:

        - Contain the full radiograph

        - Be reasonably well-lit

        - Be taken straight-on (no tilt) for best results

5. Outputs

After processing, the system outputs:

5.1 User-Captured Original Image

- Displayed immediately upon capture and after final results.

5.2 Grad-CAM Heatmap Overlay

- Highlights the areas most important for the prediction.
- Overlaid directly onto the X-ray image.

5.3 Diagnostic Predictions

- Final multi-label results for 14 chest X-ray categories.
- Shown with:
    - Probability scores
    - Clear labels

These outputs provide both interpretability and diagnostic guidance.

6. Features

- Fully offline, on-device AI pipeline
- iOS-native CoreML inference
- Real-time camera guidance
- Automated quality control (lung segmentation + sharpness scoring)
- Grad-CAM-based explainability
- Multi-label chest pathology classifier (14 classes)
- User-friendly UI with instant feedback
- No external servers, APIs, or internet required

Best Practices

1. Code Updates & Maintenance

1.1 Keep OpenCV2 Updated

The C++ and OpenCV 2 modules used for image processing should be periodically updated to the latest stable OpenCV version. This ensures:

- Compatibility with new Apple Silicon chips
- Reduced deprecated API usage
- Improved performance for edge detection and image operations
- Fewer issues when bridging between Swift and C++

1.2 Swift Version Maintenance (Swift 5)

The entire app is written in **Swift 5**, a stable and widely supported version of the language. As future Xcode versions introduce newer Swift standards:

- Review deprecation warnings regularly
- Use Xcode's automated migration tools
- Maintain backward compatibility with minimal code changes

1.3 iOS Deployment Target (17.6 Recommended)

The recommended minimum deployment version is **iOS 17.6**, ensuring:

- CoreML Compute Unit optimizations
- Full Vision framework support for Grad-CAM generation
- Stable AVFoundation and SwiftUI behavior
- Updated camera permission and privacy flow requirements

Earlier versions may limit performance or break features.

1.4 Continuous Maintenance of ML-Dependent Code

Because ML workflows evolve rapidly:

- Validate CoreML model formats after every Xcode update
- Rebuild ONNX → CoreML conversion pipelines when PyTorch versions change
- Re-test Grad-CAM generation whenever the framework updates

2. Libraries (Mobile Application Dependencies)

2.1 Swift

Primary coding language (>90%). Maintain a consistent coding style by adhering to Swift guidelines.

2.2 SwiftUI

Used for all UI views. Best practices include:

- MVVM architecture
- Consistent use of reactive bindings
- Avoiding heavy computation on the main thread

2.3 Photos

Provides the ability to save and display captured images. Best practices:

- Maintain user privacy with clear permission flows
- Reduce duplicates in photo library

2.4 AVFoundation

Handles camera sessions. Best practices:

- Use appropriate session presets for X-ray capture

- Lock focus/exposure to prevent flickering

2.5 WebKit

Used for displaying help GIFs. Use cached lightweight assets to minimize load time.

2.6 CoreML

Runs both lung segmentation and classification models. Best practices:

- Use .mlmodelc compiled format for faster load
- Choose compute units that balance speed and energy
- Keep model sizes small via quantization (when possible)

2.7 Vision

Used for Grad-CAM generation. Best practices:

- Use consistent pixel buffer formats across pipelines
- Offload heavy operations to background queues

2.8 C++ Interfacing with OpenCV2

Used for high-performance image operations. Best practices:

- Maintain clean bridging headers
- Avoid memory leaks across Swift/C++ boundaries
- Minimize C++ footprint to only performance-critical functionality

3. Repository Management

3.1 GitHub Workflow

The project is version-controlled through GitHub. Recommended practices:

- Branch naming conventions

- Pull requests with mandatory review
- Tagging stable releases
- Storing large files using Git LFS
- Documentation updates tied to each major code update

4. Machine Learning Training Best Practices

4.1 Dataset Management

- Maintain a clean directory structure with clear naming (e.g., white_closerin, yellow_furtherout).
- Ensure all training, validation, and domain adaptation subsets are reproducible.
- Document all pre-processing transformations used during training so they match on-device preprocessing.
- Check data distribution to avoid unseen class imbalance during fine-tuning.

4.2 Model Architecture Management

Stage 1 – CheXpert Pretraining

- Keep a fixed version of the DenseNet121 checkpoint for consistency.
- Store metadata such as:
    - Epoch count
    - Loss curve
    - Validation AUC
    - Normalization parameters

Stage 2 – EfficientNet-V2-S Fine-Tuning

- Keep the EfficientNet-V2-S configuration stable across experiments.

- Maintain consistency:
    - Learning rate
    - Batch size
    - Weight decay values
    - Mixed precision usage
- Track the best epoch using validation AUC.

4.3 Training Environment Best Practices

- Use consistent CUDA versions across machines when possible.
- Log training runs using:
    - CSV logs
- Train on stable hardware and document the type of GPU used.

4.4 Preprocessing & Augmentation Consistency

Ensure all image preprocessing done during training is **identically replicated on-device**.

Training pipeline included:

- Resize to 224×224
- Normalization
- RandomResizedCrop
- HorizontalFlip
- Color jitter
- Noise injection

4.5 Loss Functions & Optimization

- Use **Asymmetric Loss (ASL)** for multi-label imbalance.
- Use **AdamW** for stable training with weight decay.

Track:

- Validation AUC
- Per-class AUC
- Training loss curves
- Overfitting indicators

4.6 Deployment & Testing Best Practices

- Test on multiple iPhone models
- Validate:
    - Inference time
    - Memory usage
    - Grad-CAM
    - Segmentation model performance
    - Edge-case images

Transferring ownership to the sponsor

1. Code Repository & Models

1.1 Source Code Repository

All project source code, including the iOS application, camera pipeline, image quality assessment modules, Grad-CAM implementation, and CoreML integration, is stored in the project's GitHub repository.

Repository Link:
Github Link

Credentialing :

- Xcode -> signing and capabilities ->Signing
- Include team and bundle identifier for future development and potential app store addition

2. Contact Information

For questions, onboarding help, future updates, or project continuity, the following individuals can be contacted:

2.1 Project Members

Connor McMahon
📧 *mcmahon.connor.04@gmail.com*

Sanskar Srivastava
📧 *ssriva94@asu.edu*

Both contributors are familiar with:

- The iOS application architecture
- Camera and quality assessment pipeline
- Machine learning training and conversion workflows
- CoreML integration
- Project documentation and design decisions

Sponsor Approval:

Please provide your feedback below:

By signing this document, I, the sponsor, accept and recognize that I have got all the technical documentation, user guides, best practices, and ownership of all the components of the project.

Signature of the sponsor