Chest-X-Ray-LENS: Transfer Learning in Pre-trained Models Through Fine-tuning

Andy Chen, Benjamin Dai, Ryan Ma, Yihao Mai

Background

- There are many medical applications for deep learning models, especially with classification problems
- Current day models can detect chest-related diseases within a reasonable degree of certainty, but offer no explanation as to how it reached those results
    - “Black box” nature of such models
- Existing workflows require waiting for a radiologist’s report after X-rays, which, in a time-sensitive environment, can lead to negative outcomes if potential treatment is pushed back
- Having more robust and useful models would help promote adoption in the medical field

Motivation

- AI models can dramatically expedite the diagnosis of thoracic pathologies, helping doctors work faster.

1. We would like to see if we can finetune a model that can perform well on the ChexPert dataset
    - Increase confidence in AI tools coming from well constructed models
2. We would like to help explain the inferences that AI models make for chest x-rays

Existing Work

- **Deep Learning in Medical Image Analysis**: “Early studies of deep learning applied to lesion detection or classification have reported superior performance compared to those by conventional techniques or even better than radiologists in some tasks” (Chan et al, 2020)
- **CheXNet: Radiologist-Level Pneumonia Detection on Chest X-Rays with Deep Learning**: “An algorithm that can detect pneumonia from chest X-rays at a level exceeding practicing radiologists… CheXNet, is a 121-layer [CNN] trained on ChestX-ray14… state of the art results on all 14 diseases.” (Rajpurkar et al, 2017)
- **CheXpert: A Large Chest Radiograph Dataset with Uncertainty Labels and Expert Comparison**: AAAI 2019 Conference paper publicly releasing CheXpert as a standard benchmark dataset. (Irvin et al, 2019)

Chest-X-Ray-LENS Overview

Our objective is to finetune different models to perform well on the CheXpert dataset and generate meaningful results for doctors to use, including x-ray classification AI interpretability tooling.

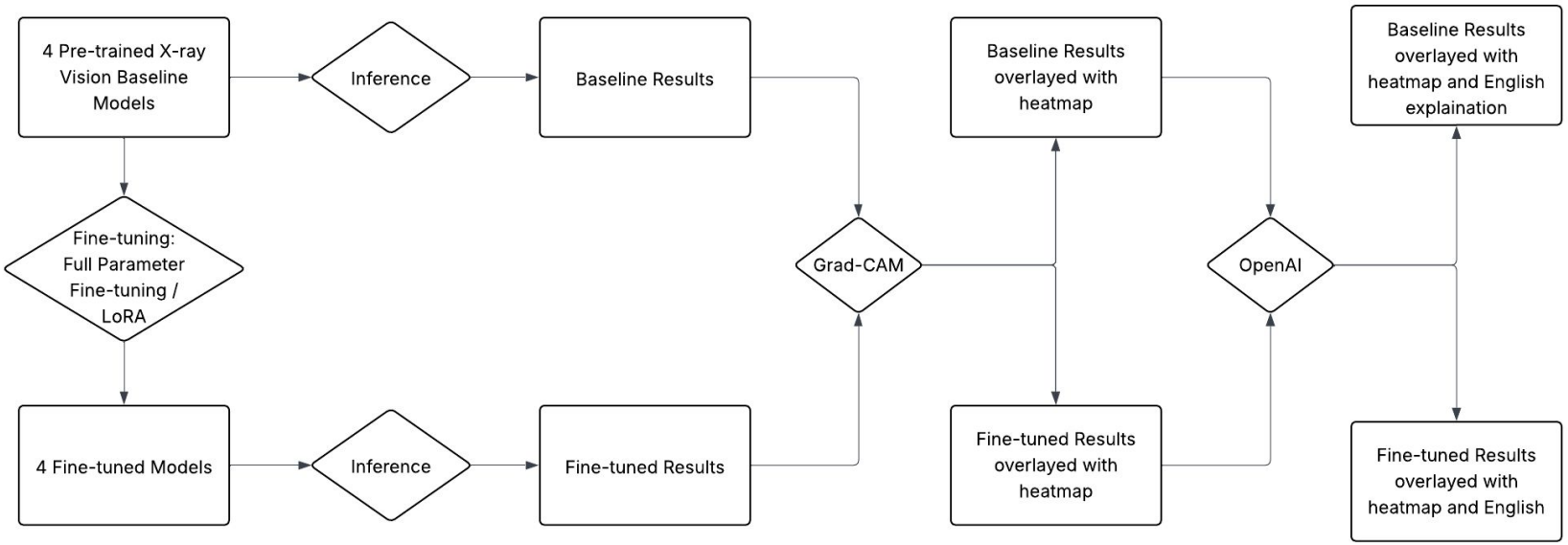
Our 3-Step Process

1. **Baseline:** Run models as is on the CheXpert dataset and store results
2. **Fine-tune:** Fine-tune (full parameter fine tuning / LoRA) the models, train on CheXpert, and store results
3. **Inference:** Generate heatmap images of predictions of the baseline and fine-tuned models along with natural language explanation

Hardware and Software Configuration

1. **CPU:** Dual Xeon Platinum 8462Y+ (64 cores/node, 2.8GHz)
2. **GPU:** H100 SXM5 80GB
3. **Memory:** 2048GB DDR5 4800 MHz
   ** all above hardwares are accessible through Georgia Tech ICE cluster
4. Software configuration (e.g. training epochs, learning rate, and LoRA config) is discussed in each pre-trained model section later

High-Level System Architecture



Dataset: CheXpert

- 223, 648 chest radiographs of 65,240 patients
- Training (223,414) and Validation (234) sets
- Only consider frontal image samples since they captures most clinically relevant information
    - Training (191,027) and Validation (202) sets
- Split training set into 8:2 for training and validation
- The original validation set serves as the testing set during inference
- https://www.kaggle.com/datasets/willarevalo/chexpert-v10-small

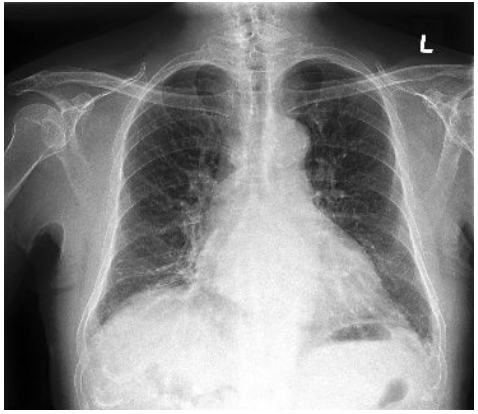
(Below table shows each label positive and negative count in training and validation sets:)

Finding	Positive_Train	Negative_Train	Positive_Valid	Negative_Valid
No Finding	16974	174053	26	176
Enlarged Cardiomediastinum	9187	181840	105	97
Cardiomegaly	23385	167642	66	136
Lung Opacity	94211	96816	117	85
Lung Lesion	7040	183987	1	201
Edema	49675	141352	42	160
Consolidation	12983	178044	32	170
Pneumonia	4675	186352	8	194
Atelectasis	29720	161307	75	127
Pneumothorax	17693	173334	7	195
Pleural Effusion	76899	114128	64	138
Pleural Other	2505	188522	1	201
Fracture	7436	183591	0	202
Support Devices	107170	83857	99	103

Train/Val Sample Data

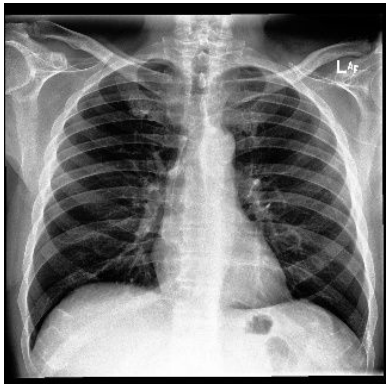
```
Path,Sex,Age,Frontal/Lateral,AP/PA,No Finding,Enlarged Cardiomediastinum,Cardiomegaly,Lung Opacity,Lung Lesion,Edema,Consolidation,Pneumonia,Atelectasis,Pneumothorax,P
CheXpert-v1.0-small/train/patient00001/study1/view1_frontal.jpg,Female,68,Frontal,AP,1.0,,,,,,,,0.0,,,,1.0
CheXpert-v1.0-small/train/patient00002/study2/view1_frontal.jpg,Female,87,Frontal,AP,,,-1.0,1.0,,-1.0,-1.0,,-1.0,,-1.0,,1.0,
CheXpert-v1.0-small/train/patient00002/study1/view1_frontal.jpg,Female,83,Frontal,AP,,,,1.0,,,-1.0,,,,,,1.0,
CheXpert-v1.0-small/train/patient00002/study1/view2_lateral.jpg,Female,83,Lateral,,,,,1.0,,,-1.0,,,,,,1.0,
CheXpert-v1.0-small/train/patient00003/study1/view1_frontal.jpg,Male,41,Frontal,AP,,,,,1.0,,,,0.0,,,,
CheXpert-v1.0-small/train/patient00004/study1/view1_frontal.jpg,Female,20,Frontal,PA,1.0,0.0,,,,,0.0,,,,0.0,,,
CheXpert-v1.0-small/train/patient00004/study1/view2_lateral.jpg,Female,20,Lateral,,1.0,0.0,,,,,0.0,,,,0.0,,,
CheXpert-v1.0-small/train/patient00005/study1/view1_frontal.jpg,Male,33,Frontal,PA,1.0,,0.0,,,,0.0,,,,0.0,,,1.0
CheXpert-v1.0-small/train/patient00005/study1/view2_lateral.jpg,Male,33,Lateral,,1.0,,0.0,,,,0.0,,,,0.0,,,1.0
```

**Null and -1.0 value labels are uncertain labels. We treat them as negative labels (0.0) for simplicity instead of removing all those samples



Test Sample Data

```
Path,Sex,Age,Frontal/Lateral,AP/PA,No Finding,Enlarged Cardiomediastinum,Cardiomegaly,Lung Opacity,Lung Lesion,Edema,Consolidation,Pneumonia,Atelectasis,Pneumothorax,Pleu
CheXpert-v1.0-small/valid/patient64541/study1/view1_frontal.jpg,Male,73,Frontal,AP,0.0,1.0,1.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
CheXpert-v1.0-small/valid/patient64542/study1/view1_frontal.jpg,Male,70,Frontal,PA,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0
CheXpert-v1.0-small/valid/patient64542/study1/view2_lateral.jpg,Male,70,Lateral,,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0
CheXpert-v1.0-small/valid/patient64543/study1/view1_frontal.jpg,Male,85,Frontal,AP,0.0,1.0,0.0,1.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
CheXpert-v1.0-small/valid/patient64544/study1/view1_frontal.jpg,Female,42,Frontal,AP,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
CheXpert-v1.0-small/valid/patient64545/study1/view1_frontal.jpg,Female,55,Frontal,AP,0.0,1.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,1.0,0.0,0.0,0.0
CheXpert-v1.0-small/valid/patient64546/study1/view1_frontal.jpg,Male,56,Frontal,AP,0.0,1.0,1.0,1.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,1.0
CheXpert-v1.0-small/valid/patient64547/study1/view1_frontal.jpg,Male,59,Frontal,PA,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0
CheXpert-v1.0-small/valid/patient64547/study1/view2_frontal.jpg,Male,59,Frontal,PA,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0
```



Model One: densenet121-res224-pc

- A pre-trained model on PadChest dataset from torchxrayvision
- 6,960,909 parameters
- 15 pre-trained labels:
  - Atelectasis, Consolidation, Infiltration, Pneumothorax, Edema, Emphysema, Fibrosis, Effusion, Pneumonia, Pleural_Thickening, Cardiomegaly, Nodule, Mass, Hernia, Fracture
    - Only **8** of them matched with the ChestXpert

Model One Finetune Workflow: baseline inference

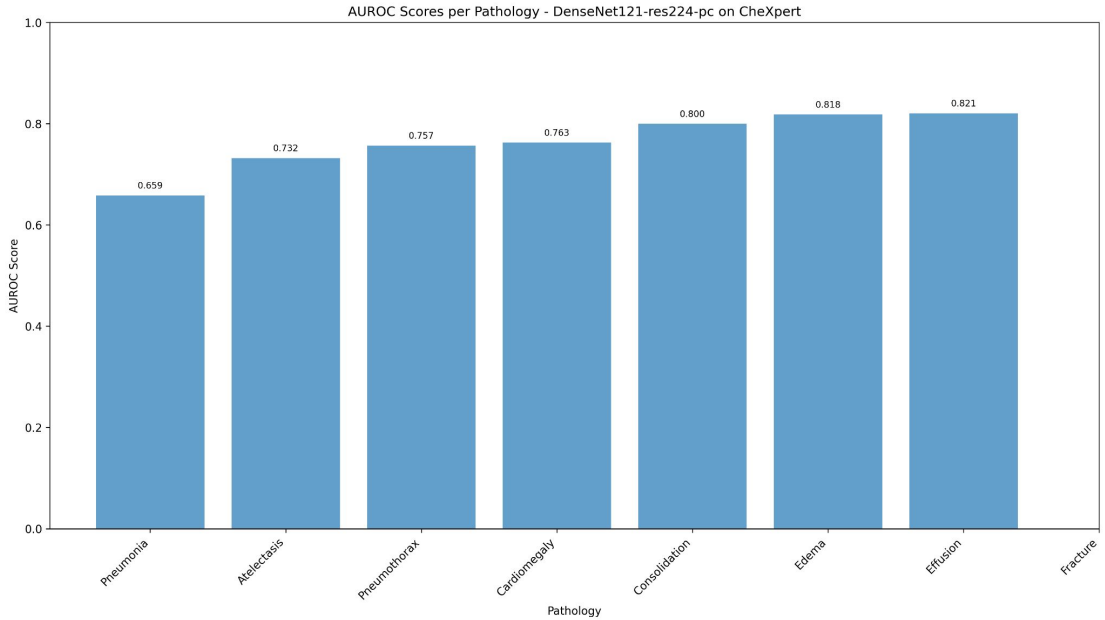
- Run a baseline inference on validation set
  - Use torch **Dataset** class and chexpert help class from torchxrayvision to load the data
  - Define a **LabelMatcher** class:
    - Extracts all matched labels between model and dataset
    - Only matched labels are considered during inference
  - Define a **ComputeMetrics** class:
    - Runs the evaluation loop

```python
dataloader = DataLoader(
    d_chex, # torchxrayvision helper dataset class
    batch_size=batch_size, # 16
    shuffle=False,
    num_workers=2
)
```

```python
class LabelMatcher:
    """
    A class to handle pathology label matching between models and datasets.

    This class provides methods to:
    - Match pathology labels between model and dataset
    - Print matching results and analysis
    - Provide comprehensive reporting of matched/unmatched labels
    """

    def __init__(self):
        """
        Initialize the LabelMatcher.
        """
        self.pathology_mapping = {}
        self.model_pathologies = []
        self.dataset_pathologies = []
        self.model_pathologies_clean = []
        self.dataset_pathologies_clean = []
```

```python
def evaluate_model(self,
                   model: Any,
                   dataloader: DataLoader,
                   pathology_mapping: Dict[int, Tuple[int, str]],
                   device: str = 'cuda' if torch.cuda.is_available() else 'cpu') -> Dict[str, Any]:
    """
    Evaluate model on given dataloader and compute metrics.

    Args:
        model: The trained model to evaluate
        dataloader: DataLoader containing validation/test data
        pathology_mapping: Mapping from model indices to (dataset_idx, pathology_name)
        device: Device to run evaluation on

    Returns:
        Dictionary containing all computed metrics
    """
    print(f"Starting model evaluation on {device}")
    print(f"Evaluating {len(pathology_mapping)} matched pathologies")

    model.eval()
    model = model.to(device)

    all_predictions = []
    all_labels = []
    with torch.no_grad():
        for batch_idx, batch in enumerate(dataloader):
            if batch_idx % 100 == 0:
                print(f"Processing batch {batch_idx}/{len(dataloader)}")

            images = batch['img'].to(device)
            labels = batch['lab'].to(device)
            outputs = model(images)
            predictions = torch.sigmoid(outputs).detach().cpu().numpy()
                all_predictions.append(predictions)
                all_labels.append(labels.detach().cpu().numpy())

    all_predictions = np.concatenate(all_predictions, axis=0)
    all_labels = np.concatenate(all_labels, axis=0)

    results = self._compute_all_metrics(
        all_predictions,
        all_labels,
        pathology_mapping
    )
    self.results = results

    return results
```

Model One Finetune Workflow: baseline inference result



Model One Finetune Workflow: LoRA fine tuning

- Define a **CheXpertFineTuner** class:
  - Swap out the unmatched labels from the classifier head and inject new label from chestxpert as well as maintaining the matched labels' weights.
- Define a **LoRAModel** class:
  - A wrapper for the fine-tuned model
  - Applies the LoRA adapters to the model
  - Freeze original model's parameters

```python
def load_pretrained_model(self, model_name: str = "densenet121-res224-pc"):
    """Load pretrained model, transfer matching weights, and set up for CheXpert fine-tuning."""

    original_model = xrv.models.DenseNet(weights=model_name)

    # Get original classifier and its labels
    original_classifier = original_model.classifier
    in_features = original_classifier.in_features

    # Get the 18 labels from the original pretrained model
    original_labels = original_model.pathologies

    # Create the new 13-class classifier for CheXpert
    num_chexpert_classes = len(self.chexpert_labels)
    new_classifier = nn.Linear(in_features, num_chexpert_classes)

    print("TRANSFERRING CLASSIFIER WEIGHTS (HEAD-SWAP)")
    copied_weights = 0
    with torch.no_grad():
        for new_idx, chexpert_label in enumerate(self.chexpert_labels):
            try:
                # Find this label in the *original* 18-label list
                old_idx = original_labels.index(chexpert_label)

                # Copy weights and bias
                new_classifier.weight.data[new_idx] = original_classifier.weight.data[old_idx]
                new_classifier.bias.data[new_idx] = original_classifier.bias.data[old_idx]
                print(f"Copied: {chexpert_label} (Old index {old_idx} -> New index {new_idx})")
                copied_weights += 1
            except ValueError:
                # This CheXpert label wasn't in the original model
                print(f"New: {chexpert_label} (New index {new_idx}) - will be randomly initialized.")
    # Now, replace the head on the original model
    original_model.classifier = new_classifier
```

Model One Finetune Workflow: LoRA fine tuning continue

```python
lora_config = {
    'r': 16,
    'lora_alpha': 32,
    'target_modules': [],
    'lora_dropout': 0.1,
}
```

```python
def _apply_lora(self, lora_config: Dict):
    """Apply LoRA adapters to selected Conv2d layers in backbone."""
    try:
        conv_leaf_names: List[str] = []
        for name, module in self.base_model.named_modules():
            if isinstance(module, nn.Conv2d) and name.startswith('features'):
                conv_leaf_names.append(name)

        if not conv_leaf_names:
            print("No Conv2d layers found under features; skipping LoRA.")
            return

        conv1_layers = [n for n in conv_leaf_names if n.endswith('conv1')]
        conv2_layers = [n for n in conv_leaf_names if n.endswith('conv2')]

        # Pick last 6 conv2 (3x3) and last 2 conv1 (1x1) as default
        # selected = conv2_layers[-6:] + conv1_layers[-2:]
        # Pick all conv layers
        selected = conv2_layers + conv1_layers

        lora_config['target_modules'] = selected

        peft_config = LoraConfig(**lora_config)
        self.base_model = get_peft_model(self.base_model, peft_config)
```

```python
def _freeze_parameters(self):
    """Freeze backbone parameters except LoRA and classifier."""
    for name, param in self.base_model.named_parameters():
        if 'lora_' in name:
            param.requires_grad = True
        # Keep classifier parameters trainable because peft has frozen the classifer too
        # and we need to fine-tune it for the new CheXpert labels
        elif 'classifier' in name:
            param.requires_grad = True
        # Freeze everything else (backbone)
        else:
            param.requires_grad = False
```

- Applying LoRA adapter to all conv layers can produce comparable performance to the full parameter fine tuning, according to the LoRA paper
- Total parameters: 8,682,765
- Trainable parameters: 1,735,181
- Trainable percentage: 19.98%

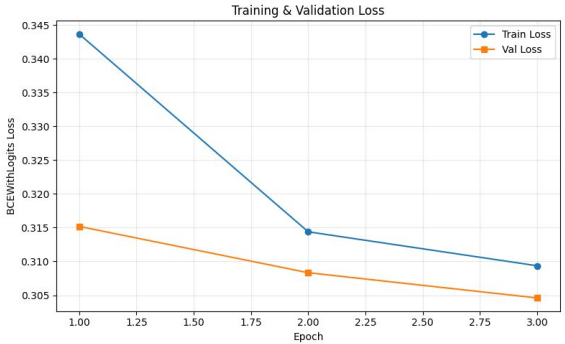
Model One Finetune Workflow: LoRA fine tuning continue

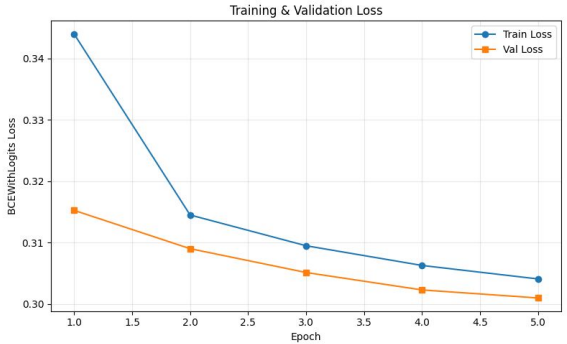
```
criterion = nn.BCEWithLogitsLoss()
optimizer = optim.AdamW(
    filter(lambda p: p.requires_grad, self.model.parameters()),
    lr=learning_rate,
    weight_decay=1e-5
)

scheduler = optim.lr_scheduler.ReduceLROnPlateau(
    optimizer, mode='min', factor=0.5, patience=3)
```

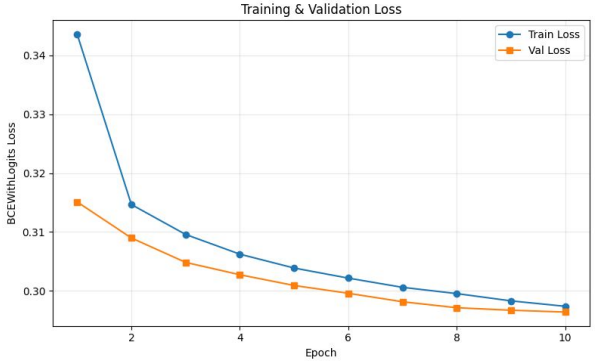
```
def _train_epoch(self, criterion, optimizer):
    """Train for one epoch."""
    self.model.train()
    total_loss = 0.0
    batch_losses: List[float] = []

    for batch_idx, batch in enumerate(self.train_loader):
        images = batch['img'].to(self.device)
        targets = batch['lab'].to(self.device)

        # convert nan and -1 to 0 for binary targets
        binary_targets = self._convert_targets_to_binary(targets)

        optimizer.zero_grad()

        outputs = self.model(images)
        loss = criterion(outputs, binary_targets)

        loss.backward()
        optimizer.step()

        total_loss += loss.item()
        batch_losses.append(loss.item())

        if batch_idx % 50 == 0:
            print(f"Batch {batch_idx}/{len(self.train_loader)}, Loss: {loss.item():.4f}")

    avg_loss = total_loss / len(self.train_loader)
    return avg_loss, batch_losses
```

```
def _validate_epoch(self, criterion):
    """Validate for one epoch."""
    self.model.eval()
    total_loss = 0.0
    batch_losses: List[float] = []

    with torch.no_grad():
        for batch in self.val_loader:
            images = batch['img'].to(self.device)
            targets = batch['lab'].to(self.device)

            binary_targets = self._convert_targets_to_binary(targets)

            outputs = self.model(images)
            loss = criterion(outputs, binary_targets)

            total_loss += loss.item()
            batch_losses.append(loss.item())

    avg_loss = total_loss / len(self.val_loader)
    return avg_loss, batch_losses
```

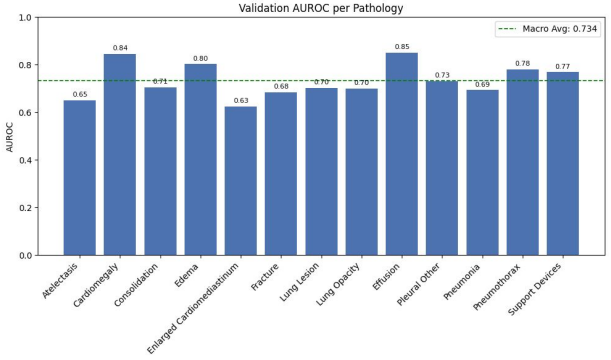
Model One Finetune Workflow: LoRA fine tuning results - 3,5,10 epochs

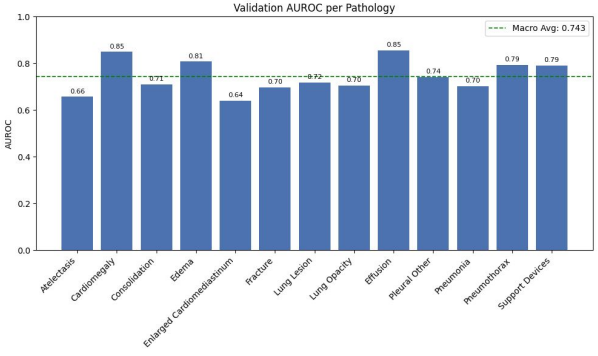


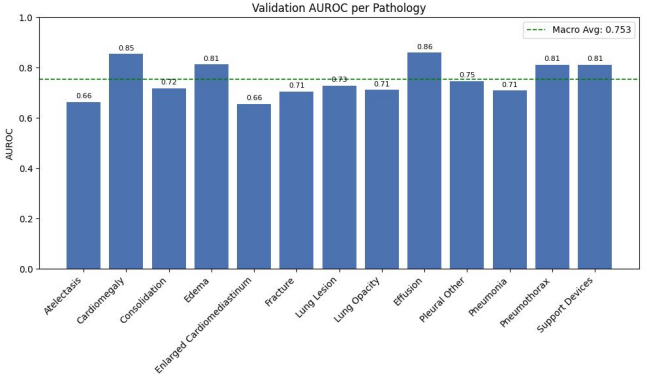




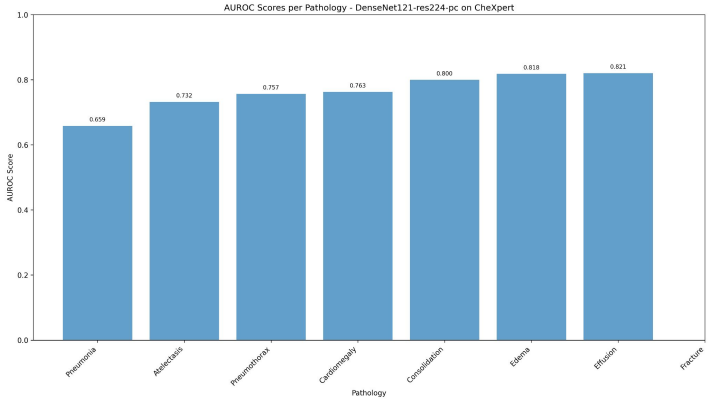
Model One Finetune Workflow: LoRA fine tuning results - 3,5,10 epochs



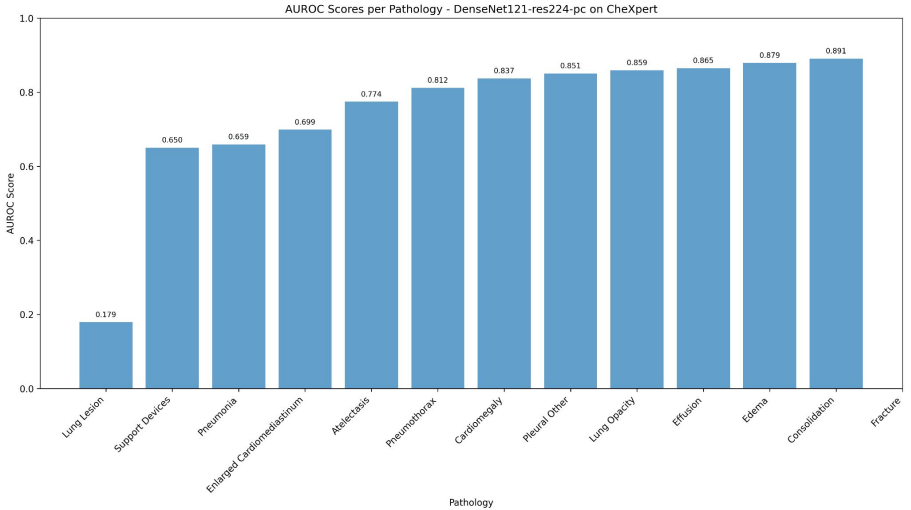




Model One Finetune Workflow: fine tuned model inference



Baseline



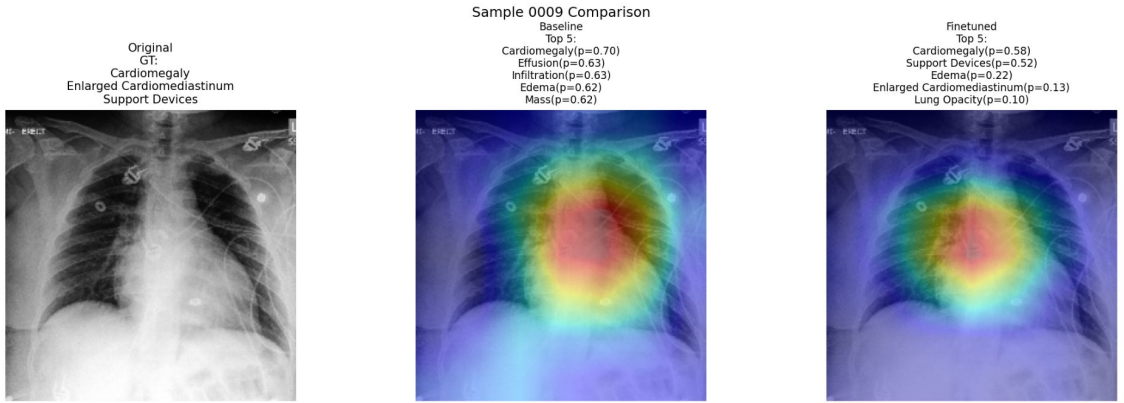
LoRA Fine Tuned

Model One Finetune Workflow: add explainability

- Use **Grad-CAM** to overlay heatmap on top of the image to explain where the model focuses on to make the predictions
- Since Model One is only classification model, we integrate OpenAI model's to extend the explainability with natural language to summarize why the model is making good or bad predictions.

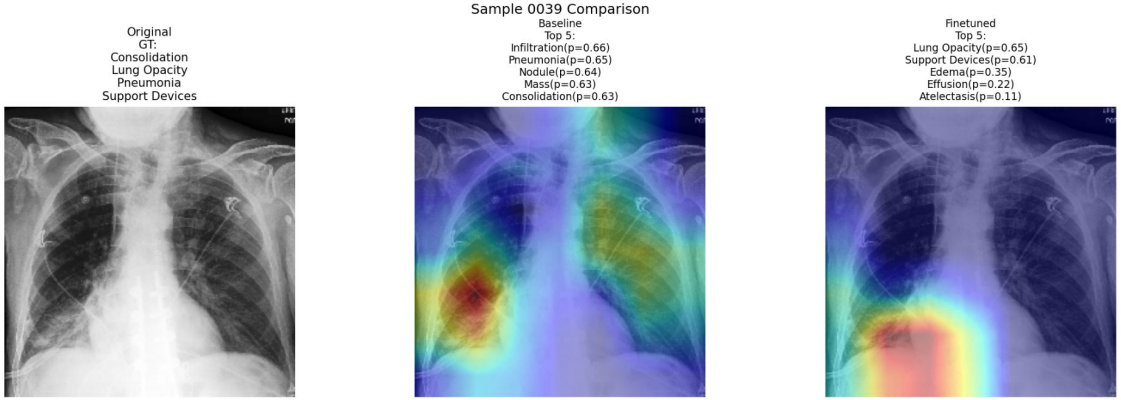
```
prompt = f"""
    You are an expert radiology AI explainer.
    Your task is to interpret two model-generated Grad-CAM heatmaps from the same chest X-ray:
    (1) a baseline model's heatmap and prediction
    (2) a fine-tuned model's heatmap and prediction.
    You will be given one image, which contains three parts:
    - The original X-ray image with ground truth labels at the top
    - The baseline model's heatmap overlaid on the original image with top 5 predicted labels and probabilities at the top
    - The fine-tuned model's heatmap overlaid on the original image with top 5 predicted labels and probabilities at the top

    There are three task types you need to perform:
    1. easy: both models are performing well
    2. averge: the finetune model performs well in some cases, and the baseline model performs well on cases where the fine-tuned model struggles
    3. hard: both models are struggling

    Note that if no task type is specified, you should infer the task type based on the heatmaps and model predictions compared to the ground truth.

    Base on the type task, please perform the following steps clearly and systematically:

    1. **Extract the ground truth labels and the top predicted labels with probabilities from both models.**

    2. **If the task is 'easy', describe both models' behavior**
    - Where is their attention concentrated?
    - Are these regions anatomically relevant to the ground truth pathology?

    3. **If the task is 'average', describe both models' behavior**
    - Where is their attention concentrated?
    - How does one model perform better than the other in terms of attention focus and relevance to the ground truth?
    - How does one model perform worse than the other in terms of attention focus and relevance to the ground truth?

    4. **If the task is 'hard', describe both models' behavior**
    - Where is their attention concentrated?
    - Are these regions anatomically relevant to the ground truth pathology?
    - Why might both models be struggling to focus on the relevant areas?

    Please analyze the attached heatmap-overlaid X-ray image with task type {task_type} and provide a structured explanation.
    You should summarize your findings in at most 5 sentences.
    """
```

Model One Easy Case



1) Ground truth: Cardiomegaly; Enlarged Cardiomediastinum; Support Devices. Baseline top 5: Cardiomegaly (p=0.70), Effusion (p=0.63), Infiltration (p=0.63), Edema (p=0.62), Mass (p=0.62). Fine-tuned top 5: Cardiomegaly (p=0.58), Support Devices (p=0.52), Edema (p=0.22), Enlarged Cardiomediastinum (p=0.13), Lung Opacity (p=0.10).
2) Both heatmaps concentrate on the cardiac silhouette and central mediastinum, with the baseline showing a broader, more diffuse hot region over the whole heart and lower mediastinum and the fine-tuned model showing a tighter central focus over the cardiac outline (and additional attention near electrode/line sites).
3) These regions are anatomically appropriate for the ground-truth labels (cardiomegaly and enlarged cardiomediastinum) and the fine-tuned model's attention on peripheral device areas matches the "Support Devices" label.
4) Overall the fine-tuned model demonstrates more specific, clinically relevant localization and correctly promotes Support Devices higher, while the baseline's broader focus likely contributed to false-positive suggestions like effusion/infiltrate.

Model One Average Case



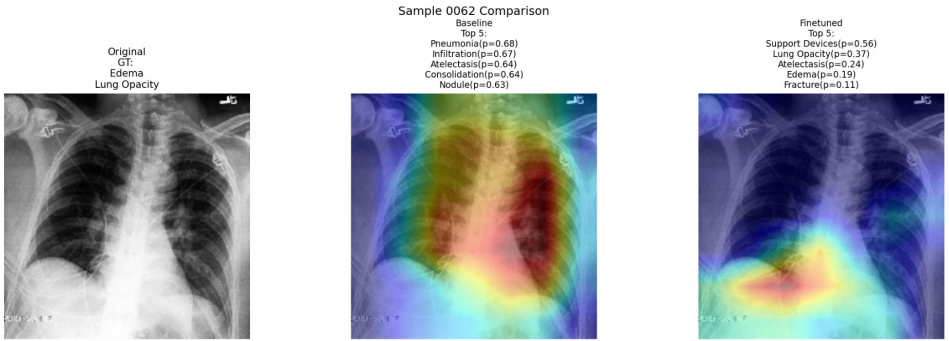
1) Ground truth: Consolidation; Lung Opacity; Pneumonia; Support Devices. Baseline top-5: Infiltration (0.66), Pneumonia (0.65), Nodule (0.64), Mass (0.63), Consolidation (0.63). Finetuned top-5: Lung Opacity (0.65), Support Devices (0.61), Edema (0.35), Effusion (0.22), Atelectasis (0.11).

2) Baseline behavior: the baseline Grad-CAM shows a tight, high-intensity focus over a focal mid-to-lower pulmonary region adjacent to the cardiac silhouette and diaphragm, which is anatomically consistent with a localized consolidation/infiltrate and supports its high Pneumonia/Infiltration/Consolidation scores.

3) Finetuned behavior: the fine-tuned map is more diffuse and concentrated inferiorly over the cardiophrenic/central lower lung region and over the anterior chest near visible hardware/support devices, matching its high Lung Opacity and Support Devices predictions but less specifically localizing the focal consolidation.

4) Comparison: the baseline better localizes the focal consolidation (better attention focus and relevance to Pneumonia/Consolidation), while the fine-tuned model is better at detecting support devices and global lung opacity but is more diffuse and less precise, causing it to miss the exact focal pathology; this trade-off explains why each model succeeds on different aspects of this case.

Model One Hard Case



1) Ground truth: Edema; Lung Opacity. Baseline top-5: Pneumonia (p=0.68), Infiltration (p=0.67), Atelectasis (p=0.64), Consolidation (p=0.64), Nodule (p=0.63). Finetuned top-5: Support Devices (p=0.56), Lung Opacity (p=0.37), Atelectasis (p=0.24), Edema (p=0.19), Fracture (p=0.11).
2) The baseline heatmap shows very broad, central/left-predominant attention over the cardiac silhouette and middle-to-lower lung fields (diffuse central hotmap extending into the left hemithorax).
3) The fine-tuned model concentrates lower and more medially around the cardiophrenic/mediastinal region and diaphragmatic recesses with a tighter hotspot, and a smaller locus on the right lateral lung.
4) These foci are only partially anatomically relevant: edema and diffuse lung opacity often present as bilateral perihilar/interstitial change, but both models emphasize the heart/central lower chest (and the finetuned also flags support-devices), so they are not clearly localizing the diffuse interstitial pattern.
5) Both models likely struggle because the pathology is subtle and diffuse (low contrast interstitial edema), projection/overlap with the heart and diaphragmatic contours and the presence of external devices can confound attention, producing central or device-related saliency rather than clear bilateral lung-field localization.

Model Two: densenet121-res224-nih

- TorchXRayVision DenseNet121 model pretrained only on NIH Chest X-Ray 14 dataset
  - https://github.com/mlmed/torchxrayvision
- Comes pretrained outputting 8 label classes:
  - ['Atelectasis', 'Cardiomegaly', 'Effusion', 'Infiltration', 'Mass', 'Nodule', 'Pneumonia', 'Pneumothorax']
  - Only has 5 true overlapping labels - Atelectasis, Cardiomegaly, Effusion, Pneumonia, Pneumothorax
- 6,966,034 total parameters

NIH Model Setup

- Mapped overlapping labels, set training variables, loaded train/val sets
  - Had to replace NaN with 0 and -1 with 0
  - CheXpertDataset wrapper class

```
train_df, val_df = load_chexpert_splits()
train_ds = CheXpertDataset(train_df, DATA_DIR)
val_ds = CheXpertDataset(val_df, DATA_DIR)

train_loader = torch.utils.data.DataLoader(
    train_ds, batch_size=BATCH_SIZE, shuffle=True, num_workers=4, pin_memory=True
)
val_loader = torch.utils.data.DataLoader(
    val_ds, batch_size=BATCH_SIZE, shuffle=False, num_workers=4, pin_memory=True
)
```

```
CHEXPERT_CLASSES = [
    "No Finding",
    "Enlarged Cardiomediastinum",
    "Cardiomegaly",
    "Lung Opacity",
    "Lung Lesion",
    "Edema",
    "Consolidation",
    "Pneumonia",
    "Atelectasis",
    "Pneumothorax",
    "Pleural Effusion",
    "Pleural Other",
    "Fracture",
    "Support Devices",
]

NIH_PATHOLOGIES = [
    "Atelectasis",
    "Cardiomegaly",
    "Effusion",
    "Infiltration",
    "Mass",
    "Nodule",
    "Pneumonia",
    "Pneumothorax",
]

OVERLAP_PAIRS = [
    ("Atelectasis", "Atelectasis"),
    ("Cardiomegaly", "Cardiomegaly"),
    ("Effusion", "Pleural Effusion"),
    ("Pneumonia", "Pneumonia"),
    ("Pneumothorax", "Pneumothorax"),
]
```

```
def load_chexpert_splits():
    train_df = pd.read_csv(TRAIN_CSV)
    val_df = pd.read_csv(VAL_CSV)
    for df in [train_df, val_df]:
        df[CHEXPERT_CLASSES] = df[CHEXPERT_CLASSES].fillna(0)
        df[CHEXPERT_CLASSES] = df[CHEXPERT_CLASSES].replace(-1, 0)
        df[CHEXPERT_CLASSES] = df[CHEXPERT_CLASSES].astype(np.float32)
    return train_df, val_df
```

```
IMG_SIZE = 224
BATCH_SIZE = 32
EPOCHS = 8
LR = 1e-6
THRESHOLD = 0.6
```

```
class CheXpertDataset(torch.utils.data.Dataset):
    def __init__(self, df, root_dir):
        self.df = df.reset_index(drop=True)
        self.root_dir = root_dir
        self.transform = T.Compose([
            T.Resize((IMG_SIZE, IMG_SIZE)),
            T.ToTensor(),
            T.Normalize([0.5], [0.25]),
        ])

    def __len__(self):
        return len(self.df)

    def __getitem__(self, idx):
        row = self.df.iloc[idx]
        rel_path = row["Path"]
        if rel_path.startswith(self.root_dir + "/"):
            rel_path = rel_path[len(self.root_dir) + 1:]
        img_path = os.path.join(self.root_dir, rel_path)
        img = Image.open(img_path).convert("L")
        img = self.transform(img)
        labels = torch.tensor(row[CHEXPERT_CLASSES].values.astype(np.float32))
        return img, labels
```

NIH Model Setup

- Created AUROC compute function for eval use, loaded baseline model from xrv.models.DenseNet(weights="densenet121-res224-nih")

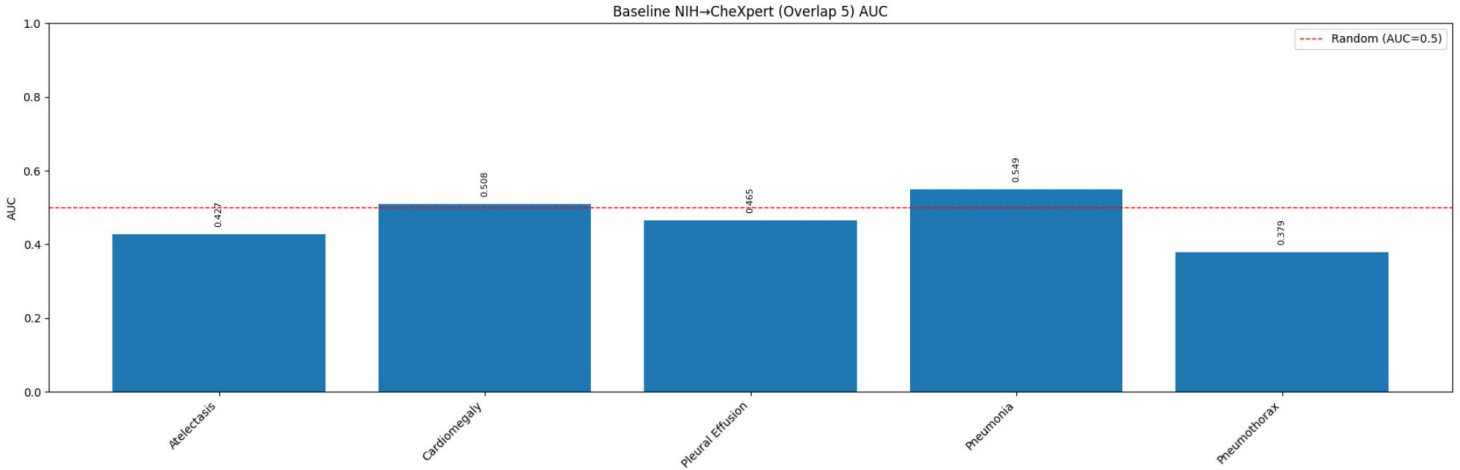
```
nih_model = xrv.models.DenseNet(weights="densenet121-res224-nih")
nih_model.op_norm = None
nih_model.op_threshs = None
nih_model.to(device)

chex_idx = [CHEXPERT_CLASSES.index(dst) for _, dst in OVERLAP_PAIRS]
nih_idx = [NIH_PATHOLOGIES.index(src) for src, _ in OVERLAP_PAIRS]
```

```
def compute_per_class_metrics_from_labels(class_names, y_true, y_prob, threshold):
    metrics = {}
    for i, cls in enumerate(class_names):
        t = y_true[:, i]
        p = y_prob[:, i]
        preds = (p > threshold).astype(int)
        tp = int(((preds == 1) & (t == 1)).sum())
        fp = int(((preds == 1) & (t == 0)).sum())
        tn = int(((preds == 0) & (t == 0)).sum())
        fn = int(((preds == 0) & (t == 1)).sum())
        prec = tp / (tp + fp + 1e-8)
        rec = tp / (tp + fn + 1e-8)
        f1 = 2 * prec * rec / (prec + rec + 1e-8)
        try:
            auc = roc_auc_score(t, p) if len(np.unique(t)) > 1 else None
        except:
            auc = None
        metrics[cls] = {
            "precision": float(prec),
            "recall": float(rec),
            "f1": float(f1),
            "auc": float(auc) if auc is not None else None,
            "tp": tp, "fp": fp, "tn": tn, "fn": fn
        }
    return metrics
```

NIH Model: Finetuning Workflow

```python
def evaluate_baseline_nih8_overlap5(model, loader, device, chex_idx, nih_idx):
    model.eval()
    all_probs, all_true = [], []
    with torch.no_grad():
        for imgs, labs in tqdm(loader, desc="Baseline NIH8→CheXpert5 Eval"):
            imgs = imgs.to(device)
            labs = labs.to(device)
            out = model(imgs)
            probs = torch.sigmoid(out)
            all_probs.append(probs[:, nih_idx].cpu().numpy())
            all_true.append(labs[:, chex_idx].cpu().numpy())
    return np.vstack(all_true), np.vstack(all_probs)
```

- First evaluated the baseline NIH model on the 5 overlapping labels
  - Performs at basically random AUROC = 0.5 because different output domain spaces



NIH Model: Finetuning Workflow

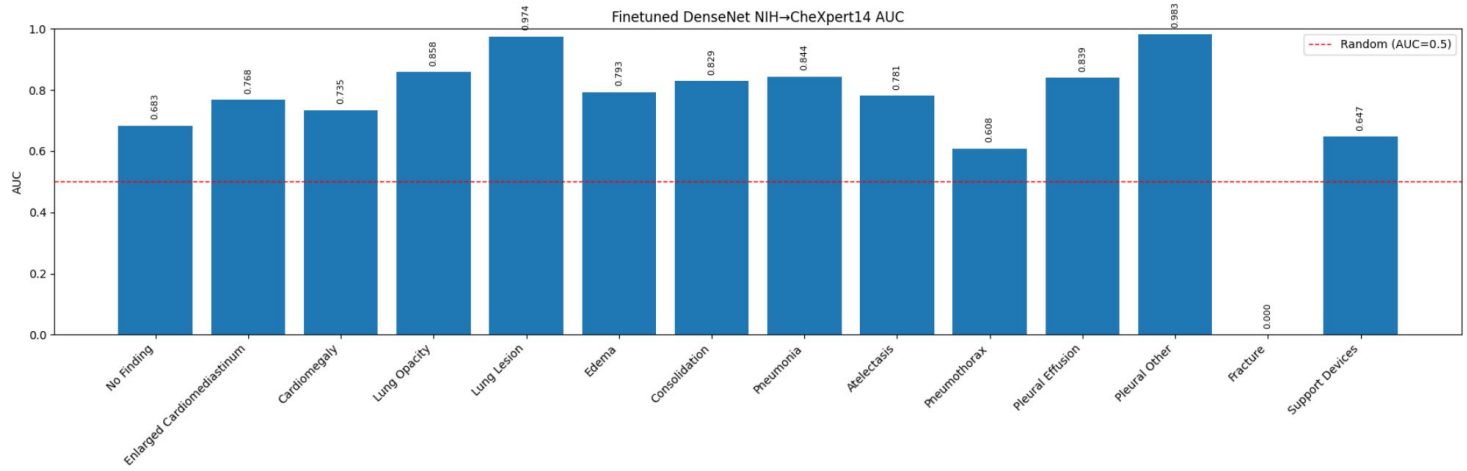
- Next performed full-parameter finetuning for 3 epochs with batch size 32 and learning rate 1e-6


```python
def finetune_full_14(model, train_loader, val_loader, device):
    train_losses = []
    val_losses = []

    criterion = nn.BCEWithLogitsLoss()
    optimizer = optim.Adam(model.parameters(), lr=LR)

    for ep in range(EPOCHS):
        model.train()
        total = 0.0
        for imgs, labs in tqdm(train_loader, desc=f"Finetune Epoch {ep+1}/{EPOCHS}"):
            imgs, labs = imgs.to(device), labs.to(device)
            out = model(imgs)
            loss = criterion(out, labs)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            total += loss.item()
        train_loss = total / len(train_loader)
        train_losses.append(train_loss)
        print(f"[Finetune] Epoch {ep+1} TRAIN loss: {train_loss:.4f}")

        model.eval()
        val_total = 0.0
        with torch.no_grad():
            for imgs, labs in val_loader:
                imgs, labs = imgs.to(device), labs.to(device)
                val_out = model(imgs)
                val_loss = criterion(val_out, labs)
                val_total += val_loss.item()
        val_loss_epoch = val_total / len(val_loader)
        val_losses.append(val_loss_epoch)
        print(f"[Finetune] Epoch {ep+1} VAL loss:   {val_loss_epoch:.4f}")
```

NIH Model: Finetuning Workflow

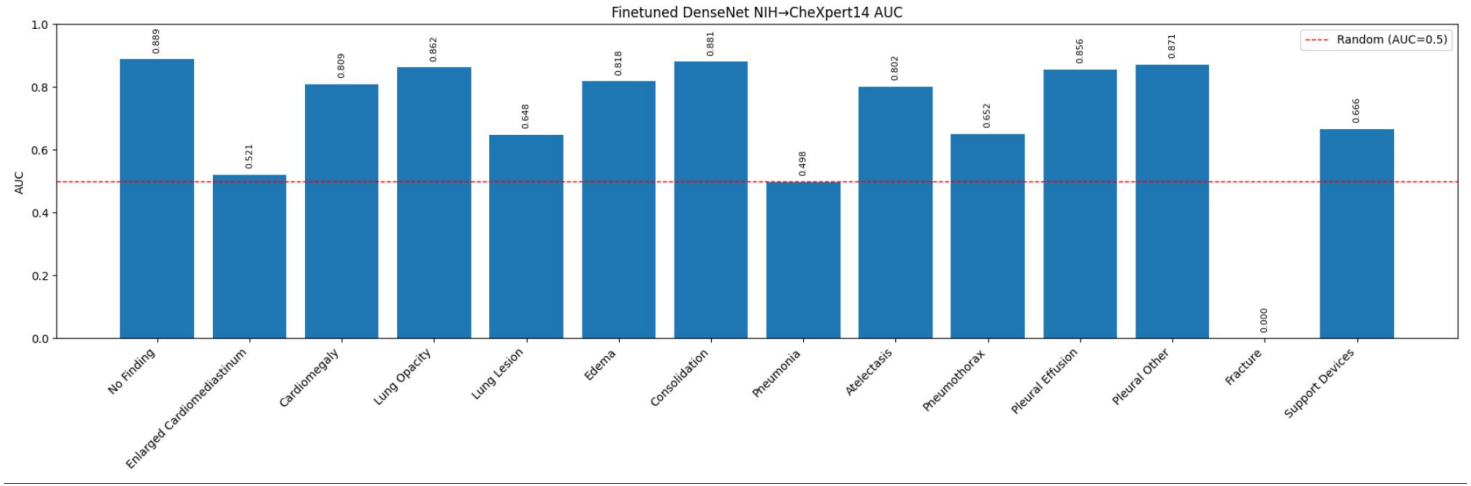
```python
def evaluate_14(model, loader, device):
    model.eval()
    all_probs, all_true = [], []
    with torch.no_grad():
        for imgs, labs in tqdm(loader, desc="Finetuned 14-label Eval"):
            imgs = imgs.to(device)
            labs = labs.to(device)
            probs = torch.sigmoid(model(imgs)).cpu().numpy()
            all_probs.append(probs)
            all_true.append(labs.cpu().numpy())
    return np.vstack(all_true), np.vstack(all_probs)
```

- Significant increase in AUROC after full finetuning
  - Makes sense that 5 overlapping classes are a little lower - signals difference in datasets

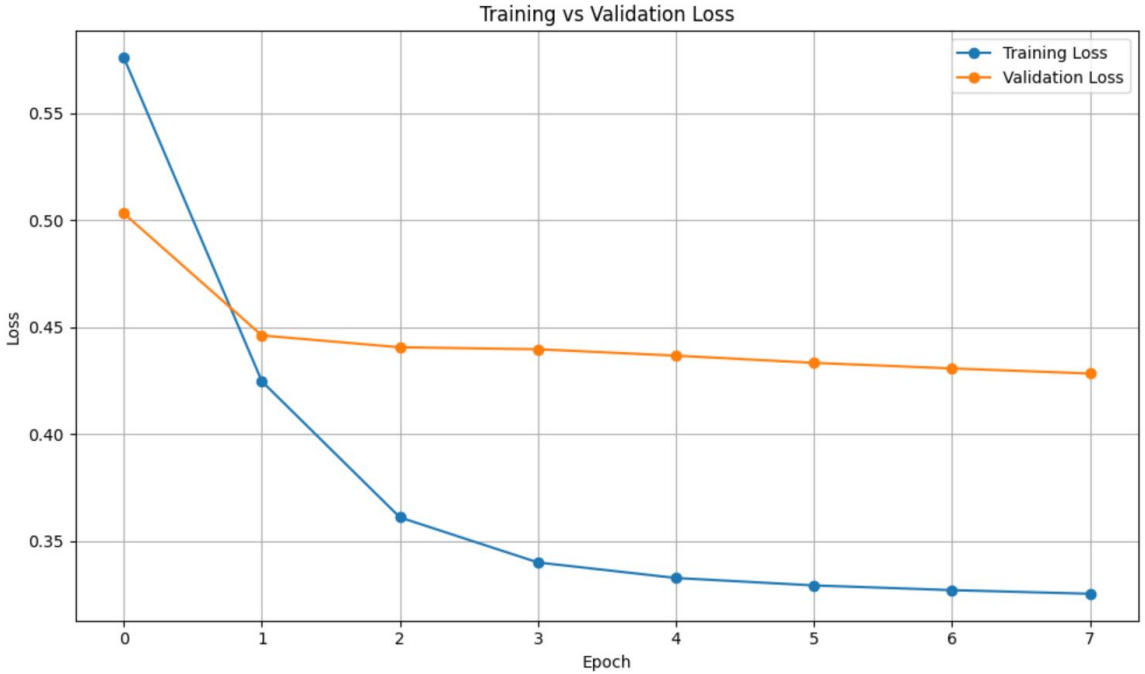


NIH Model: Finetuning Workflow

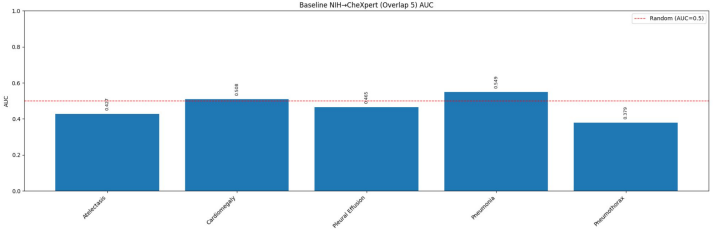
- Also finetuned for 8 epochs, still 32 batch size and 1e-6 learning rate

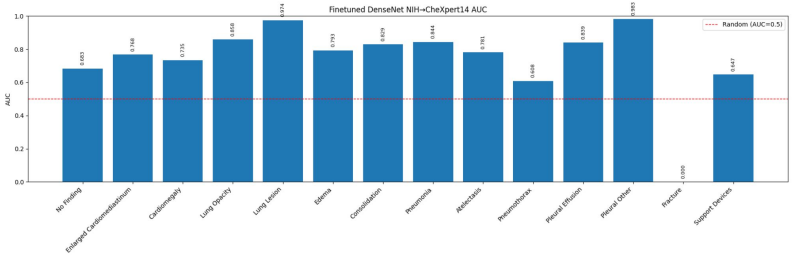


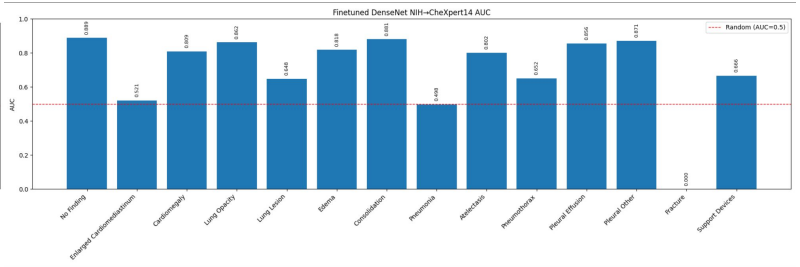
NIH Model: Finetuning Workflow



NIH Model: Side-by-Side







NIH Model: Grad-CAM & B-Box Explainer

- Then used Grad-CAM to generate heatmaps of specific X-ray image prediction evaluation cases, generated bounding boxes on the original X-ray overlaid with the smoothed heatmap, and integrated OpenAI ChatGPT-4o to give interpretation of bounding box selections/lack thereof
    - Displayed function hooks Grad-CAM to last convolutional layer of given model, producing 7x7 heatmap

```python
def compute_gradcam(model, img_t, target_idx, conv_layer, device, img_size=IMG_SIZE):
    acts = {}
    grads = {}
    def fwd_hook(m, i, o):
        acts["value"] = o
    def bwd_hook(m, gi, go):
        grads["value"] = go[0]
    h1 = conv_layer.register_forward_hook(fwd_hook)
    h2 = conv_layer.register_backward_hook(bwd_hook)
    img = img_t.unsqueeze(0).to(device)
    model.zero_grad()
    out = model(img)
    prob = torch.sigmoid(out)[0, target_idx]
    prob.backward()
    A = acts["value"]
    G = grads["value"]
    weights = G.mean(dim=(2, 3), keepdim=True)
    cam = (weights * A).sum(dim=1, keepdim=True)
    cam = F.relu(cam)
    cam_min = cam.min()
    cam_max = cam.max()
    if (cam_max - cam_min) > 1e-8:
        cam = (cam - cam_min) / (cam_max - cam_min)
    else:
        cam = torch.zeros_like(cam)
    cam_raw = cam[0, 0].detach().cpu().numpy()
    cam_up = F.interpolate(
        cam,
        size=(img_size, img_size),
        mode="bilinear",
        align_corners=False,
    )[0, 0].detach().cpu().numpy()
    h1.remove()
    h2.remove()
    return cam_up, cam_raw, float(prob.item())
```

NIH Model: Grad-CAM & B-Box Explainer

```
INTENSITY_V_THRESH = 0.15
INTENSITY_S_THRESH = 0.1
MIN_REGION_FRAC = 0.005

BASELINE_BOX_COLOR = "red"
FINETUNED_BOX_COLOR = "red"
BOX_WIDTH = 3
```

- Bounding box algorithm detects pixel saturation + value
- OpenAI API call directly attaches 2x2 panel PNG

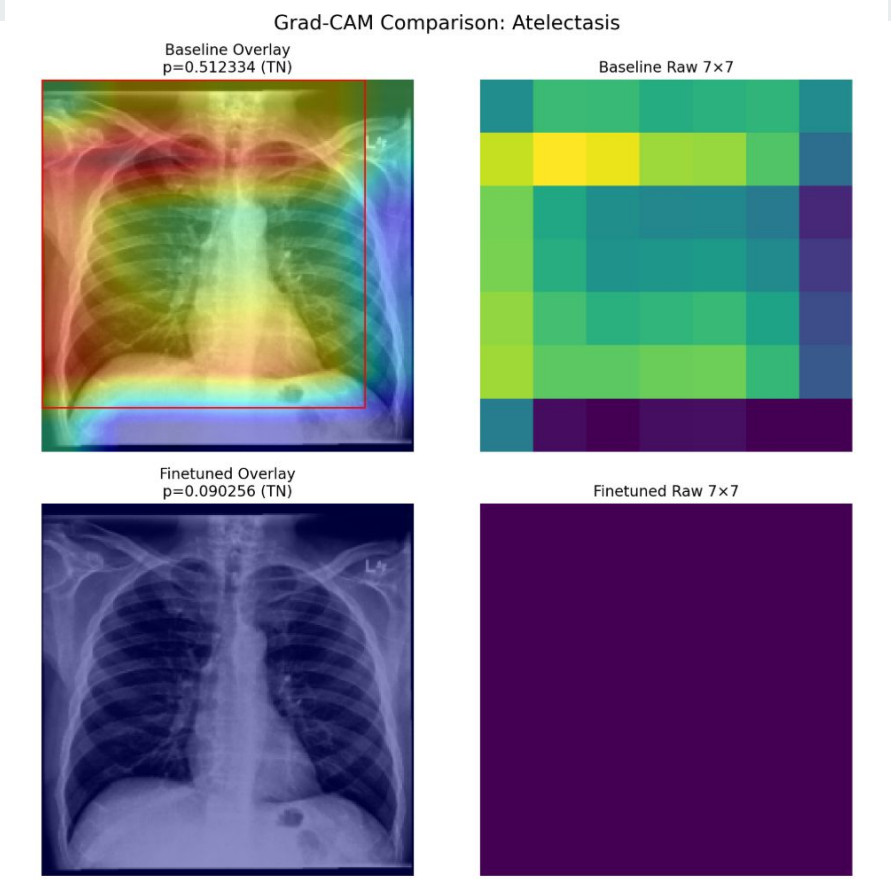
```
    prompt = f"""
Act as a professional radiologist and AI model-interpretability expert.

You are given a Grad-CAM panel showing:
- Top-left: Baseline model heatmap overlay
- Bottom-left: Full-parameter finetuned model heatmap overlay
- Bounding boxes drawn around detected high-activation regions.

Task:
1. State whether the boxed regions are actually relevant to the pathology classified: {class_name}.
2. Make a judgment on whether the bounding boxes reflect correct or incorrect reasoning behind the model's prediction.
3. If relevant and correct, explain what details in (or around) the boxed regions actually present in this specific x-ray case have led to the correct prediction (don't just say the bounding boxes are in correct location, tell what details are visually present in those locations indicative of the predicted pathology). If relevant and incorrect, explain why incorrect. If irrelevant, explain why the model may have gone with an irrelevant activation area.
"""
```

```
def detect_heat_boxes(overlay_np):
    img_float = overlay_np.astype(np.float32) / 255.0
    hsv = mcolors.rgb_to_hsv(img_float)
    Hc, S, V = hsv[..., 0], hsv[..., 1], hsv[..., 2]

    mask = (V > INTENSITY_V_THRESH) & (S > INTENSITY_S_THRESH)

    hot_mask = (Hc < 0.25) | (Hc > 0.95)
    mask &= hot_mask

    if mask.sum() == 0:
        return []

    labels, num = ndimage.label(mask)
    Hh, Wh = mask.shape
    min_pixels = int(MIN_REGION_FRAC * Hh * Wh)

    boxes = []
    for lab in range(1, num + 1):
        ys, xs = np.where(labels == lab)
        if ys.size == 0:
            continue
        y0, y1 = ys.min(), ys.max()
        x0, x1 = xs.min(), xs.max()
        area = (x1 - x0 + 1) * (y1 - y0 + 1)
        if area < min_pixels:
            continue
        boxes.append((int(x0), int(y0), int(x1), int(y1)))

    return boxes
```

NIH Model: Easy Case

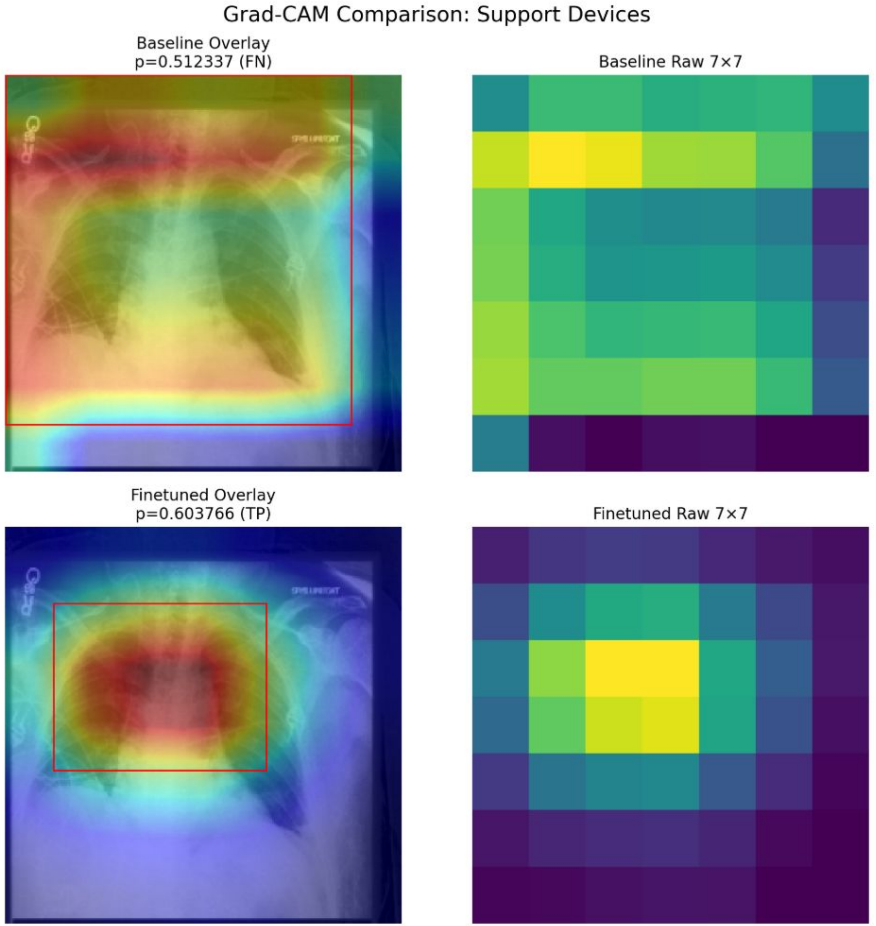
1. **Relevance of Boxed Regions to Atelectasis:**
  - **Baseline Model:**
    The boxed region in the baseline model's heatmap overlay appears to cover the upper lung fields and part of the mediastinum. These areas are not typically where atelectasis is primarily identified, which usually involves volume loss and increased density in the affected lung area, often seen in the lower lobes or specific segments.
  - **Finetuned Model:**
    The finetuned model shows no significant activation, suggesting it does not identify any relevant features indicative of atelectasis.
2. **Judgment on Bounding Boxes:**
  - **Baseline Model:**
    The bounding box reflects incorrect reasoning. The highlighted areas do not correspond to typical radiographic signs of atelectasis, such as increased opacity, displacement of interlobar fissures, or elevation of the diaphragm.
  - **Finetuned Model:**
    The lack of activation suggests that the model does not find relevant features for atelectasis, which aligns with the true negative (TN) prediction.
3. **Explanation:**
  - **Baseline Model:**
    The incorrect activation in the baseline model may be due to the model focusing on high-contrast areas or edges rather than specific pathological signs. This can happen if the model is not well-trained to distinguish between normal anatomical variations and pathological changes.
  - **Finetuned Model:**
    The finetuned model's lack of activation indicates it correctly does not identify features of atelectasis, which is consistent with the true negative prediction. This suggests improved specificity and a better understanding of the relevant features for atelectasis, avoiding false positives.

In summary, the baseline model incorrectly highlights irrelevant regions, while the finetuned model correctly shows no significant activation, aligning with the absence of atelectasis in this case.



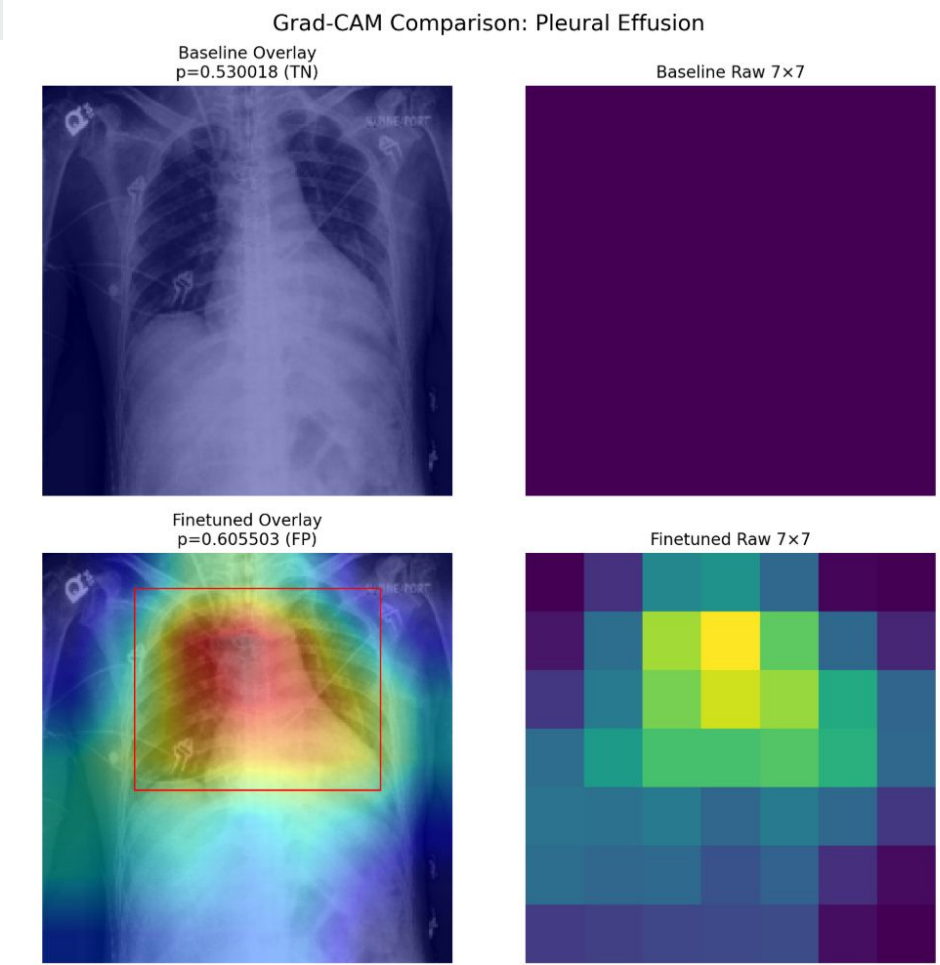
NIH Model: Average Case

1. **Relevance of Boxed Regions:**
   - **Baseline Model (Top-Left):** The boxed region is not relevant to the pathology of "Support Devices." The activation is spread across a large area, including regions that are unlikely to contain support devices, such as the upper chest and shoulders.
   - **Finetuned Model (Bottom-Left):** The boxed region is relevant. The activation is focused on the central thoracic area, where support devices such as central lines or endotracheal tubes are typically located.
2. **Judgment on Bounding Boxes:**
   - **Baseline Model:** The bounding box reflects incorrect reasoning. The model's prediction seems to be influenced by irrelevant areas, possibly due to noise or artifacts in the image.
   - **Finetuned Model:** The bounding box reflects correct reasoning. The model is focusing on the appropriate region where support devices are commonly found.
3. **Details in Boxed Regions:**
   - **Baseline Model (Incorrect):** The model might have been misled by the overall brightness or contrast in the image rather than specific features indicative of support devices.
   - **Finetuned Model (Correct):** In the finetuned model, the activation is centered on the mediastinal area, which is where you would expect to see support devices like a central venous catheter or an endotracheal tube. The model likely identified linear structures or radiopaque lines that are characteristic of these devices, leading to a correct prediction.



NIH Model: Difficult Case

1. **Relevance of Boxed Regions:**
  - The boxed regions in the finetuned model's heatmap overlay (bottom-left) are not relevant to the pathology of Pleural Effusion. Pleural effusion typically presents as a blunting of the costophrenic angles, a meniscus sign, or fluid layering on the lateral chest wall, usually in the lower zones of the chest X-ray.
2. **Judgment on Bounding Boxes:**
  - The bounding boxes reflect incorrect reasoning behind the model's prediction. The activation is centered in the mid and upper thoracic regions rather than the lower zones where pleural effusion signs are typically observed.
3. **Explanation:**
  - **Incorrect and Irrelevant Activation:**
    - The finetuned model is focusing on the central thoracic area, which is not typically where pleural effusions manifest. This could be due to the model picking up on irrelevant features such as normal anatomical structures or artifacts unrelated to pleural effusion.
    - The model may have been misled by patterns or textures in the lung fields or mediastinum that are not indicative of effusion but might have statistical correlation in the training data.
In summary, the model's attention is misplaced, and it does not correctly identify the areas indicative of pleural effusion, leading to a false positive prediction.



Model 3 - clip-vit-base-patch32

- Standard OpenAI Contrastive Language-Image Processing Vision Transformer model
- Around 88 million parameters
- Entirely non-medical image encoder, not pre-trained on any radiographs
- Relies on text embedding (label) probabilities rather than target image probabilities

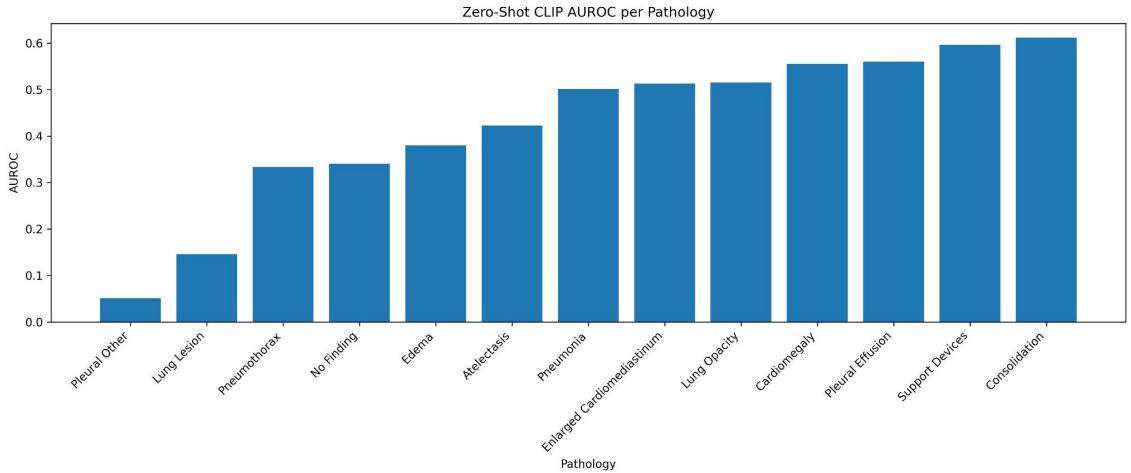
Model 3 - clip-vit-base-patch32 fine-tune workflow

- Run baseline inference using CheXpert validation set

```python
def calculate_auroc(predictions, labels):
    aurocs = {}
    for i, name in enumerate(CHEXPERT_LABELS):
        if len(np.unique(labels[:, i])) > 1:
            try:
                aurocs[name] = roc_auc_score(labels[:, i], predictions[:, i])
            except Exception:
                aurocs[name] = np.nan
        else:
            aurocs[name] = np.nan
    valid = [v for v in aurocs.values() if not np.isnan(v)]
    mean_auroc = np.mean(valid) if valid else np.nan
    return aurocs, mean_auroc
```

```python
def compute_zero_shot_predictions(model, processor, dataloader, device):
    """Compute zero-shot predictions across the dataloader.
       Returns (predictions, labels) where predictions shape=(N, num_labels)."""
    model.eval()
    text_emb = prepare_text_embeddings(model, processor, device)
    num_templates = len(TEMPLATES)
    num_texts = text_emb.shape[0]
    assert num_texts == len(CHEXPERT_LABELS) * num_templates

    all_preds = []
    all_labels = []

    with torch.no_grad():
        for images, labels in tqdm(dataloader):
            image_inputs = processor(images=images, return_tensors="pt").to(device)
            if hasattr(model, "get_image_features"):
                image_emb = model.get_image_features(**{"pixel_values": image_inputs["pixel_values"]})
            else:
                out = model.get_image_features(**{"pixel_values": image_inputs["pixel_values"]})
                image_emb = out

            image_emb = image_emb / image_emb.norm(dim=-1, keepdim=True)

            similarity = image_emb @ text_emb.T

            logits_per_label = []
            for i in range(len(CHEXPERT_LABELS)):
                start = i * num_templates
                end = (i + 1) * num_templates

                group = similarity[:, start:end]
                score = group.mean(dim=1)
                logits_per_label.append(score)
            logits_per_label = torch.stack(logits_per_label, dim=1)

            mins = logits_per_label.min(dim=1, keepdim=True).values
            maxs = logits_per_label.max(dim=1, keepdim=True).values
            probs = (logits_per_label - mins) / (maxs - mins + 1e-8)

            all_preds.append(probs.cpu().numpy())
            all_labels.append(labels.cpu().numpy())

    predictions = np.vstack(all_preds)
    labels = np.vstack(all_labels)
    return predictions, labels
```

Model 3 - Baseline Inference Results



Model 3 - clip-vit-base-patch32 fine-tune workflow

- Utilize LoRA to finetune the model

```
BATCH_SIZE = 32
LR = 1e-6
EPOCHS = 5
```

```python
def train_lora():
    print("Loading model...")
    model = CLIPModel.from_pretrained(MODEL_NAME, cache_dir=CACHE_DIR)

    for p in model.parameters():
        p.requires_grad = False

    lora_config = LoraConfig(
        r=16,
        lora_alpha=32,
        target_modules=["q_proj", "k_proj", "v_proj", "out_proj"],
        lora_dropout=0.1,
        bias="none",
    )
```

Model 3 - clip-vit-base-patch32 fine-tune workflow

Within same training loop

- Compute validation loss
- Compute AUROC

```python
def calculate_auroc(predictions, labels):
    aurocs = {}
    for i, name in enumerate(CHEXPERT_LABELS):
        if len(np.unique(labels[:, i])) > 1:
            try:
                aurocs[name] = roc_auc_score(labels[:, i], predictions[:, i])
            except:
                aurocs[name] = np.nan
        else:
            aurocs[name] = np.nan
    valid = [v for v in aurocs.values() if not np.isnan(v)]
    mean_auroc = np.mean(valid) if valid else np.nan
    return aurocs, mean_auroc
```

```python
model.eval()
val_loss = 0
val_preds = []
val_labels = []

with torch.no_grad():
    for images, labels in val_dl:
        labels = labels.to(DEVICE)
        img_tok = processor(images=images, return_tensors="pt").to(DEVICE)
        img_emb = model.get_image_features(**{"pixel_values": img_tok["pixel_values"]})
        img_emb = img_emb / img_emb.norm(dim=-1, keepdim=True)

        sim = scale * (img_emb @ text_emb.T)
        logits = []
        for i in range(len(CHEXPERT_LABELS)):
            start = i * num_templates
            end = (i + 1) * num_templates
            logits.append(sim[:, start:end].mean(dim=1))
        logits = torch.stack(logits, dim=1)

        loss = loss_fn(logits, labels)
        val_loss += loss.item()

        val_preds.append(logits.cpu().numpy())
        val_labels.append(labels.cpu().numpy())

avg_val_loss = val_loss / len(val_dl)
val_loss_history.append(avg_val_loss)

preds = np.vstack(val_preds)
lbls = np.vstack(val_labels)
aurocs, mean_auroc = calculate_auroc(preds, lbls)
epoch_aurocs[epoch + 1] = aurocs

print(f"Epoch {epoch+1}/{EPOCHS} - Train Loss: {avg_train_loss:.4f}  "
      f"Val Loss: {avg_val_loss:.4f}  Mean AUROC: {mean_auroc:.4f}")
```

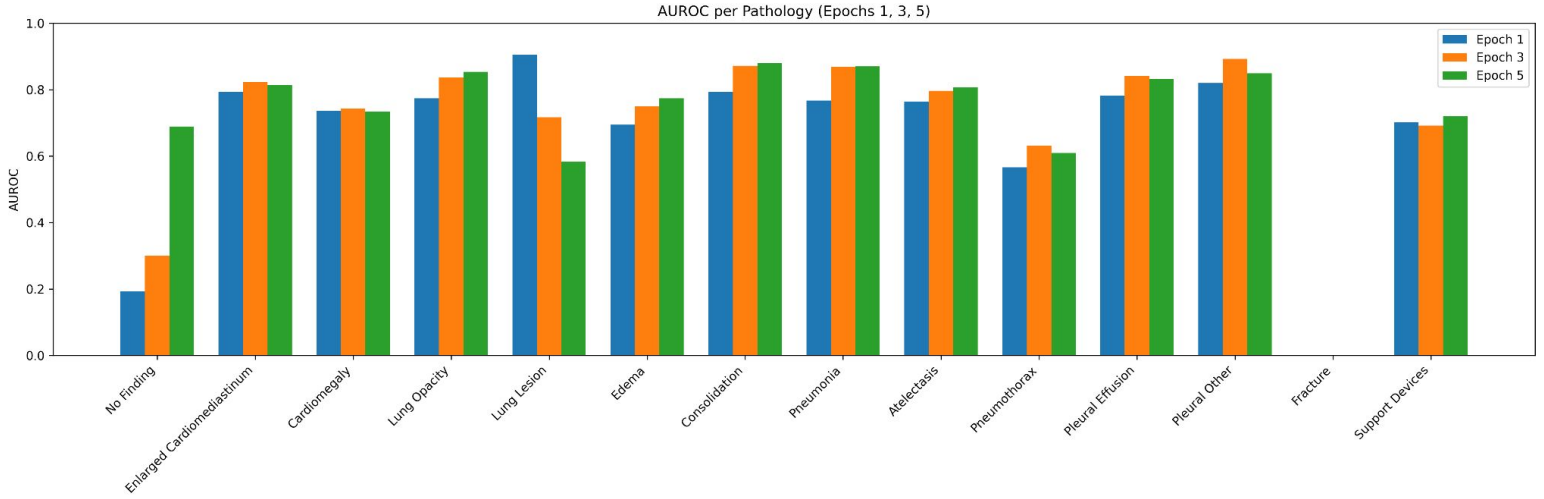
Model 3 - clip-vit-base-patch32 fine-tune workflow

Training loop

- Compute similarity with text embeddings
- Aggregate logits
- Compute loss and backward
- Record statistics

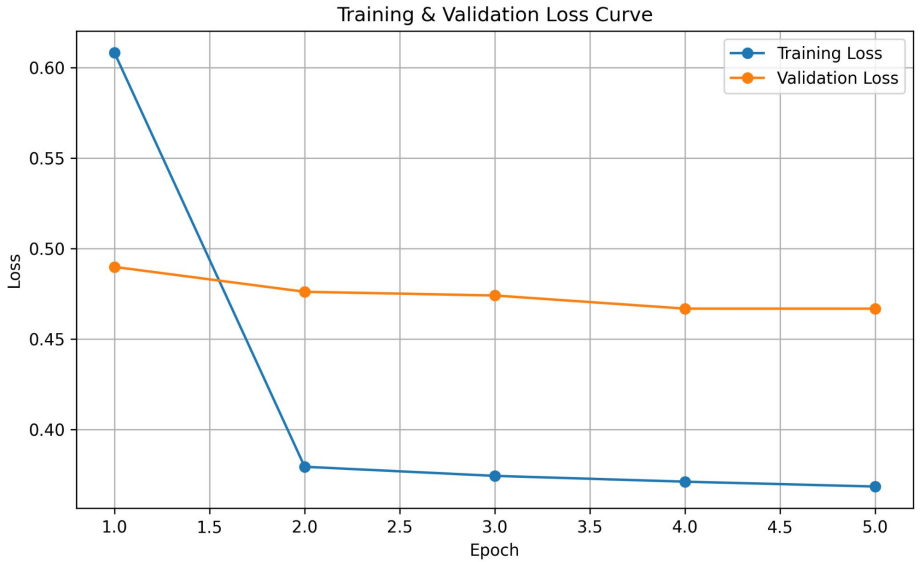
```
for epoch in range(EPOCHS):
    model.train()
    total_loss = 0
    all_preds = []
    all_labels = []

    for images, labels in tqdm(train_dl):
        labels = labels.to(DEVICE)
        img_tok = processor(images=images, return_tensors="pt").to(DEVICE)
        img_emb = model.get_image_features(**{"pixel_values": img_tok["pixel_values"]})
        img_emb = img_emb / img_emb.norm(dim=-1, keepdim=True)

        sim = scale * (img_emb @ text_emb.T)
        logits = []
        for i in range(len(CHEXPERT_LABELS)):
            start = i * num_templates
            end = (i + 1) * num_templates
            logits.append(sim[:, start:end].mean(dim=1))
        logits = torch.stack(logits, dim=1)

        loss = loss_fn(logits, labels)
        optim.zero_grad()
        loss.backward()
        optim.step()

        total_loss += loss.item()
        all_preds.append(logits.detach().cpu().numpy())
        all_labels.append(labels.detach().cpu().numpy())

    avg_train_loss = total_loss / len(train_dl)
    train_loss_history.append(avg_train_loss)
```

Model Three Fine-tuned Results

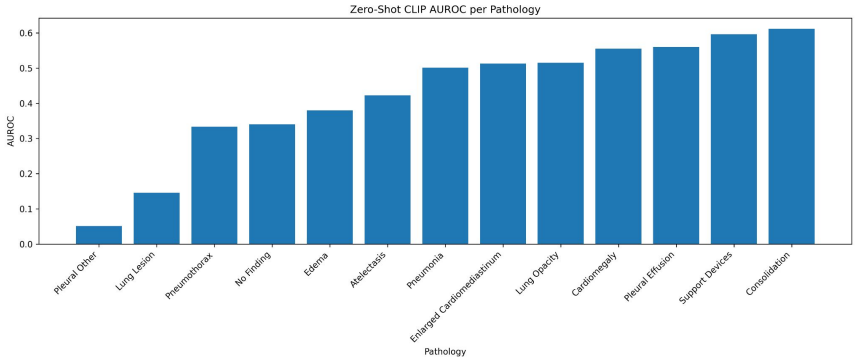


Model Three Fine-tuned Results

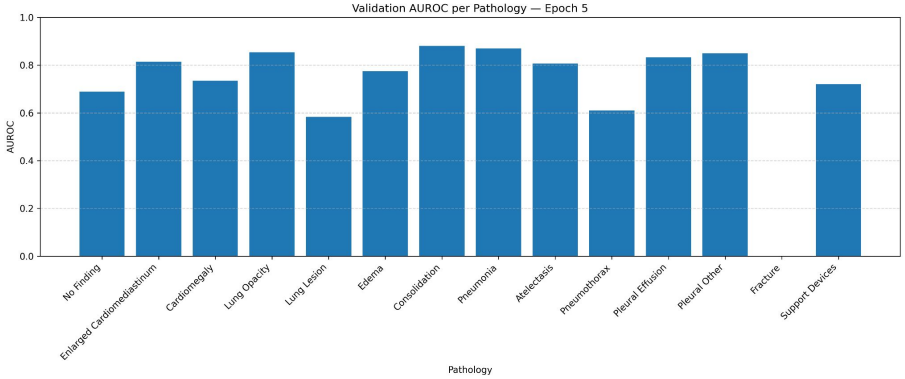
- Good at training data
- Small impact on validation data



Model Three Fine-tuned Results



Mean AUROC: ~0.423



Mean AUROC: ~0.771

Model Three Fine-tuned Workflow: interpretability

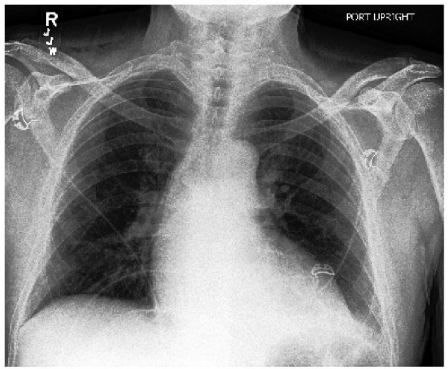
- Generate heatmaps to help display what the model “looked” at to make its classifications
- Integrate OpenAI to extend the explainability with natural language

```
    prompt = """You are an expert radiologist analyzing chest X-ray interpretations and AI model behavior.
This image shows three panels:
1. LEFT: Original chest X-ray with ground truth labels
2. MIDDLE: Baseline CLIP model's heatmap and predictions
3. RIGHT: Fine-tuned model's heatmap and predictions

Please provide a detailed clinical and technical analysis:

**Fine-tuned Model Analysis**:
- Compare predictions to actual radiographic findings
- For CORRECT predictions: What clinically relevant features is the model focusing on?
- For INCORRECT predictions: What is the model misinterpreting and why?
- Evaluate if the attention pattern demonstrates appropriate clinical reasoning
- Compare attention strategy to baseline: Is it more anatomically informed?

Focus on explaining the "WHY" behind both the radiographic appearances and the model behaviors, connecting visual features to clinical knowledge."""
```

```
def generate(self, image_tensor, text_tensor):
    image_tensor = image_tensor.unsqueeze(0).cuda()
    image_tensor.requires_grad = True

    with torch.enable_grad():
        outputs = self.model(pixel_values=image_tensor, input_ids=text_tensor)
        logits = outputs.logits_per_image

        score = logits[0, 0]

        self.model.zero_grad()
        score.backward(retain_graph=True)

    gradients = self.gradients
    activations = self.activations

    weights = gradients.mean(dim=1, keepdim=True)
    cam = torch.relu(torch.matmul(activations, weights.transpose(-1, -2)))
    cam = cam.squeeze(-1).squeeze(0)

    cam = cam[1:]  # remove CLS token
    h = w = int(cam.shape[0] ** 0.5)
    cam = cam.reshape(h, w)

    cam -= cam.min()
    cam /= cam.max() + 1e-6
    return cam.detach().cpu()
```

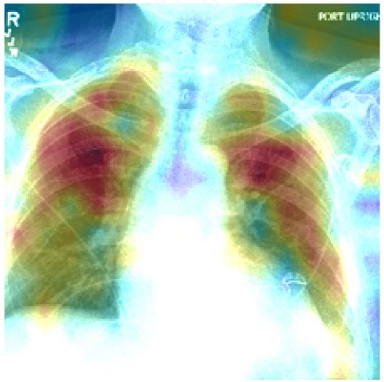
Model Three Easy Case

Ground Truth:
• Enlarged Cardiomediastinum
• Cardiomegaly
• Lung Opacity
• Atelectasis



Baseline Model

Top Predictions:
• Cardiomegaly (31.506)
• Pneumothorax (31.295)
• Lung Opacity (30.786)



Finetuned Model

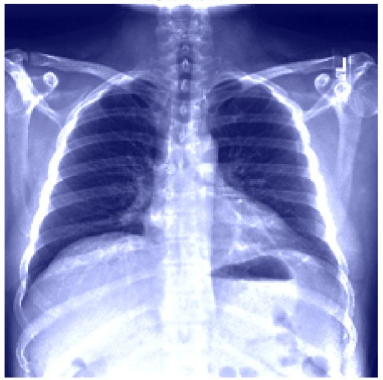
Top Predictions:
• Lung Opacity (15.456)
• Support Devices (13.105)
• No Finding (5.500)

Model Three Medium Case

Ground Truth:
• No Finding

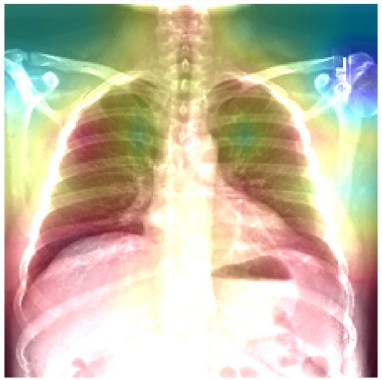
Baseline Model

Top Predictions:
• Pleural Effusion (32.284)
• Pneumothorax (31.921)
• Lung Opacity (31.578)



Finetuned Model

Top Predictions:
• No Finding (12.589)
• Lung Opacity (10.983)
• Support Devices (9.429)



Model Three Hard Case

Ground Truth:
• No Finding

Baseline Model

Top Predictions:
• Pleural Effusion (32.547)
• Lung Opacity (31.980)
• Pneumothorax (31.755)

Finetuned Model

Top Predictions:
• Lung Opacity (15.224)
• Support Devices (10.329)
• No Finding (9.599)

```
CLASSIFICATIONS = [
    'Atelectasis', 'Cardiomegaly', 'Consolidation', 'Edema', 'Enlarged Cardiomediastinum', 'Lung Lesion',
    'Lung Opacity', 'Pneumonia', 'Pneumothorax', 'Pleural Effusion', 'Pleural Other'
]
```

Model 4- EfficientNet-B0 Setup

```
def process_chexpert_labels(df, class_names):
    print("Processing labels...")
    processed = df.copy()
    # convert NaN to 0
    processed[class_names] = processed[class_names].fillna(0)
    # convert -1 to 1
    processed[class_names] = processed[class_names].replace(-1, 1)
    print("Label processing complete.")
    return processed

# define Image Transformations, EfficientNet-B0 expects 224x224 imagesm
size = 224
val_transform = transforms.Compose([
    transforms.Resize((size, size)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])
```

- Pretrained on imagenet dataset.
  - ILSVRC 2012 version with 1.2 million images and 1,000 classes
- Run baseline inference using CheXpert validation set

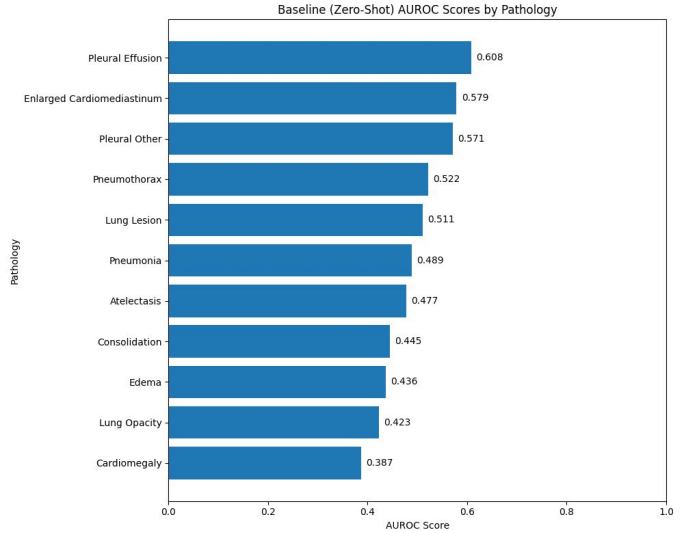
```
# create custom pytorch dataset
class CheXpertValidationDataset(Dataset):
    def __init__(self, df, image_root_dir, class_names, transform):
        self.df = df
        self.image_root_dir = image_root_dir
        self.class_names = class_names
        self.transform = transform

    def __len__(self):
        return len(self.df)

    def __getitem__(self, idx):
        row = self.df.iloc[idx]
        image = Image.open(os.path.join(self.image_root_dir, row['Path'])).convert("RGB")
        labels = torch.tensor(row[self.class_names].values.astype(np.float32))
        if self.transform:
            image = self.transform(image)
        return image, labels
```

```
# load the CSV
try:
    val_raw = panda.read_csv(VAL_CSV)
except FileNotFoundError:
    print(f"Error: did not find valid.csv file")
# need to process the labels
val_processed = process_chexpert_labels(val_raw, CLASSIFICATIONS)
# create the dataset
val_dataset = CheXpertValidationDataset(
    df=val_processed,
    image_root_dir=ROOT_DIR,
    class_names=CLASSIFICATIONS,
    transform=val_transform
)
# create the dataLoader
val_loader = DataLoader(
    val_dataset,
    batch_size=BATCH_SIZE,
    shuffle=False,
    num_workers=2
)
```

Model 4- Baseline

```python
# import from the chexpert_loader.py
try:
    from chexpert_loader import val_loader, NUM_CLASSES, CLASSIFICATIONS
    print("Imported 'val_loader' and 'NUM_CLASSES' from chexpert_loader.py")
    if NUM_CLASSES != 12 or len(CLASSIFICATIONS) != 12:
        print(f"Warning: Expected 12 classes, but found {NUM_CLASSES} and {len(CLASSIFICATIONS)} labels.")
    print(f"Found {NUM_CLASSES} classes: {CLASSIFICATIONS}")
except ImportError:
    print("could not import from 'chexpert_loader.py'.")
    exit()
except Exception as e:
    print(f"An error occurred during import: {e}")
    exit()

#seting up the 12-Class Model
device = "cuda" if torch.cuda.is_available() else "cpu"
model = models.efficientnet_b0(weights=models.EfficientNet_B0_Weights.IMAGENET1K_V1)
feature_size = model.classifier[1].in_features
model.classifier[1] = nn.Linear(in_features=feature_size, out_features=NUM_CLASSES)
print(f"Model classifier replaced. Output features: {NUM_CLASSES}")
model.to(device)

# setting up the auroc metric
auroc = MultilabelAUROC(
    num_labels=NUM_CLASSES,
    average="none"
).to(device)
```

```python
# do the the evaluation
print("\nStarting baseline evaluation")
model.eval()
with torch.no_grad():
    for img, labels in tqdm(val_loader, desc="Evaluating Baseline"):
        img = img.to(device)
        labels = labels.to(device)
        auroc.update(torch.sigmoid(model(img.to(device))), labels.int())

# the final scores per class
scores_np = auroc.compute().cpu().numpy()
avg_auroc = np.mean(scores_np)

print("\nBaseline Results")
print(f"Overall AUROC: {avg_auroc:.4f}")
print("Individual Scores by Pathology:")
```



Model 4 - add explainability to heatmap

```
prompt = f"""
You are an expert Radiologist and AI Researcher evaluating a deep learning experiment.
You are comparing two models trained to detect pathologies in Chest X-rays:
1.Baseline Model: Pre-trained on ImageNet (generic images).
2.Finetuned Model: The same architecture finetuned on Chest X-rays using LoRA.
Please analyze the attached comparison image for the pathology: **{pathology_name}**.

Image Layout:
- Left: Original X-ray with Ground Truth labels.
- Middle: Baseline Model Heatmap (Grad-CAM) + Top 3 Predictions.
- Right: Finetuned Model Heatmap (Grad-CAM) + Top 3 Predictions.

Please provide a structured report in the following format:

1. Image Quality & Ground Truth:
- Is the X-ray image clear enough for diagnosis?
- Does the Ground Truth label confirm the presence of {pathology_name}?

2. Baseline Model Analysis:
- Heatmap Focus: Where is the model looking? Is it focused on relevant anatomy (lungs/heart) or background artifacts (text, bones, empty space)?
- Prediction: Does {pathology_name} appear in its Top 3 predictions?

3. Finetuned Model Analysis:
- Heatmap Focus: Has the focus shifted? Does the heatmap now align with the *radiological indicators* of {pathology_name}? (e.g., lung opacities for Pneumonia, pleural edge for Pneumothorax).
- Prediction: Is {pathology_name} in the Top 3? Did the confidence score increase compared to the Baseline?

4. Final Assessment:
- Verdict: [IMPROVED / WORSE / NO CHANGE]
- Summary: Briefly explain why based on the visual evidence. Did the model learn the correct features, or is it hallucinating?
"""
```

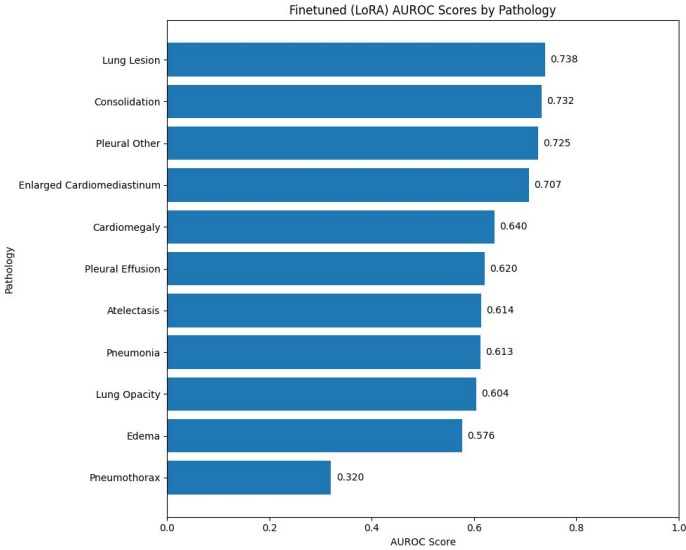
```
BATCH_SIZE = 32
NUM_CLASSES = 11
LEARNING_RATE = 1e-5
NUM_EPOCHS = 5
WORKER_COUNT = 1
```

```
# create the dataloaders
train_dataset = CheXpertDataset(process_chexpert_labels(panda.read_csv(TRAIN_CSV_PATH), CLASS_NAMES),
                                IMAGE_ROOT_DIR, CLASS_NAMES, train_transform)
train_loader = DataLoader(
    train_dataset,
    batch_size=BATCH_SIZE,
    shuffle=True,
    num_workers=WORKER_COUNT
)

# create the validation dataloader
val_dataset = CheXpertDataset(process_chexpert_labels(panda.read_csv(VAL_CSV_PATH), CLASS_NAMES), IMAGE_ROOT_DIR, CLASS_NAMES, val_transform)
val_loader = DataLoader(
    val_dataset,
    batch_size=BATCH_SIZE,
    shuffle=False,
    num_workers=WORKER_COUNT
)

print(f"Train loader: {len(train_loader)} batches")
print(f"Valid loader: {len(val_loader)} batches")
```

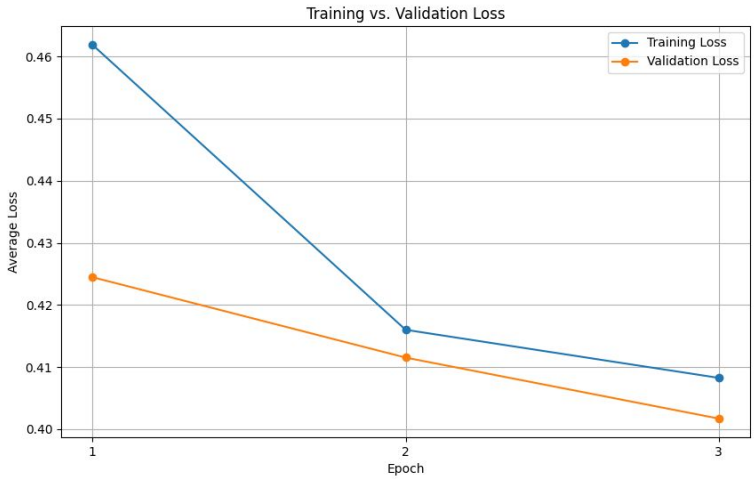
Model 4- Fine-tuned (lora) + result

```
for epoch in range(NUM_EPOCHS):
    print(f"\nEpoch {epoch + 1}/{NUM_EPOCHS}")
    # start training phase
    lora_model.train()
    running_loss_30_batch = 0.0
    start_time_30_batch = time.time()
    train_loss = 0.0

    for i, (images, labels) in enumerate(tqdm(train_loader, desc="Training")):
        images = images.to(device)
        labels = labels.to(device)

        optimizer.zero_grad()
        outputs = lora_model(images)
        loss = criterion(outputs, labels)

        loss.backward()
        optimizer.step()

        curr_loss = loss.item()
        train_loss += curr_loss

        running_loss_30_batch += curr_loss
        all_batch_losses.append(curr_loss)

        if (i + 1) % 30 == 0:
            end_time_30_batch = time.time()
            duration = end_time_30_batch - start_time_30_batch
            avg_loss_30_batch = running_loss_30_batch / 30

            print(f"[Epoch {epoch + 1}, Batch {i + 1}]")
            print(f"Avg Loss (last 30 batches): {avg_loss_30_batch:.4f}")
            print(f"Time (last 30 batches): {duration:.2f} seconds")

            running_loss_30_batch = 0.0
            start_time_30_batch = time.time()

    avg_epoch_train_loss = train_loss / len(train_loader)
    all_epoch_train_losses.append(avg_epoch_train_loss)
    print(f"Epoch {epoch+1} Average Training Loss: {avg_epoch_train_loss:.4f}")
```

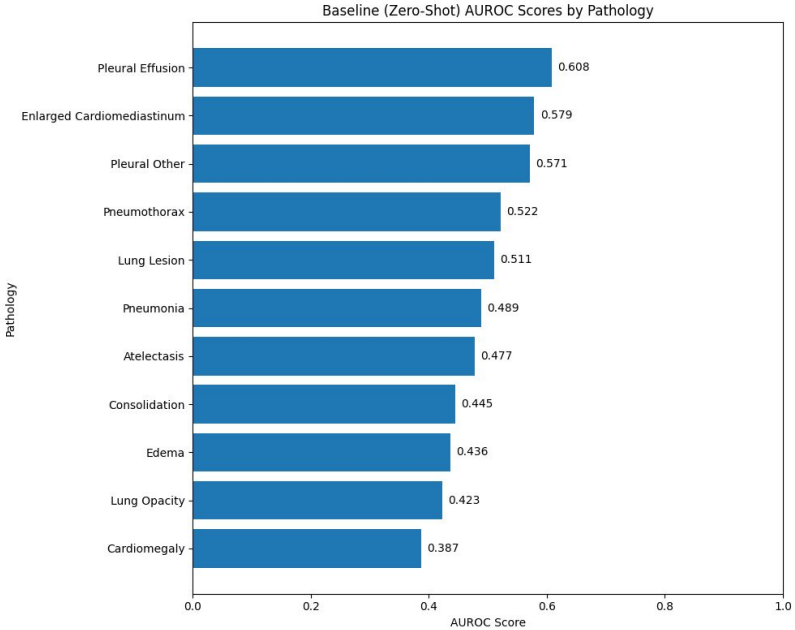
```
# setting up the model and applying LoRA
print("Setting up model and applying LoRA...")
device = "cuda" if torch.cuda.is_available() else "cpu"
model = models.efficientnet_b0(weights=models.EfficientNet_B0_Weights.IMAGENET1K_V1)
model.classifier[1] = nn.Linear(in_features=model.classifier[1].in_features, out_features=NUM_CLASSES)
target = []
for name, module in model.named_modules():
    if isinstance(module, nn.Conv2d):
        if module.groups == 1:
            target.append(name)
target.append("classifier.1")
print(f"found {len(target)} modules to target with LoRA.")
lora_config = LoraConfig(
    r=8,
    lora_alpha=16,
    target_modules=target,
    lora_dropout=0.05,
    bias="none",
)

lora_model = get_peft_model(model, lora_config)
lora_model.print_trainable_parameters()
lora_model.to(device)
```

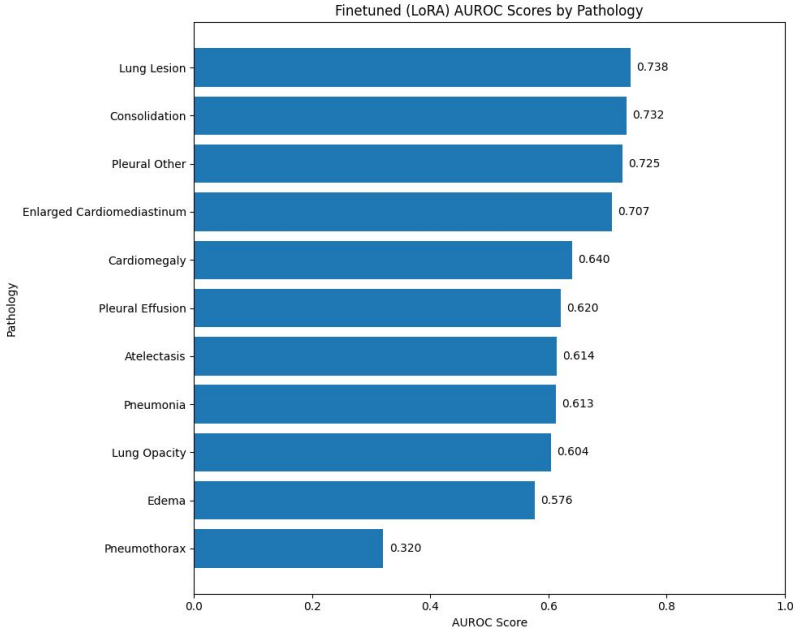


Model 4 - Fine-tuned Results (3, 5 epochs)



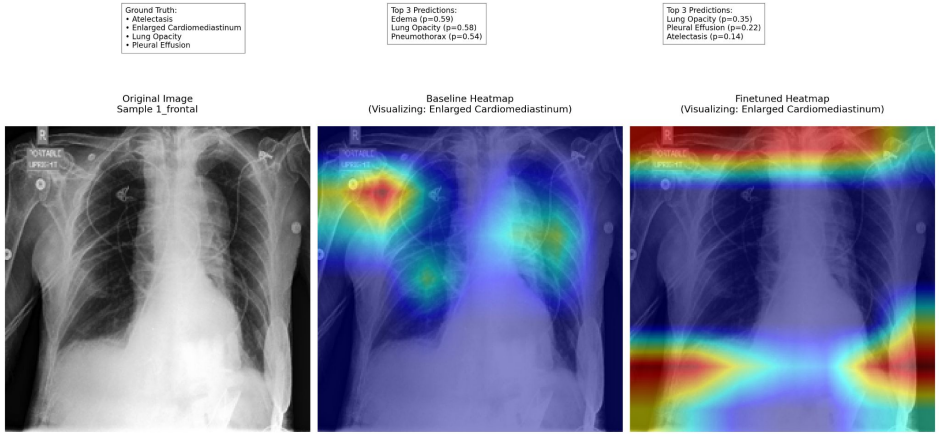
Model-4 - Fine-tuned Results Comparison





Model 4 - Easy Case

- Verdict: IMPROVED
- The finetuned Model shows a notable improvement in focusing on relevant anatomical features associated with the recorded pathology, indicating that it is learning the appropriate characteristics related to Enlarged Cardiomediastinum. Although it still does not predict this specific label, the adjustment in heatmap focus suggests enhanced feature recognition compared to the Baseline Model. The model may require further training or data to fully capture and predict all relevant pathologies present in the X-ray.

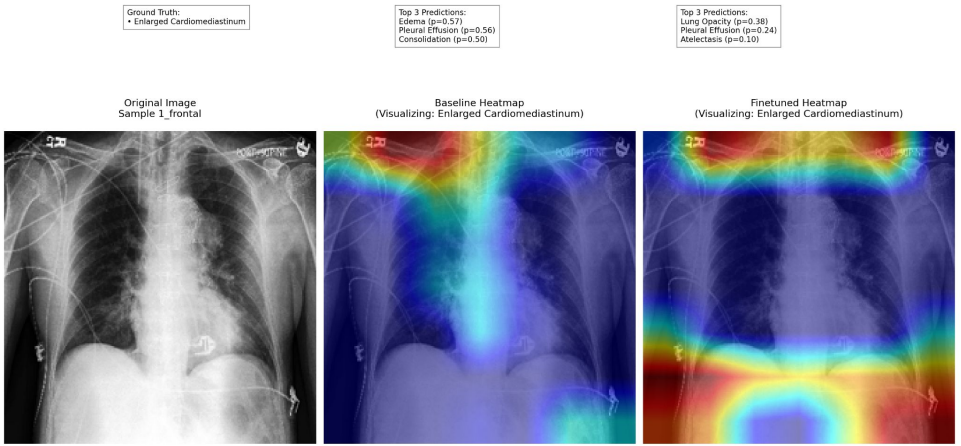


Model 4 - Medium Case

- Verdict:IMPROVED
- Summary: The finetuned model demonstrated an enhanced ability to recognize features associated with Pleural Effusion in the heatmap analysis. Although the confidence for Pleural Effusion in the Finetuned Model was relatively low, the improved anatomical focus compared to the Baseline indicates that the model has learned better representations pertinent to the pathology. This suggests that the additional training on Chest X-rays using LoRA has facilitated the model to capture relevant features more effectively.

Model 4 - Hard Case

- Verdict: NO CHANGE
- Summary: The finetuned model has improved its anatomical focus significantly, targeting areas relevant to Enlarged Cardiomediastinum compared to the Baseline Model. However, it still fails to predict the pathology accurately and confidently. This suggests that while the model has learned to target better features, it may still struggle with the distinction between various pathologies, indicating that further adjustments in training data or model architecture may be necessary to enhance diagnostic accuracy.



Finetuned Model LoRA Configs

	r	α	Dropout
Model 1	16	32	0.1
Model 3	16	32	0.1
Model 4	8	16	0.05

* Model 2 utilized full-parameter finetuning

Finetuned Model Performance Analysis

	Mean AUROC	Batch Size,Epochs	~Train Time
Model 1	0.753	32,10	~ 5 hours
Model 2	0.735	32,3/8	50 min (3e) / 2 hr 30 min (8e)
Model 3	0.771	32,5	1 hr 45 min
Model 4	0.626	32, 5	~2 hours

Heatmap Case Overview

	Easy	Medium	Hard
Model 1	Patient#64549	Patient#64579	Patient#64601
Model 2	Patient#64542	Patient#64546	Patient#64687
Model 3	Patient#64569	Patient#64556	Patient#64575
Model 4	Patient#64740	Patient#64720	Patient#64726

Cases categorized as Easy-Both are correct, Medium-Only one is correct, Hard-Both are incorrect for that model

Model 1 Case Analysis

	Easy	Medium	Hard
Model 2	Finetuned	Finetuned	Both
Model 3	Baseline	Finetuned	Finetuned
Model 4	Finetuned	Finetuned	Finetuned

Model 1 shows improved performance in almost all cases across other three models. Its baseline outperforms in the Model 3 easy case, which contains ground truth Atelectasis, Cardiomegaly, Enlarged Cardiomediastinum, and Lung Opacity. The Baseline can identify Atelectasis because it maintained focus on the lower lung fields and retained high confidence in specific pathologies. The Finetuned model performed worse on Atelectasis because it got distracted by noise in the neck area (the Support Devices artifact) and shifted its probability weight toward the broader Lung Opacity label rather than the specific subtype.

Model 2 Case Analysis

	Easy	Medium	Hard
Model 1	Finetuned	Finetuned	Neither
Model 3	Finetuned	Finetuned	Neither
Model 4	Finetuned	Finetuned	Finetuned

Model 2 shows improved performance in almost all cases across other three models, especially compared to baseline. The baseline never performs well because the original baseline NIH model was trained on the NIH Chest X-Ray 14 dataset, which has a different domain and classification methodology compared to CheXpert. Model 1 Hard case ground truth contains Edema and Lung Opacity, which the finetuned NIH model has had trouble with, especially Lung Opacity and detecting different shapes of fluid buildup. Model 3 Hard case ground truth contains ground truth of No Finding, but the finetuned NIH model seems to be having trouble with the seemingly irregular shapes/patterns found in that specific patient's x-rays. A human could be confused that there is in fact no finding in the x-ray as well.

Model 3 Case Analysis

	Easy	Medium	Hard
Model 1	Both	Finetuned	Finetuned
Model 2	Finetuned	Both	Both
Model 4	Both	Finetuned	Both

Model 3 performs better than expected on all cases of other models, which could be explained by how it uses text embeddings rather than image classification

Model 4 Case Analysis

	Easy	Medium	Hard
Model 1	Both	Both	Both
Model 2	Finetuned	Finetuned	Neither
Model 3	Both	Finetuned	Neither

The model achieved its strongest improvements in easy cases, where it successfully learned to identify gross geometric features such as enlarged heart silhouettes. It also showed solid performance in medium, texture-based cases like Pneumonia and Lung Opacity. However, the model struggled a lot with the hard cases. The fine-detail cases such as Pneumothorax shows a minimal improvement. This failure is potentially cause by the model's lower input resolution (224x224) which blurs out the tiny and high-frequency details. For example, the thin pleural lines can me hard to identify which is critical for diagnosis.

Division of Labor

- 4 team members, 4 separately fine-tuned models:
  - Andy: EfficientNet-B0
  - Ben: densenet121-res224-nih
  - Ryan: clip-vit-base-patch32
  - Yihao: densenet121-res224-pc

README Outline

- Contains direct link to this final report presentation
- Contains instructions for setting up the python environment and downloading the CheXpert dataset
  - Dataset needs to local for our code to work
- Contains instructions for requesting a PACE GPU
  - Highly recommended due to the highly computational task of training models
- Each sub folder (clip, densenet, densenet-nih, imagenet) contains further instructions on how to run their respective baseline inference, finetuning + inference, heatmap generation, and interpretability

Summary/Lessons Learned

- Fine-tuning is an interesting and difficult problem
  - Highest score on CheXpert was a 0.930 with a model fine-tuned to maximize AUC
  - Swapping out the classifier head is important for the model to learn the new labels from a new dataset
- It is inherently difficult to choose a single prediction threshold that balances all metrics across all labels
- LoRA rank and alpha parameters don't affect the fine-tuning performance; only the number of LoRA adapter applied does
- Heatmaps are a great way to visualize image classification problems
- Text embedding approach needs a lot of work

Next Steps

- Modify our fine-tuning approach
  - Full parameters
  - LoRA parameters
- Replace a single prediction threshold with best threshold per label
- Explore other models
- Potentially an interface for user interaction
- Analyze Failure Cases with Grad-CAM
- Benchmark Against External Datasets
  - Test model on a subset of MIMIC-CXR

References

Chan HP, Samala RK, Hadjiiski LM, Zhou C. (2020). Deep Learning in Medical Image Analysis. Adv Exp Med Biol, 2020.

Irvin, J. et al. (2019). CheXpert: A Large Chest Radiograph Dataset with Uncertainty Labels and Expert Comparison. AAAI, 2019.

Rajpurkar, P. et al. (2017). CheXNet: Radiologist-Level Pneumonia Detection on Chest X-Rays with Deep Learning.

Hu, Edward J., et al. "Lora: Low-rank adaptation of large language models." ICLR 1.2 (2022): 3.