Zero-Shot Radiological Screening with Contrastive Pretraining

MedVista Diagnostics -- Building a Language-Guided Medical Image Retrieval System

---

Section 1: Industry Context and Business Problem

Industry: Healthcare -- Diagnostic Radiology

MedVista Diagnostics is a mid-size healthcare AI company serving 340 hospitals across the southeastern United States. Their core product is an AI-assisted radiology workflow system that helps radiologists triage and interpret medical images more efficiently.

Company Profile

- **Founded:** 2019
- **Employees:** 180 (45 ML engineers, 12 radiologists on advisory board)
- **Revenue:** USD 28M ARR from SaaS contracts with hospital networks
- **Flagship Product:** RadAssist -- an AI copilot for radiologists that flags potential abnormalities in chest X-rays, CT scans, and MRIs

The Business Challenge

MedVista's current system is built on traditional supervised classification models. Each model is trained on a specific pathology: one model for pneumonia detection, another for cardiomegaly, another for pleural effusion, and so on. This creates three critical problems:

1. **Label Bottleneck:** Training each new pathology detector requires 10,000--50,000 labeled images, annotated by board-certified radiologists at USD 15--40 per annotation. Adding support for a new condition costs USD 150K--500K in labeling alone.

2. **Long Tail of Conditions:** There are over 200 clinically significant radiological findings. MedVista currently supports only 14. Rare conditions (e.g., sarcoidosis, pneumomediastinum) will never have enough labeled data for supervised training.

3. **Deployment Rigidity:** When a new pathology needs to be detected, MedVista must retrain a new model, validate it against FDA requirements, and deploy it -- a process that takes 6--9 months. Competitors with more flexible systems are winning contracts.

Stakes

Three major hospital networks (representing USD 8.2M in annual revenue) have issued RFPs requiring support for at least 50 radiological findings. MedVista's current 14-condition system will lose these contracts unless they dramatically expand coverage.

Constraints

- Must operate on standard radiology PACS (Picture Archiving and Communication System) hardware
- Inference latency must be under 500ms per image
- Must maintain or exceed current per-pathology AUC (>0.88) for the 14 existing conditions
- FDA 510(k) clearance pathway requires interpretable similarity scores, not black-box classifications
- Training data is limited: USD 200K budget for additional annotations

Proposed Solution

Build a contrastive pretraining system (CLIP-style) that learns a shared embedding space between radiology images and clinical text descriptions. This enables:

- **Zero-shot classification:** Detect any condition by providing its text description, no labeled data needed
- **Text-guided retrieval:** Find similar cases by describing symptoms in natural language
- **Flexible expansion:** Add new conditions in minutes (write a text prompt) instead of months (train a new model)

Section 2: Technical Problem Formulation

Problem Type: Multimodal Contrastive Representation Learning

Justification: The core challenge is learning an alignment between visual features in medical images and semantic concepts expressed in clinical text. This is fundamentally a representation learning problem where the goal is to create a shared embedding space that captures medically meaningful relationships between images and their textual descriptions. Contrastive pretraining is the appropriate approach because:

1. It does not require per-pathology labeled data
2. It leverages existing radiology reports as natural language supervision
3. It enables zero-shot generalization to unseen conditions
4. The learned embeddings are inherently interpretable (similar images have similar embeddings)

Input/Output Specifications

Inputs: - Medical images: chest X-rays, 224x224 grayscale, normalized to [0, 1] - Clinical text: radiology report findings, tokenized to max 77 tokens

Outputs: - Image embeddings: $z_I \in \mathbb{R}^{512}$, L2-normalized - Text embeddings: $z_T \in \mathbb{R}^{512}$, L2-normalized - Similarity scores: $s_{ij} = z_{I_i} \cdot z_{T_j} \in [-1, 1]$

Mathematical Foundation

The model consists of two encoders:

$$f_\theta : \mathcal{X}_{\text{image}} \to \mathbb{R}^d, \quad g_\phi : \mathcal{X}_{\text{text}} \to \mathbb{R}^d$$

where $d = 512$ is the embedding dimension.

For a batch of N image-text pairs $\{(x_{I_i}, x_{T_i})\}_{i=1}^N$, we compute normalized embeddings:

$$\hat{z}_{I_i} = \frac{f_\theta(x_{I_i})}{||f_\theta(x_{I_i})||_2}, \quad \hat{z}_{T_i} = \frac{g_\phi(x_{T_i})}{||g_\phi(x_{T_i})||_2}$$

Loss Function

The loss is a symmetric InfoNCE objective:

$$\mathcal{L} = \frac{1}{2}\left(\mathcal{L}_{I\to T} + \mathcal{L}_{T\to I}\right)$$

where:

$$\mathcal{L}_{I\to T} = -\frac{1}{N}\sum_{i=1}^N \log \frac{\exp(\hat{z}_{I_i} \cdot \hat{z}_{T_i}/\tau)}{\sum_{j=1}^N \exp(\hat{z}_{I_i} \cdot \hat{z}_{T_j}/\tau)}$$

Per-term justification:

- **Numerator** $\exp(\hat{z}_{I_i} \cdot \hat{z}_{T_i}/\tau)$: Measures alignment between matched image-text pairs. Maximizing this pushes correct pairs together in embedding space.
- **Denominator** $\sum_j \exp(\hat{z}_{I_i} \cdot \hat{z}_{T_j}/\tau)$: Normalizes across all texts in the batch, creating an implicit N -way classification problem. This pushes non-matching pairs apart.
- **Temperature** τ : Controls the concentration of the distribution. Learned during training. Lower values increase discrimination but risk training instability.
- **Symmetry** $\frac{1}{2}(\mathcal{L}_{I\to T} + \mathcal{L}_{T\to I})$: Ensures both modalities contribute equally. Without symmetry, the model might learn good image representations but poor text representations (or vice versa).
- **Log and negative sign**: Converting the softmax probability to negative log-likelihood creates a proper cross-entropy loss that can be minimized with gradient descent.

Evaluation Metrics

Metric	Description	Target
Zero-shot AUC	Per-pathology AUC using text prompts only	>0.80
Retrieval R@5	Fraction of queries where correct match is in top 5	>0.70
Embedding alignment	Mean cosine similarity for matched pairs	>0.60
Inference latency	Time per image embedding	<200ms
Coverage	Number of conditions detectable without fine-tuning	>50

Baseline

The baseline is MedVista's existing supervised ResNet-50 classifiers, each trained on 10K+ labeled images per pathology. These achieve AUC >0.88 on their 14 supported conditions but cannot generalize to unseen conditions.

Why Contrastive Pretraining

Supervised classification requires expensive per-class labels and cannot scale to hundreds of conditions. Contrastive pretraining solves this by:

1. Learning from radiology reports (free, existing data) rather than manual labels
2. Enabling zero-shot transfer to any condition describable in text
3. Producing interpretable similarity scores (required for FDA clearance)
4. Supporting both classification and retrieval in a unified system

---

Section 3: Implementation Notebook Structure

3.1 Data Loading and Preparation

```
def load_medical_dataset(data_dir, split='train'):
    """
    Load the chest X-ray dataset with paired radiology reports.

    Args:
        data_dir: path to dataset root
        split: 'train', 'val', or 'test'

    Returns:
        images: list of image paths
        reports: list of radiology report strings
        labels: dict mapping pathology names to binary labels

    # ============ TODO ============
    # Step 1: Load image paths from data_dir/split/
    # Step 2: Load corresponding radiology reports from CSV
    # Step 3: Parse multi-label annotations for evaluation
    # Step 4: Apply preprocessing (resize to 224x224, normalize)
    # Hint: Use torchvision.transforms for image preprocessing
    # ==============================
    """
    pass
```

3.2 Exploratory Data Analysis

```
def exploratory_analysis(images, reports, labels):
    """
    Analyze the dataset distribution and characteristics.

    Tasks:
    - Plot label frequency distribution across pathologies
    - Show sample images with their report excerpts
    - Compute report length statistics (tokens, sentences)
    - Identify class imbalance and rare conditions
    - Visualize image quality distribution (brightness, contrast)

    # ============ TODO ============
    # Step 1: Count frequency of each pathology label
    # Step 2: Plot horizontal bar chart of label frequencies
    # Step 3: Display 3x3 grid of sample images with report snippets
    # Step 4: Histogram of report lengths
    # Step 5: Flag conditions with fewer than 100 examples
    # ==============================
    """
    pass
```

3.3 Baseline Model

```
def train_supervised_baseline(train_loader, val_loader, num_classes=14):
    """
    Train a supervised ResNet-50 classifier as baseline.

    Architecture:
    - ResNet-50 pretrained on ImageNet
    - Replace final FC layer with num_classes outputs
    - Binary cross-entropy loss for multi-label classification

    # ============ TODO ============
    # Step 1: Load pretrained ResNet-50
    # Step 2: Replace classifier head
    # Step 3: Train for 10 epochs with Adam optimizer
    # Step 4: Evaluate per-class AUC on validation set
    # Step 5: Report which conditions have AUC < 0.80
    # ==============================
    """
    pass
```

3.4 CLIP-Style Model Architecture

```
class MedCLIP(nn.Module):
    """
    Medical CLIP model with image and text encoders.

    Image encoder: ViT-B/16 initialized from ImageNet pretraining
    Text encoder: BioClinicalBERT (domain-specific)
    Projection: Linear layers mapping to 512-d shared space

    # ============ TODO ============
    # Step 1: Initialize image encoder (ViT-B/16 or ResNet-50)
    # Step 2: Initialize text encoder (BioClinicalBERT)
    # Step 3: Add projection heads (image_dim -> 512, text_dim -> 512)
    # Step 4: Initialize learnable temperature parameter
    # Step 5: Implement forward() that returns normalized embeddings
    # ==============================
    """
    def __init__(self, image_encoder, text_encoder, embed_dim=512):
        super().__init__()
        pass
    def forward(self, images, input_ids, attention_mask):
        pass

    def compute_loss(self, image_emb, text_emb):
        pass
```

3.5 Training Pipeline

```
def train_medclip(model, train_loader, val_loader, num_epochs=50):
    """
    Train the MedCLIP model with contrastive learning.

    Key considerations:
    - Use mixed-precision training (AMP) for memory efficiency
    - Gradient accumulation for effective batch size of 1024
    - Learning rate warmup + cosine decay schedule
    - Monitor contrastive accuracy and retrieval metrics

    # ============ TODO ============
    # Step 1: Set up optimizer (AdamW, lr=5e-5, weight_decay=0.01)
    # Step 2: Set up scheduler (linear warmup + cosine decay)
    # Step 3: Training loop with gradient accumulation
    # Step 4: Log contrastive loss, accuracy, and temperature
    # Step 5: Validate every epoch with retrieval metrics
    # ==============================
    """
    pass
```

3.6 Evaluation and Zero-Shot Classification

```
def evaluate_zero_shot(model, test_loader, pathology_names, templates):
    """
    Evaluate zero-shot classification performance.

    For each pathology:
    1. Create text prompts using templates
    2. Compute average text embedding (prompt ensembling)
    3. Compute image-text similarities for all test images
    4. Compute AUC using similarities as prediction scores

    # ============ TODO ============
    # Step 1: Pre-compute text embeddings for all pathologies
    # Step 2: For each test image, compute similarity to all pathologies
    # Step 3: Compute per-pathology AUC
    # Step 4: Compare with supervised baseline AUCs
    # Step 5: Identify conditions where zero-shot matches/exceeds supervised
    # ==============================
    """
    pass
```

3.7 Error Analysis

```
def error_analysis(model, test_loader, pathology_names):
    """
    Analyze failure modes of the zero-shot system.

    Tasks:
    - Compute confusion between similar conditions
    - Identify images where zero-shot fails but supervised succeeds
    - Analyze correlation between report quality and embedding quality
    - Test sensitivity to prompt wording variations
    - Examine embedding space for cluster quality

    # ============ TODO ============
    # Step 1: Collect all predictions and ground truth
    # Step 2: Build confusion matrix for top-3 predictions
    # Step 3: Visualize failure cases with images and similarity scores
    # Step 4: Test 5 different prompt templates per condition
    # Step 5: t-SNE of image embeddings colored by pathology
    # ==============================
    """
    pass
```

3.8 Deployment Considerations

```
def optimize_for_deployment(model, sample_input):
    """
    Prepare the model for production deployment.

    Tasks:
    - Export to ONNX format for cross-platform inference
    - Benchmark inference latency on CPU and GPU
    - Implement batch processing for throughput optimization
    - Set up embedding caching for text prompts
    - Validate numerical consistency between PyTorch and ONNX

    # ============ TODO ============
    # Step 1: Trace model with torch.jit.trace
    # Step 2: Export to ONNX with dynamic batch size
    # Step 3: Benchmark latency for single image and batch of 16
    # Step 4: Implement text embedding cache (dictionary)
    # Step 5: Verify ONNX output matches PyTorch within tolerance
    # ==============================
    """
    pass
```

3.9 Ethical Considerations

```
def fairness_analysis(model, test_loader, demographic_metadata):
    """
    Evaluate model fairness across demographic groups.

    Tasks:
    - Compute zero-shot AUC stratified by age, sex, and ethnicity
    - Identify conditions with significant performance gaps
    - Test for bias in text encoder (prompt sensitivity analysis)
    - Document limitations for FDA submission

    # ============ TODO ============
    # Step 1: Stratify test set by demographic attributes
    # Step 2: Compute per-group AUC for each pathology
    # Step 3: Flag conditions with AUC gap > 0.05 between groups
    # Step 4: Test if different prompts affect different groups differently
    # Step 5: Generate fairness report table
    # ==============================
    """
    pass
```

Section 4: Production and System Design Extension

System Architecture

The production MedCLIP system consists of the following components:

1. **Image Ingestion Service:** Receives DICOM images from hospital PACS via HL7 FHIR interface, performs DICOM-to-PNG conversion, and applies standardized preprocessing (resizing, windowing, normalization).

2. **Embedding Service:** GPU-backed microservice running the MedCLIP image encoder. Receives preprocessed images and returns 512-d normalized embeddings. Deployed as a containerized service with auto-scaling.

3. **Prompt Registry:** Database of validated text prompts for each pathology, stored as pre-computed text embeddings. Updated when new conditions are added (no model retraining required).

4. **Similarity Engine:** Computes cosine similarity between image embeddings and all registered pathology embeddings. Returns ranked results with confidence scores.

5. **Result Aggregation:** Combines similarity scores with clinical rules (age-appropriate screening, anatomical constraints) to produce final findings.

API Design

```
POST /api/v1/screen
  Input: DICOM image (binary)
  Output: {
    "findings": [
      {"pathology": "cardiomegaly", "confidence": 0.92, "rank": 1},
      {"pathology": "pleural_effusion", "confidence": 0.78, "rank": 2}
    ],
    "embedding": [0.12, -0.34, ...],  // 512-d vector
    "latency_ms": 187,
    "model_version": "medclip-v2.1"
  }

POST /api/v1/retrieve
  Input: {"query": "bilateral ground-glass opacities", "top_k": 10}
  Output: {
    "results": [
      {"image_id": "CXR-2024-00142", "similarity": 0.89, "metadata": {...}},
      ...
    ]
  }

POST /api/v1/register-condition
  Input: {"name": "sarcoidosis", "prompts": ["bilateral hilar lymphadenopathy", ...]}
  Output: {"status": "registered", "embedding_id": "emb-00234"}
```

Serving Infrastructure

- **GPU Cluster:** 4x NVIDIA A10G GPUs behind a load balancer
- **Batch Inference:** Images accumulated into batches of 16 for throughput optimization
- **Embedding Cache:** Redis-backed cache for text embeddings (invalidated on prompt updates)
- **Model Serving:** TorchServe with ONNX Runtime backend for inference optimization
- **Target Throughput:** 200 images/second at p99 latency <500ms

Monitoring

Metric	Description	Alert Threshold
Inference latency (p95)	Image encoding time	>400ms

Metric	Description	Alert Threshold
Similarity distribution	Mean similarity for top prediction	Shift >0.05 from baseline
Embedding drift	Cosine distance between rolling averages	>0.10 per week
Zero-shot AUC	Weekly AUC on validation holdout	Drop >0.03
Cache hit rate	Fraction of text embeddings served from cache	<0.95

Drift Detection

The system monitors for two types of drift:

1. **Data drift:** The distribution of incoming images changes (new scanner types, different patient demographics, acquisition protocol changes). Detected by monitoring the mean and variance of image embeddings over time.

2. **Concept drift:** The relationship between images and pathologies changes (new clinical guidelines, updated terminology). Detected by periodically evaluating zero-shot AUC on a held-out validation set.

When drift is detected, the system: - Triggers an alert to the ML ops team - Automatically increases the validation frequency - Logs affected image embeddings for review - Does NOT automatically retrain (regulatory constraint -- any model update requires FDA re-evaluation)

A/B Testing

For evaluating model improvements: - Shadow mode: new model runs in parallel, predictions logged but not surfaced - Clinical validation: 2-week parallel run with radiologist review of discordant predictions - Rollout: gradual traffic shift (10% -> 25% -> 50% -> 100%) over 4 weeks - Rollback criterion: if per-pathology AUC drops by >0.02 for any existing condition

CI/CD Pipeline

```
Code Push -> Unit Tests -> Integration Tests (synthetic data)
  -> Model Training (full pipeline, ~8 hours)
  -> Validation Suite (zero-shot AUC, retrieval R@5, fairness checks)
  -> Staging Deployment (shadow mode, 48 hours)
  -> Clinical Review (radiologist spot-check)
  -> Production Deployment (canary -> full rollout)
```

Cost Analysis

Component	Monthly Cost
GPU inference (4x A10G)	USD3,200
Storage (embeddings, images)	USD450
Redis cache	USD180
Monitoring and logging	USD120

Component	Monthly Cost
Training (periodic, 8x A100 for 12 hours)	USD960/run
Total operational	**USD4,910/month**

Compared to the supervised approach (training and maintaining 50+ separate models at ~USD2,000/model/year for annotation and retraining), MedCLIP reduces per-condition costs by approximately 85%.