DDPM Case Study: Synthetic Medical Image Generation for Rare Disease Detection

Section 1: Industry Context and Business Problem

1.1 Industry: Healthcare — Medical Imaging and Radiology

Medical imaging is a \$45 billion global market, with chest X-rays being the most commonly ordered diagnostic imaging procedure worldwide. Hospitals and radiology departments process millions of scans annually, and AI-assisted diagnosis has become a critical area of investment. Machine learning models trained to detect pathologies from X-rays can reduce diagnostic errors, accelerate reporting, and extend diagnostic capability to underserved regions.

1.2 Company Profile: RadianceAI

RadianceAI is a Series B medical AI startup headquartered in Boston, Massachusetts. The company develops AI-powered diagnostic tools for chest X-ray interpretation, deployed across 120 hospitals in the United States and Southeast Asia. RadianceAI's flagship product, RadianceScreen, detects 14 common chest pathologies (pneumonia, cardiomegaly, pleural effusion, etc.) with radiologist-level accuracy. The company employs 85 people, including a team of 22 machine learning engineers and 8 board-certified radiologists who serve as clinical advisors.

Key Metrics: - 2.4 million X-rays processed in 2025 - 94.2% mean AUC across the 14 common pathologies - FDA 510(k) clearance for the core product - \$38 million in annual recurring revenue

1.3 Business Challenge: The Rare Disease Gap

RadianceAI's model performs well on common pathologies but struggles with rare conditions. The company has identified 6 critical rare pathologies — including pulmonary alveolar proteinosis (PAP), lymphangioleiomyomatosis (LAM), and pulmonary Langerhans cell histiocytosis (PLCH) — that collectively affect fewer than 200,000 patients in the US annually.

The problem is acute:

1. **Data scarcity:** Across 2.4 million scans, fewer than 800 confirmed cases of these 6 rare conditions exist. Standard deep learning models require thousands to tens of thousands of examples per class.

2. **Clinical impact:** Misdiagnosis of rare pulmonary diseases leads to delayed treatment, with average diagnostic delays exceeding 3 years for some conditions. RadianceAI's clinical advisory board reports that 40% of PAP patients receive at least one incorrect diagnosis before the condition is identified.

3. **Regulatory requirement:** To pursue FDA clearance for rare disease detection, RadianceAI needs to demonstrate model performance on a statistically significant number of cases — far more than currently available.

4. **Market opportunity:** Hospitals are willing to pay a 35% premium for AI tools that can flag rare conditions. This represents a \$15 million incremental ARR opportunity.

1.4 Stakes and Constraints

Stakes: - Patient safety: missed rare diagnoses lead to disease progression and potentially irreversible lung damage - Regulatory: FDA requires robust validation sets; fewer than 100 cases per condition is insufficient - Commercial: capturing the rare disease module would differentiate RadianceAI from 4 direct competitors

Constraints: - Patient privacy (HIPAA): real patient data cannot be shared or artificially duplicated - Clinical validity: synthetic images must be radiologically plausible — reviewed by board-certified radiologists - Compute budget: \$50,000 allocated for training infrastructure (GPU compute on AWS) - Timeline: 6 months to deliver a validated synthetic data pipeline

1.5 Proposed Solution

RadianceAI will use **Denoising Diffusion Probabilistic Models (DDPMs)** to generate synthetic chest X-rays for the 6 rare pathologies. The diffusion model will be trained on the existing (small) set of confirmed rare disease X-rays, augmented with class-conditional generation techniques. The synthetic images will then be used to augment the training set for the downstream diagnostic classifier.

The choice of DDPMs over alternatives: - **GANs:** Mode collapse is unacceptable in medical imaging — the model must produce diverse pathological presentations, not variations of a single pattern. DDPMs are known for superior mode coverage. - **VAEs:** VAE-generated images tend to be blurry, which would obscure fine-grained pathological features critical for diagnosis. - **DDPMs:** Produce high-fidelity, diverse images without mode collapse. The iterative denoising process naturally captures multi-scale features — from large anatomical structures to subtle opacities.

Section 2: Technical Problem Formulation

2.1 Problem Type: Conditional Image Generation

We formulate this as a **class-conditional image generation** problem. Given a pathology label $c \in \{c_1, c_2, \ldots, c_6\}$, generate a synthetic chest X-ray $\mathbf{x} \in \mathbb{R}^{256\times256}$ that exhibits the radiological features characteristic of condition c.

Justification: Conditional generation is the correct formulation because we need control over which pathology appears in the generated image. Unconditional generation would produce images distributed across all conditions, failing to address the targeted data scarcity problem.

2.2 Input/Output Specifications

Input: - Pathology class label $c \in \{0, 1, 2, 3, 4, 5\}$ (one of 6 rare conditions) - Random noise $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, dimension 256×256

Output: - Synthetic chest X-ray image $\mathbf{x}_0 \in \mathbb{R}^{256\times256}$ - Radiologically plausible image exhibiting features of the specified pathology

2.3 Mathematical Foundation

Forward Process (Fixed)

The forward process adds noise over $T = 1000$ steps:

$$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}\left(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\,\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I}\right)$$

where $\bar{\alpha}_t = \prod_{s=1}^{t}(1-\beta_s)$ and $\beta_1, \ldots, \beta_T$ is a linear noise schedule from 10^{-4} to 0.02.

Reverse Process (Learned)

The reverse process is parameterized as:

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t, c) = \mathcal{N}\left(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t, c), \sigma_t^2\mathbf{I}\right)$$

where c is the class label injected via class-conditional embeddings.

Noise Prediction Formulation

The model predicts the noise $\epsilon_\theta(\mathbf{x}_t, t, c)$ rather than the mean directly, following the DDPM simplification.

2.4 Loss Function

$$\mathcal{L} = \mathbb{E}_{t,\mathbf{x}_0,\boldsymbol{\epsilon},c}\left[\left\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta\left(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon},\ t,\ c\right)\right\|^2\right]$$

Per-term justification: - ϵ : the ground-truth noise sampled from $\mathcal{N}(\mathbf{0}, \mathbf{I})$ - $\epsilon_\theta(\cdot)$: the U-Net's predicted noise, conditioned on the noisy image, timestep, and class label - $\|\cdot\|^2$: mean squared error — the natural loss for Gaussian noise prediction since the true posterior has fixed variance - The expectation averages over uniformly sampled timesteps t , training images $\mathbf{x}_0$, noise samples ϵ , and class labels c

2.5 Evaluation Metrics

1. **Frechet Inception Distance (FID):** Measures the distributional distance between generated and real images. Lower is better. Target: FID < 30 for medical X-rays.
2. **Clinical Turing Test:** Board-certified radiologists classify images as real or synthetic in a blinded study. Target: radiologist accuracy below 60% (near chance level).
3. **Downstream Classifier Improvement:** AUC improvement on the rare disease detection task when training data is augmented with synthetic images. Target: minimum 5% absolute AUC improvement per condition.
4. **Diversity Score (MS-SSIM):** Multi-Scale Structural Similarity between generated image pairs within the same class. Target: MS-SSIM < 0.5 (indicating diversity, not mode collapse).
5. **Pathological Feature Accuracy:** Expert radiologist scoring of whether generated images contain the correct pathological features (opacities, ground-glass patterns, cysts, etc.). Target: > 85% feature accuracy.

2.6 Baseline

The baseline approach is standard data augmentation (rotation, flipping, brightness adjustment, elastic deformation) applied to the existing rare disease images. This typically provides 2-3x augmentation but does not introduce novel pathological presentations.

2.7 Why DDPMs

DDPMs are the right solution because: 1. **Mode coverage:** Unlike GANs, DDPMs cover the full data distribution, generating diverse pathological presentations 2. **Stable training:** No adversarial training dynamics, mode collapse, or training instability 3. **Progressive generation:** The iterative denoising captures features at multiple scales — from gross anatomy to subtle opacities 4. **Class conditioning:** Straightforward to condition on pathology labels via embedding injection 5. **Quality control:** The denoising process naturally produces smooth, artifact-free images

---

Section 3: Implementation Notebook Structure

3.1 Data Loading and Preprocessing

```
def load_rare_disease_dataset(data_dir, image_size=256):
    """
    TODO: Load the chest X-ray dataset with rare disease labels.

    Args:
        data_dir: Path to the dataset directory containing subfolders
                  for each pathology class
        image_size: Target image resolution (default 256x256)

    Returns:
        dataset: PyTorch Dataset with (image, label) pairs
        class_counts: Dict mapping class index to number of samples

    Implementation hints:
        - Use torchvision.datasets.ImageFolder for directory-based loading
        - Apply transforms: Resize, ToTensor, Normalize to [-1, 1]
        - Log class distribution to identify imbalance
        - Expected: ~100-150 images per rare condition

    Verification:
        - Print class distribution
        - Visualize 5 random samples from each class
        - Verify image dimensions are (1, 256, 256)
    """
    pass
```

3.2 Exploratory Data Analysis

```
def eda_rare_diseases(dataset, class_names):
    """
    TODO: Perform exploratory data analysis on the rare disease dataset.

    1. Plot class distribution (bar chart)
    2. Compute and display pixel intensity histograms per class
    3. Compute mean and std images per class
    4. Display example images in a grid (5 per class)
    5. Compute pairwise SSIM between classes to measure visual overlap

    This analysis will reveal:
    - How severe the class imbalance is
    - Whether different pathologies have distinctive visual signatures
    - The baseline visual similarity between conditions
    """
    pass
```

3.3 Baseline: Standard Data Augmentation

```
def train_classifier_with_augmentation(dataset, num_epochs=50):
    """
    TODO: Train a ResNet-18 classifier using standard augmentation only.

    Standard augmentation pipeline:
        - Random horizontal flip
        - Random rotation (up to 10 degrees)
        - Random brightness/contrast adjustment
        - Random elastic deformation

    Train/val split: 80/20 stratified by class
    Optimizer: Adam, lr=1e-4, weight_decay=1e-5
    Loss: Cross-entropy with class weights (inverse frequency)

    Returns:
        model: Trained classifier
        val_metrics: Dict with per-class AUC, accuracy, F1
```

```
    This establishes the baseline performance WITHOUT synthetic data.

    Verification:
        - Plot training/validation loss curves
        - Print per-class AUC scores
        - Print confusion matrix
    """
    pass
```

3.4 Model Architecture: Class-Conditional DDPM

```
def build_conditional_unet(in_channels=1, base_channels=128, num_classes=6, time_dim=512):
    """
    TODO: Build a class-conditional U-Net for noise prediction.

    Architecture:
        - Sinusoidal timestep embedding -> MLP -> time_dim
        - Class embedding: nn.Embedding(num_classes, time_dim)
        - Combined conditioning: time_emb + class_emb (added)
        - U-Net with 4 resolution levels: 256->128->64->32->16
        - Channel progression: 1->128->256->512->512
        - ResBlocks with GroupNorm and SiLU activation
        - Self-attention at 32x32 and 16x16 resolutions
        - Skip connections between matching encoder/decoder levels

    Args:
        in_channels: Number of input channels (1 for grayscale X-rays)
        base_channels: Base channel count (128)
        num_classes: Number of pathology classes (6)
        time_dim: Dimension of time/class embedding (512)

    Returns:
        model: nn.Module that takes (x_t, t, class_label) and returns predicted noise

    Verification:
        - Print total parameter count (target: ~50M)
        - Verify output shape matches input shape
        - Test with random inputs at different timesteps
    """
    pass
```

3.5 Training Loop

```
def train_ddpm(model, dataset, num_epochs=200, batch_size=16, lr=2e-4, T=1000):
    """
    TODO: Implement the full DDPM training loop.

    For each batch:
        1. Sample random timesteps t ~ Uniform(1, T)
        2. Sample noise epsilon ~ N(0, I)
        3. Compute x_t using closed-form forward process
        4. Predict noise: epsilon_hat = model(x_t, t, class_label)
        5. Compute loss: MSE(epsilon, epsilon_hat)
        6. Backpropagate and update

    Additional requirements:
        - Use EMA (Exponential Moving Average) of model weights for sampling
        - Log loss every 100 steps
        - Save model checkpoint every 10 epochs
        - Generate and save sample images every 5 epochs

    Args:
        model: The conditional U-Net
        dataset: Training dataset
        num_epochs: Number of training epochs (200)
        batch_size: Batch size (16 — limited by GPU memory at 256x256)
        lr: Learning rate (2e-4)
        T: Number of diffusion steps (1000)

    Returns:
```

```
        model: Trained model
        ema_model: EMA model for sampling
        loss_history: List of loss values

    Verification:
        - Loss should decrease from ~1.0 to ~0.05-0.1
        - Generated samples should become recognizable after ~50 epochs
        - EMA samples should be higher quality than non-EMA
    """
    pass
```

3.6 Evaluation: Image Quality Assessment

```
def evaluate_generated_images(model, real_dataset, num_samples_per_class=500, T=1000):
    """
    TODO: Comprehensive evaluation of generated image quality.

    Metrics to compute:
        1. FID score per class and overall
        2. MS-SSIM diversity score per class
        3. Pixel intensity histogram comparison (real vs generated)
        4. Generate sample grids for visual inspection

    Steps:
        1. Generate num_samples_per_class images per pathology class
        2. Compute FID between real and generated distributions per class
        3. Compute pairwise MS-SSIM within generated images (diversity check)
        4. Create comparison grids: 5 real + 5 generated per class

    Returns:
        metrics: Dict with FID, MS-SSIM, and histogram stats per class

    Verification:
        - FID < 30 for each class
        - MS-SSIM < 0.5 for each class (high diversity)
        - Histogram shapes should roughly match
    """
    pass
```

3.7 Error Analysis

```
def error_analysis(model, real_dataset, radiologist_labels=None):
    """
    TODO: Analyze failure modes of the generative model.

    1. Identify generated images with highest FID contribution
       (most "unrealistic" samples)
    2. Cluster generated images and check for mode collapse
       (are all generated images similar?)
    3. Compare edge cases: images near the boundary between classes
    4. Check for anatomical implausibility:
       - Heart on wrong side
       - Missing anatomical landmarks
       - Unrealistic rib structure
    5. If radiologist labels available, analyze which features
       radiologists find most/least convincing

    Returns:
        failure_report: Dict with categorized failure modes and examples

    Verification:
        - Display the 10 most unrealistic generated images with explanation
        - Show the 10 most realistic generated images for comparison
        - Plot FID contribution distribution
    """
    pass
```

3.8 Deployment: Synthetic Data Pipeline

```
def deploy_synthetic_pipeline(model, output_dir, num_images_per_class=2000):
    """
    TODO: Build a production-ready synthetic data generation pipeline.

    Pipeline steps:
        1. Generate images using DDIM (50 steps) for speed
        2. Apply automatic quality filtering:
           - Reject images with FID score > threshold
           - Reject images with anatomical implausibility score > threshold
        3. Save generated images with metadata:
           - Class label
           - Generation timestamp
           - Quality score
           - DDIM step count
        4. Generate dataset statistics report

    Args:
        model: Trained DDPM model
        output_dir: Output directory for generated images
        num_images_per_class: Target number per class (2000)

    Returns:
        report: Dict with generation statistics

    Verification:
        - Total generated images should be ~12,000 (2000 x 6 classes)
        - Quality filtering should reject < 10% of images
        - File sizes should be consistent (~50-100KB per image)
    """
    pass
```

3.9 Ethics and Responsible AI

```
def ethics_assessment(real_dataset, generated_dataset):
    """
    TODO: Conduct an ethics assessment for the synthetic data pipeline.

    Key considerations:
        1. Patient privacy: Verify that no generated image can be traced
           back to a specific patient (nearest-neighbor analysis)
        2. Demographic bias: Check if the model reproduces or amplifies
           demographic biases present in the training data
           (e.g., disproportionate representation of certain body types)
        3. Clinical safety: Ensure that generated images do not introduce
           spurious correlations that could mislead the downstream classifier
        4. Informed consent: Document data provenance and usage rights
        5. Regulatory compliance: Map to FDA guidelines for AI/ML-based
           software as a medical device (SaMD)

    Returns:
        ethics_report: Dict with findings and recommendations

    Verification:
        - Minimum L2 distance between generated and real images > threshold
        - Demographic attribute distribution matches training data
        - No spurious anatomical features that could mislead diagnosis
    """
    pass
```

Section 4: Production and System Design Extension

4.1 System Architecture

The production system consists of three main components:

1. **Synthetic Data Generator Service:** A GPU-backed microservice that generates chest X-rays on demand. Accepts class label and quality parameters, returns synthetic images. Deployed on AWS SageMaker with auto-scaling based on queue depth.

2. **Quality Assurance Module:** An automated pipeline that filters generated images using a combination of FID scoring, anatomical landmark detection, and a pre-trained discriminator. Images that pass quality checks are added to the augmented training pool.

3. **Model Retraining Pipeline:** A scheduled pipeline (weekly) that retrains the downstream diagnostic classifier using the augmented dataset. Includes A/B testing infrastructure to compare classifier performance with and without synthetic data.

4.2 API Design

```
POST /api/v1/generate
{
    "pathology_class": "PAP",
    "num_images": 100,
    "quality_threshold": 0.8,
    "sampling_method": "ddim",
    "ddim_steps": 50,
    "image_size": 256,
    "seed": 42
}

Response:
{
    "job_id": "gen-2024-001",
    "status": "processing",
    "estimated_time_seconds": 180,
    "callback_url": "/api/v1/jobs/gen-2024-001"
}
```

4.3 Model Serving

- **Inference:** DDIM with 50 steps on A10G GPU. Per-image latency: ~2 seconds.
- **Batch generation:** Batch size 16 on A100 GPU. Throughput: ~500 images/hour.
- **Model storage:** EMA model weights stored in S3, versioned with MLflow.
- **Model format:** TorchScript for production serving, ONNX for edge deployment validation.

4.4 Monitoring

Real-time metrics: - Generation latency (p50, p95, p99) - FID score of generated batches (rolling 1000-image window) - GPU utilization and memory usage - Error rate (failed generations)

Batch metrics (daily): - FID trend over time (detecting model degradation) - Class distribution of generated images - Quality filter rejection rate - Downstream classifier AUC on validation set

Alerts: - FID score exceeds 50 (quality degradation) - Rejection rate exceeds 15% (model drift) - Downstream AUC drops by more than 2% (regression) - GPU utilization below 20% for more than 1 hour (infrastructure waste)

4.5 Drift Detection

Data drift is monitored at two levels:

1. **Input drift:** Compare the distribution of real X-rays arriving at the hospital against the distribution used to train the DDPM. Use Maximum Mean Discrepancy (MMD) between feature embeddings. If MMD exceeds threshold, trigger model retraining.
2. **Output drift:** Compare the quality distribution of generated images over time. A sudden increase in FID or decrease in feature accuracy indicates that the model may need retraining on updated real data.

4.6 A/B Testing

Two parallel downstream classifiers are maintained: - **Control:** Trained on real data only (with standard augmentation) - **Treatment:** Trained on real data + DDPM-generated synthetic data

A/B tests run on a held-out validation set of 200 confirmed rare disease cases (collected prospectively). Statistical significance is assessed using McNemar's test with Bonferroni correction for multiple comparisons across 6 pathology classes.

4.7 CI/CD Pipeline

```
GitHub Push → Lint + Unit Tests → Integration Tests (generate 10 samples)
    → FID Smoke Test (< 50 on 100 samples)
    → Radiologist Spot Check (manual gate)
    → Staging Deployment → Canary (5% traffic) → Full Rollout
```

Model retraining is triggered by: - Scheduled weekly cadence - Data drift alert - New annotated rare disease cases (> 20 new cases)

4.8 Cost Analysis

Training costs (one-time): - DDPM training: 4x A100 GPUs for 72 hours = \$12,000 - Classifier retraining: 1x A100 GPU for 8 hours = \$200

Operational costs (monthly): - Inference serving: 1x A10G on-demand = \$800/month - Storage (synthetic images): S3 = \$50/month - Monitoring and logging: \$100/month - Total monthly operational cost: ~\$1,000/month

ROI: - \$15M incremental ARR opportunity / \$65K annual infrastructure cost = 230x return - Breakeven: within 2 months of deployment