Case Study: Denoising Score Matching for Synthetic Medical Image Generation

Section 1: Industry Context and Business Problem

Industry: Healthcare -- Medical Imaging

Medical imaging is one of the most data-hungry domains in AI. Training robust diagnostic models for detecting brain tumors, strokes, and neurodegenerative diseases requires vast datasets of labeled MRI scans. However, medical image data is scarce, expensive to annotate, and heavily regulated under privacy laws such as HIPAA and GDPR.

Company Profile: RadiSynth AI

RadiSynth AI is a Series A healthcare AI startup based in Boston, MA, founded by former radiologists and machine learning researchers from Massachusetts General Hospital. The company has 25 employees and has raised USD12M in funding. Their mission is to democratize access to high-quality medical imaging datasets by generating synthetic, privacy-compliant MRI brain scans that are statistically indistinguishable from real patient data.

Business Challenge

RadiSynth's hospital partners have access to approximately 8,000 labeled brain MRI scans. However, the downstream diagnostic models they are building for tumor detection require at least 50,000 diverse training samples to achieve clinical-grade accuracy. Acquiring additional real patient data would cost approximately USD2.5M and take 18 months due to IRB approvals and radiologist annotation time.

The proposed solution is to use generative AI to synthesize an additional 42,000 realistic brain MRI slices that: 1. Capture the full diversity of brain anatomy (shape, size, ventricle morphology) 2. Include realistic tumor presentations when conditioned on pathology labels 3. Pass statistical indistinguishability tests against real scans 4. Contain zero real patient information (fully synthetic)

Stakes

- **Clinical impact:** Downstream diagnostic models trained on augmented data must achieve AUC > 0.95 for tumor detection. Current models trained on 8,000 samples achieve AUC 0.89.
- **Regulatory:** Generated images must be certifiably synthetic with no memorization of real patient data. A single identifiable patient image in the training set would trigger HIPAA violations.

- **Financial:** Successfully generating synthetic training data would save USD2.5M and 18 months compared to acquiring real data.
- **Competitive:** Two competitor startups are pursuing GAN-based synthesis. RadiSynth believes score-based models offer superior sample quality and training stability.

Constraints

- Compute budget: 4 NVIDIA A100 GPUs for up to 2 weeks of training
- Resolution: 128x128 grayscale brain MRI slices (2D axial slices)
- Latency: Generation speed of at least 10 images per second at inference
- Quality: FID (Frechet Inception Distance) score below 15.0 on held-out real images
- Privacy: Nearest-neighbor distance between any generated image and the training set must exceed a threshold to prevent memorization

Section 2: Technical Problem Formulation

Problem Type: Unconditional and Class-Conditional Image Generation

This is a generative modeling problem where we learn the data distribution of brain MRI scans and sample new images from it. We use **Denoising Score Matching** to train a score network, followed by **Langevin Dynamics** (or its annealed variant) for sampling.

Why DSM over other approaches: - GANs suffer from mode collapse and training instability, especially on limited medical data - VAEs produce blurry images due to the KL divergence penalty - DSM provides stable training with a simple MSE objective and produces high-quality samples - Score-based models offer theoretical guarantees on distribution coverage (no mode collapse)

Input/Output Specifications

Training inputs: - Real MRI brain slices: tensor of shape $(N, 1, 128, 128)$, normalized to $[-1, 1]$ - Optional class labels: $y \in \{0, 1\}$ (healthy vs. tumor-present)

Generation inputs: - Random noise: $z \sim \mathcal{N}(0, I)$, shape $(B, 1, 128, 128)$ - Number of Langevin steps: T - Noise schedule: $\{\sigma_1, \sigma_2, \ldots, \sigma_L\}$

Outputs: - Synthetic MRI slices: tensor of shape $(B, 1, 128, 128)$, values in $[-1, 1]$

Mathematical Foundation from First Principles

The Score Function

Given the data distribution $p_{\text{data}}(x)$ of MRI images, the score function is:

$$s(x) = \nabla_x \log p_{\text{data}}(x)$$

This is a vector field over the image space that points toward regions of higher density. For a 128x128 image, the score is also a 128x128 "image" where each pixel value indicates how much that pixel should change to make the image more likely.

Denoising Score Matching Objective

We corrupt training images x with Gaussian noise at level σ :

$$\tilde{x} = x + \sigma\epsilon, \quad \epsilon \sim \mathcal{N}(0, I)$$

The DSM loss trains the score network s_θ to predict the direction back to clean data:

$$\mathcal{L}_{\text{DSM}}(\theta; \sigma) = \mathbb{E}_{x \sim p_{\text{data}}} \mathbb{E}_{\epsilon \sim \mathcal{N}(0,I)} \left[\left\| s_\theta(x + \sigma\epsilon, \sigma) + \frac{\epsilon}{\sigma} \right\|^2 \right]$$

Multi-Scale Noise (NCSN)

A single noise level is insufficient for high-quality generation. We use L noise levels $\sigma_1 > \sigma_2 > \cdots > \sigma_L$:

$$\mathcal{L}_{\text{NCSN}}(\theta) = \frac{1}{L} \sum_{i=1}^{L} \sigma_i^2 \cdot \mathcal{L}_{\text{DSM}}(\theta; \sigma_i)$$

The weighting by σ_i^2 ensures each noise level contributes equally to the loss.

Annealed Langevin Dynamics for Sampling

For each noise level from largest to smallest:

$$x_t^{(i)} = x_{t-1}^{(i)} + \eta_i \cdot s_\theta(x_{t-1}^{(i)}, \sigma_i) + \sqrt{2\eta_i} \cdot z_t, \quad z_t \sim \mathcal{N}(0, I)$$

where $\eta_i = \alpha \cdot \sigma_i^2 / \sigma_L^2$ and α is a base step size.

Loss Function with Per-Term Justification

The total training loss is:

$$\mathcal{L}_{\text{total}} = \frac{1}{L} \sum_{i=1}^{L} \sigma_i^2 \cdot \mathbb{E}_{x,\epsilon} \left[\left\| s_\theta(\tilde{x}, \sigma_i) + \frac{\epsilon}{\sigma_i} \right\|^2 \right]$$

Term 1: $\|s_\theta(\tilde{x}, \sigma_i) + \epsilon/\sigma_i\|^2$ This is the MSE between the predicted score and the true score of the noisy conditional distribution. It drives the network to learn the correct denoising direction at each noise level.

Term 2: σ_i^2 **weighting** Without this, the loss is dominated by small noise levels (where the score magnitude is $\sim 1/\sigma^2$). The σ_i^2 weighting normalizes the contribution across scales, ensuring the network learns coarse structure (large σ) and fine details (small σ) equally well.

Term 3: $1/L$ averaging Normalizes across the number of noise levels for stable gradient magnitudes regardless of L.

Evaluation Metrics

1. **FID Score** (Frechet Inception Distance): Measures statistical similarity between generated and real image distributions. Target: FID < 15.0.
2. **IS** (Inception Score): Measures quality and diversity. Higher is better.
3. **Precision & Recall**: Precision measures sample quality; recall measures mode coverage.
4. **Nearest Neighbor Distance**: Minimum L2 distance between each generated image and the training set. Must exceed a threshold to ensure no memorization.
5. **Downstream Task AUC**: Tumor detection AUC when a classifier is trained on real + synthetic data.

Baseline

- **Real-data-only baseline:** Classifier trained on 8,000 real images achieves AUC 0.89.
- **GAN baseline:** StyleGAN2 trained on the same data achieves FID 22.3 but suffers from mode collapse (generates limited diversity of brain shapes).
- **VAE baseline:** Standard VAE achieves FID 35.1 with visibly blurry outputs.

Why This Concept is the Right Technical Solution

Denoising Score Matching is ideal for this problem because: 1. **Training stability:** Unlike GANs, DSM uses a simple MSE loss with no adversarial dynamics 2. **Mode coverage:** Score-based models theoretically cover all modes of the distribution 3. **Quality:** State-of-the-art FID scores on image generation benchmarks 4. **Privacy guarantees:** The denoising objective naturally regularizes against memorization 5. **Conditioning:** Easy to extend to class-conditional generation by adding label information to the score network

Section 3: Implementation Notebook Structure

3.1 Data Loading and Preprocessing

```
def load_mri_dataset(data_dir: str, img_size: int = 128) -> torch.utils.data.Dataset:
    """
    Load brain MRI slices and preprocess them.

    Args:
        data_dir: path to directory containing MRI images
        img_size: target resolution (default 128x128)

    Returns:
        dataset: PyTorch Dataset yielding (image, label) tuples
                 image shape: (1, img_size, img_size), range [-1, 1]
                 label: 0 (healthy) or 1 (tumor)

    Steps:
        1. Load images from data_dir (PNG or DICOM format)
        2. Resize to img_size x img_size
        3. Normalize pixel values to [-1, 1]
        4. Return as PyTorch Dataset
    """
    # ============ TODO ============
    # Implement data loading pipeline
    # Hint: Use torchvision.transforms for resizing and normalization
    # For this case study, we use a simulated dataset
    # ==============================
    pass
```

3.2 Exploratory Data Analysis

```
def visualize_dataset_statistics(dataset):
    """
    Visualize key statistics of the MRI dataset.

    Produces:
        1. Grid of 16 sample images
        2. Pixel intensity histogram
        3. Class distribution bar chart
        4. Mean image visualization

    Args:
        dataset: PyTorch Dataset of MRI images
    """
    # ============ TODO ============
    # Create a 4-panel figure showing dataset characteristics
    # ==============================
    pass
```

3.3 Baseline Model

```
def train_pixel_noise_baseline(dataset, n_samples: int = 1000):
    """
    Generate synthetic images by adding Gaussian noise to real images.
    This is the simplest possible baseline.

    Args:
        dataset: real MRI dataset
        n_samples: number of synthetic images to generate

    Returns:
        synthetic_images: tensor of shape (n_samples, 1, 128, 128)
        fid_score: FID against the real dataset

    Steps:
        1. Randomly select real images
        2. Add Gaussian noise with sigma=0.3
        3. Clip to [-1, 1]
        4. Compute FID against real images
    """
    # ============ TODO ============
    # Implement the noise-augmentation baseline
    # ==============================
    pass
```

3.4 Score Network Architecture

```
class ConditionalScoreNetwork(nn.Module):
    """
    U-Net-based score network conditioned on noise level sigma.

    Architecture:
        - Encoder: 4 downsampling blocks (Conv + GroupNorm + SiLU)
        - Bottleneck: 2 residual blocks
        - Decoder: 4 upsampling blocks with skip connections
        - Sigma conditioning: embed sigma using sinusoidal encoding,
          add to each block via FiLM (Feature-wise Linear Modulation)

    Args:
        in_channels: 1 (grayscale MRI)
        base_channels: 64
        channel_mults: (1, 2, 4, 8)
        n_sigma_embed: 128

    Forward:
        x: (B, 1, 128, 128) noisy image
        sigma: (B,) noise level for each sample

    Returns:
        score: (B, 1, 128, 128) predicted score (same shape as input)
    """
    # ============ TODO ============
    # Implement the U-Net score network
    # Key components:
    #   - Sinusoidal sigma embedding
    #   - Encoder blocks with downsampling
    #   - Skip connections
    #   - Decoder blocks with upsampling
    #   - FiLM conditioning on sigma
    # ==============================
    pass
```

3.5 DSM Training Loop

```
def train_dsm(model, dataset, noise_levels, n_epochs=100, batch_size=32, lr=1e-4):
    """
    Train the score network using multi-scale Denoising Score Matching.

    Args:
        model: ConditionalScoreNetwork
        dataset: MRI dataset
        noise_levels: list of L sigma values (e.g., geometric sequence from 50 to 0.01)
        n_epochs: training epochs
        batch_size: samples per batch
        lr: learning rate

    Returns:
        model: trained model
        losses: list of per-epoch losses

    Training loop:
        For each batch:
            1. Sample clean images x from dataset
            2. Sample random noise level index i ~ Uniform(1, L)
            3. Sample noise epsilon ~ N(0, I)
            4. Compute noisy image: x_tilde = x + sigma_i * epsilon
            5. Predict score: s = model(x_tilde, sigma_i)
            6. Compute loss: sigma_i^2 * ||s + epsilon/sigma_i||^2
            7. Backpropagate and update
    """
    # ============ TODO ============
    # Implement the multi-scale DSM training loop
    # ==============================
    pass
```

3.6 Evaluation

```
def evaluate_generated_images(real_images, generated_images):
    """
    Compute comprehensive evaluation metrics.

    Args:
        real_images: (N_real, 1, 128, 128) real MRI images
        generated_images: (N_gen, 1, 128, 128) generated images

    Returns:
        metrics: dict with keys:
            - 'fid': Frechet Inception Distance
            - 'precision': Precision score
            - 'recall': Recall score
            - 'nn_distance': mean nearest-neighbor distance
            - 'nn_min_distance': minimum nearest-neighbor distance

    Steps:
        1. Extract features using a pretrained network
        2. Compute FID between feature distributions
        3. Compute precision and recall
        4. Compute nearest-neighbor distances for privacy check
    """
    # ============ TODO ============
    # Implement evaluation metrics
    # ==============================
    pass
```

3.7 Error Analysis

```
def analyze_generation_errors(model, real_dataset, generated_images, noise_levels):
    """
    Analyze failure modes and quality issues in generated images.

    Produces:
        1. Worst-case generated images (highest reconstruction error)
        2. Per-noise-level score prediction accuracy
        3. Frequency spectrum analysis (real vs generated)
        4. Anatomical plausibility checks

    Args:
        model: trained score network
        real_dataset: real MRI dataset
        generated_images: generated samples
        noise_levels: list of sigma values used in training
    """
    # ============ TODO ============
    # Implement error analysis pipeline
    # ==============================
    pass
```

3.8 Deployment Considerations

```
def create_inference_pipeline(model, noise_levels, device='cuda'):
    """
    Create a production-ready inference pipeline for image generation.

    Args:
        model: trained score network
        noise_levels: list of sigma values
        device: 'cuda' or 'cpu'

    Returns:
        generate_fn: callable that takes (n_samples, n_steps_per_level)
                     and returns generated images

    Optimizations:
        1. Model compilation with torch.compile
        2. Mixed precision (fp16) inference
        3. Batch generation
        4. Deterministic seeding for reproducibility
    """
    # ============ TODO ============
    # Implement production inference pipeline
    # ==============================
    pass
```

3.9 Ethical Considerations

```
def privacy_audit(generated_images, training_images, threshold=0.1):
    """
    Audit generated images for potential patient data memorization.
    Args:
        generated_images: (N_gen, 1, 128, 128) generated images
        training_images: (N_train, 1, 128, 128) training images
        threshold: minimum acceptable L2 distance

    Returns:
        audit_report: dict with:
            - 'passed': bool (True if all images pass)
            - 'min_distance': minimum distance found
            - 'flagged_indices': list of generated image indices too close to training data
            - 'distance_histogram': distribution of nearest-neighbor distances

    Steps:
        1. For each generated image, find nearest neighbor in training set
        2. Flag any image with distance below threshold
        3. Visualize flagged images alongside their nearest training neighbors
    """
    # ============ TODO ============
    # Implement privacy audit
    # ==============================
    pass
```

Section 4: Production and System Design Extension

System Architecture

The production system for RadiSynth AI consists of five main components:

1. **Training Pipeline:** Distributed training across 4 A100 GPUs using PyTorch DDP. Data loading with DICOM preprocessing, intensity normalization, and augmentation. Model checkpointing every 5 epochs.

2. **Generation Service:** FastAPI-based REST API that accepts generation requests with parameters (number of images, class label, random seed). Returns generated images as base64-encoded PNGs or DICOM files.

3. **Quality Assurance Module:** Automated post-generation quality checks including FID score monitoring, anatomical plausibility scoring (ventricle size, brain symmetry), and privacy audit.

4. **Storage and Delivery:** Generated images stored in HIPAA-compliant cloud storage (AWS S3 with encryption). Metadata stored in PostgreSQL with audit trails.

5. **Monitoring Dashboard:** Real-time metrics for generation quality, latency, and usage patterns.

API Design

```
POST /api/v1/generate
{
    "n_samples": 100,
    "class_label": "tumor",      // "healthy" or "tumor"
    "seed": 42,                   // for reproducibility
    "quality_check": true,        // run automated QA
    "output_format": "dicom"      // "png" or "dicom"
}

Response:
```

```
{
    "job_id": "gen-2026-001",
    "status": "completed",
    "n_generated": 100,
    "quality_metrics": {
        "mean_fid": 12.3,
        "privacy_passed": true,
        "anatomical_score": 0.94
    },
    "download_url": "https://storage.radisynth.ai/gen-2026-001.zip"
}
```

Model Serving

- **Framework:** TorchServe with custom handler for Langevin sampling
- **Hardware:** NVIDIA T4 GPU for inference (cost-effective)
- **Batching:** Dynamic batching with max batch size 32
- **Latency:** Target p99 latency of 500ms for a single 128x128 image
- **Throughput:** 10+ images per second on T4

Monitoring and Observability

- **Generation quality:** FID score computed hourly on a rolling window of 1,000 generated images
- **Latency metrics:** p50, p95, p99 generation latency tracked via Prometheus
- **Error rates:** Failed generations, QA rejections, API errors
- **Model health:** Score magnitude distribution, gradient norms during fine-tuning
- **Privacy alerts:** Instant notification if any generated image has nearest-neighbor distance below threshold

Data Drift Detection

- **Feature drift:** Monitor the distribution of extracted features (InceptionV3) from generated images. Use Maximum Mean Discrepancy (MMD) between recent generations and a reference set.
- **Score drift:** Track the mean score magnitude at each noise level. Significant changes indicate distribution shift in the training data.
- **Trigger:** If MMD exceeds threshold for 3 consecutive hours, alert the team and pause generation.

A/B Testing

- **Approach:** When training a new model version, run both old and new models in parallel.
- **Metrics:** Compare FID, precision, recall, downstream classifier AUC, and radiologist preference scores.
- **Traffic split:** 90% old model / 10% new model initially, gradually shifting based on metrics.
- **Duration:** Minimum 1 week of parallel operation before full rollover.

CI/CD Pipeline

1. **Model training:** Triggered by new data ingestion or hyperparameter changes. Runs on dedicated GPU cluster.
2. **Automated evaluation:** After training, compute FID, IS, precision/recall on held-out test set. Gate: FID must be within 10% of best historical score.
3. **Privacy audit:** Automated nearest-neighbor check against full training set. Gate: zero flagged images.
4. **Staging deployment:** Deploy to staging environment, run 1,000 test generations.
5. **Radiologist review:** 50 randomly selected generated images reviewed by a board-certified radiologist. Gate: anatomical plausibility score >= 0.90.
6. **Production deployment:** Blue-green deployment with automatic rollback if quality metrics degrade.

Cost Analysis

- **Training cost:** 4 A100 GPUs x 2 weeks = approximately USD8,000 on cloud compute
- **Inference cost:** 1 T4 GPU instance at approximately USD0.35/hour. At 10 images/second, generating 42,000 images takes approximately 70 minutes, costing approximately USD0.50.
- **Total project cost:** approximately USD15,000 (including engineering time) vs. USD2.5M for real data acquisition.
- **Ongoing cost:** approximately USD300/month for the inference API (assuming 100,000 images/month).