Case Study: Accelerated MRI Reconstruction Using Score-Based Generative Priors

Section 1: Industry Context and Business Problem

Industry: Medical Imaging / Radiology

Magnetic Resonance Imaging (MRI) is one of the most valuable diagnostic tools in modern medicine, providing high-resolution soft-tissue contrast without ionizing radiation. However, MRI has a fundamental bottleneck: scan time. A standard knee MRI takes 20-30 minutes, a brain MRI takes 30-45 minutes, and cardiac MRI can take over an hour. During this time, the patient must remain perfectly still -- a particular challenge for pediatric patients, elderly patients with pain, and emergency cases.

The physics behind MRI imposes this constraint. An MRI scanner samples the frequency domain (called **k-space**) of the image, and acquiring enough k-space samples for a high-quality reconstruction requires many sequential measurements. Reducing the number of measurements (called **undersampling**) proportionally reduces scan time but introduces aliasing artifacts in the reconstructed image.

Company Profile: MedScanAI

MedScanAI is a fictional medical AI startup based in Boston, Massachusetts, founded by former researchers from Massachusetts General Hospital and MIT CSAIL. The company focuses on AI-accelerated medical imaging pipelines for hospital radiology departments. Their flagship product, **ReconPrior**, is a software module that integrates with existing MRI scanner hardware to enable high-quality image reconstruction from undersampled k-space data, reducing scan times by 4-8x.

Business Challenge

MedScanAI has been approached by a network of 12 community hospitals in the Midwest that collectively perform over 200,000 MRI scans per year. The hospitals face three critical problems:

1. **Patient throughput:** Each MRI scanner costs \$1.5M-\$3M and can only serve 15-20 patients per day due to long scan times. Waitlists average 3-4 weeks.
2. **Motion artifacts:** Approximately 15% of scans require repeat acquisitions due to patient motion, costing the network an estimated \$8M per year in lost scanner time and additional radiologist review.

3. **Emergency bottleneck:** Trauma patients requiring brain or spinal MRI must wait for scheduled slots, delaying diagnosis by an average of 6 hours.

If MedScanAI can reduce scan times by 4x while maintaining diagnostic image quality (as measured by radiologist blind assessments), the hospital network projects: - \$12M annual savings from increased throughput - 60% reduction in motion-related repeat scans - Emergency MRI availability within 30 minutes

Constraints

- The reconstruction model must run on hospital-grade NVIDIA A100 GPUs (available in existing AI infrastructure)
- Reconstruction time must be under 60 seconds per image volume
- Image quality must meet ACR (American College of Radiology) accreditation standards
- The system must be FDA 510(k) clearable (predicate devices exist for AI-assisted reconstruction)
- Patient data privacy (HIPAA compliance) requires all processing to occur on-premises

---

Section 2: Technical Problem Formulation

Problem Type: Inverse Problem with Learned Prior

MRI reconstruction from undersampled k-space data is a **linear inverse problem**. The forward model is:

$$y = Ax + n$$

where: - $x \in \mathbb{R}^N$ is the target MRI image (e.g., $N = 320 \times 320$) - $A \in \mathbb{C}^{M \times N}$ is the undersampled Fourier encoding matrix ($M \ll N$ for acceleration) - $y \in \mathbb{C}^M$ is the measured k-space data - n is acquisition noise

The acceleration factor $R = N/M$ determines the scan time reduction. At $R = 4$, we acquire only 25% of k-space, reducing scan time by 4x.

Why is this underdetermined? With $M < N$, the system $y = Ax$ has infinitely many solutions. We need a **prior** -- knowledge about what MRI images look like -- to select the correct solution from the infinite set.

Why Score-Based Priors?

Traditional compressed sensing uses hand-crafted priors like sparsity in wavelets. Deep learning methods (e.g., U-Net) learn a direct mapping from undersampled to fully-sampled images but often produce hallucinated details or fail on out-of-distribution anatomy.

Score-based generative priors offer a principled alternative. Instead of learning a direct mapping, we train a Noise Conditioned Score Network to learn the probability distribution of MRI images. Then, during reconstruction, we combine this learned prior with the physics-based data consistency constraint.

The reconstruction becomes a posterior sampling problem:

$$p(x|y) \propto p(y|x) \cdot p(x)$$

- $p(y|x)$: likelihood (physics-based, known from the forward model)
- $p(x)$: prior (learned by NCSN)
- $p(x|y)$: posterior (what we want to sample from)

Using the score function, the reconstruction alternates between: 1. **Score update:** Take a step toward high-probability images using $\nabla_x \log p(x)$ 2. **Data consistency:** Project back onto the set of images consistent with measurements y

Input/Output Specification

Input: - Undersampled k-space data y : complex-valued tensor of shape (C, H, W_{sub}) where C is the number of coil channels, H is the full k-space height, and $W_{sub} = W/R$ is the undersampled width - Sampling mask M : binary tensor of shape (H, W) indicating which k-space lines were acquired - Acceleration factor R : integer (typically 4 or 8)

Output: - Reconstructed MRI image $\hat{x}$: real-valued tensor of shape (H, W) , magnitude image - Uncertainty map (optional): pixel-wise standard deviation from multiple posterior samples

Mathematical Foundation

Score function of MRI images:

The NCSN learns $s_\theta(x, \sigma) \approx \nabla_x \log q_\sigma(x)$ where q_σ is the distribution of MRI images perturbed with Gaussian noise of level σ .

Worked example: Consider a single pixel with value $x_0 = 0.7$ (normalized intensity). At noise level $\sigma = 0.1$, the noisy pixel is $\tilde{x} = 0.7 + 0.1 \times 0.3 = 0.73$ (where $\epsilon = 0.3$). The score target is $-\epsilon/\sigma = -0.3/0.1 = -3.0$. This tells the network: "at this noisy value, move left (toward lower intensity) with strength 3.0."

Reconstruction via Annealed Langevin Dynamics with Data Consistency:

At each step of ALD, we add a data consistency projection:

$$x_{t+1} = x_t + \alpha_i s_\theta(x_t, \sigma_i) + \sqrt{2\alpha_i} z_t$$

$x_{t+1} \leftarrow x_{t+1} - \lambda A^H (A x_{t+1} - y)$

The second line ensures the reconstruction is consistent with the measured k-space data.

Worked example: Suppose after the score update, a reconstructed k-space value at a measured frequency is $\hat{y}_k = 0.5 + 0.3j$ but the actual measurement is $y_k = 0.4 + 0.2j$. The data consistency correction replaces $\hat{y}_k$ with the measured value y_k at measured positions, and keeps the predicted value at unmeasured positions.

Loss Function

The NCSN is trained with the standard multi-scale denoising score matching loss:

$$\mathcal{L}(\theta) = \frac{1}{L} \sum_{i=1}^{L} \sigma_i^2 \mathbb{E}_{x \sim p_{data}} \mathbb{E}_{\epsilon \sim \mathcal{N}(0,I)} \left\| s_\theta(x + \sigma_i \epsilon, \sigma_i) + \frac{\epsilon}{\sigma_i} \right\|^2$$

Term-by-term justification:

1. $s_\theta(x + \sigma_i \epsilon, \sigma_i)$: The network's score prediction for noisy input at noise level σ_i
2. ϵ/σ_i : The target score derived from the denoising score matching identity
3. σ_i^2 : Weighting factor ensuring balanced gradients across noise levels
4. $1/L$: Averaging over noise levels for stable training
5. Expectations: Averaged over the training data distribution and noise realizations

Evaluation Metrics

Metric	Description	Target
SSIM	Structural similarity to fully-sampled reference	> 0.92 at R=4
PSNR	Peak signal-to-noise ratio	> 33 dB at R=4
NMSE	Normalized mean squared error	< 0.02 at R=4
FID	Frechet Inception Distance (distribution quality)	< 30
Radiologist score	Blind assessment on 1-5 Likert scale	> 4.0
Reconstruction time	Wall-clock time per image slice	< 5 seconds

Baseline

- **Zero-filled reconstruction:** Simple inverse FFT of undersampled k-space (SSIM ~0.65 at R=4)
- **Compressed sensing (TV regularization):** Iterative reconstruction with total variation prior (SSIM ~0.82 at R=4)
- **U-Net direct mapping:** Supervised deep learning baseline (SSIM ~0.89 at R=4)
- **NCSN with ALD + data consistency:** Our approach (target SSIM > 0.92 at R=4)

Section 3: Implementation Notebook Structure

3.1 Data Loading and Preprocessing

```
def load_fastmri_dataset(data_dir, challenge='singlecoil', split='train'):
    """
    Load the fastMRI dataset for training.

    Args:
        data_dir: path to fastMRI data directory
        challenge: 'singlecoil' or 'multicoil'
        split: 'train', 'val', or 'test'

    Returns:
        Dataset object yielding (image, kspace, mask) tuples

    Notes:
        - fastMRI knee dataset: ~1,600 volumes, ~35,000 slices
        - Images normalized to [0, 1] range
        - k-space data is complex-valued
    """
    # TODO: Implement fastMRI data loading
    # Hint: Use h5py to read .h5 files
    # Each file contains: 'kspace' (complex), 'reconstruction_rss' (magnitude)
    pass
```

```
def create_undersampling_mask(shape, acceleration, center_fraction=0.08):
    """
    Create a random undersampling mask for k-space.

    Args:
        shape: (H, W) image dimensions
        acceleration: acceleration factor R (e.g., 4 or 8)
        center_fraction: fraction of center k-space lines to always acquire

    Returns:
        Binary mask of shape (H, W)

    Notes:
        - Always acquire the center_fraction of k-space (low frequencies)
        - Randomly sample remaining lines to achieve target acceleration
    """
    # TODO: Implement undersampling mask creation
    pass
```

3.2 Exploratory Data Analysis

```
def visualize_kspace_and_reconstruction(kspace, mask, acceleration):
    """
    Visualize the effect of undersampling on MRI reconstruction.

    Create a 2x3 grid showing:
        Row 1: Full k-space | Mask | Undersampled k-space
        Row 2: Full reconstruction | Zero-filled recon | Difference map

    Args:
        kspace: complex k-space data (H, W)
        mask: binary undersampling mask (H, W)
        acceleration: acceleration factor for title
    """
    # TODO: Implement visualization
    # Hint: Use np.fft.ifft2 for reconstruction, np.abs for magnitude
    pass
```

3.3 Baseline: Zero-Filled and Compressed Sensing

```
def zero_filled_reconstruction(kspace, mask):
    """
    Simplest baseline: inverse FFT of undersampled k-space.

    Args:
        kspace: full k-space data (H, W), complex
        mask: binary mask (H, W)

    Returns:
        Magnitude image (H, W)
    """
    # TODO: Implement zero-filled reconstruction
    pass
```

```
def compressed_sensing_reconstruction(kspace, mask, n_iterations=50, lambd=0.01):
    """
    Compressed sensing with Total Variation regularization.

    Solves: argmin_x ||Ax - y||^2 + lambda * TV(x)

    Args:
        kspace: measured k-space (H, W), complex
        mask: sampling mask (H, W)
        n_iterations: ISTA iterations
        lambd: TV regularization weight

    Returns:
        Reconstructed magnitude image (H, W)

    Hints:
        - Use proximal gradient descent (ISTA)
        - TV(x) = sum of |gradient_x| + |gradient_y|
        - Proximal operator of TV is soft-thresholding of image gradients
    """
    # TODO: Implement CS-TV reconstruction
    pass
```

3.4 Model Architecture: U-Net Score Network

```
class UNetScoreNetwork(nn.Module):
    """
    U-Net based Noise Conditioned Score Network for MRI images.

    Architecture:
        - Encoder: 4 downsampling blocks with ResNet blocks
        - Bottleneck: 2 ResNet blocks
        - Decoder: 4 upsampling blocks with skip connections
        - Noise level conditioning via FiLM (Feature-wise Linear Modulation)

    Args:
        in_channels: number of input channels (1 for magnitude, 2 for complex)
        base_channels: base channel count (doubled at each level)
        n_noise_levels: number of discrete noise levels for embedding
    """
    def __init__(self, in_channels=1, base_channels=64, n_noise_levels=100):
        super().__init__()
        # TODO: Implement U-Net architecture with noise conditioning
        # Hint: Use nn.Embedding for noise level -> vector
        # Apply FiLM: scale * features + bias at each layer
        pass

    def forward(self, x, sigma_idx):
        """
        Args:
            x: noisy image (B, 1, H, W)
            sigma_idx: noise level index (B,) integer

        Returns:
            Predicted score (B, 1, H, W)
```

```
        """
        # TODO: Implement forward pass
        pass
```

3.5 Training Loop

```
def train_ncsn_mri(model, dataloader, sigmas, n_epochs=100, lr=1e-4):
    """
    Train NCSN on MRI images.

    Args:
        model: UNetScoreNetwork
        dataloader: yields batches of MRI images (B, 1, H, W)
        sigmas: noise levels tensor (L,)
        n_epochs: training epochs
        lr: learning rate

    Training procedure per batch:
        1. Sample random noise level index for each image
        2. Add noise: x_noisy = x + sigma * epsilon
        3. Target: -epsilon / sigma
        4. Loss: sigma^2 * ||model(x_noisy, sigma_idx) - target||^2

    Returns:
        Training loss history
    """
    # TODO: Implement training loop
    # Hint: Use mixed precision (torch.cuda.amp) for speed
    # Log loss every epoch
    pass
```

3.6 Evaluation: Reconstruction Quality

```
def reconstruct_with_ncsn(model, kspace, mask, sigmas, n_steps=100, eps=5e-5):
    """
    Reconstruct MRI from undersampled k-space using NCSN + ALD + data consistency.

    Args:
        model: trained UNetScoreNetwork
        kspace: measured k-space (H, W), complex
        mask: sampling mask (H, W)
        sigmas: noise levels (L,)
        n_steps: Langevin steps per noise level
        eps: base step size

    Returns:
        Reconstructed image (H, W)

    Algorithm:
        1. Initialize x from zero-filled reconstruction
        2. For each sigma (large to small):
            a. Compute alpha = eps * (sigma / sigma_L)^2
            b. For n_steps:
                i.   Score update: x += alpha * model(x, sigma) + sqrt(2*alpha) * z
                ii.  Data consistency: replace measured k-space lines
        3. Return |x| (magnitude)
    """
    # TODO: Implement NCSN reconstruction with data consistency
    pass
```

```
def evaluate_reconstruction(pred, target):
    """
    Compute reconstruction quality metrics.

    Args:
        pred: reconstructed image (H, W)
        target: fully-sampled reference (H, W)

    Returns:
        dict with SSIM, PSNR, NMSE values
```

```
    """
    # TODO: Implement metric computation
    # Hint: Use skimage.metrics.structural_similarity for SSIM
    pass
```

3.7 Error Analysis

```
def analyze_reconstruction_errors(model, test_dataset, sigmas, accelerations=[4, 8]):
    """
    Systematic error analysis across acceleration factors and anatomy types.

    For each acceleration factor:
        1. Reconstruct all test images
        2. Compute per-image metrics
        3. Identify failure cases (SSIM < 0.85)
        4. Visualize worst-case reconstructions

    Generate:
        - Box plots of SSIM/PSNR by acceleration factor
        - Error maps for best/worst cases
        - Frequency analysis of errors (which k-space regions cause most issues)
    """
    # TODO: Implement error analysis
    pass
```

3.8 Deployment Considerations

```
def benchmark_inference_speed(model, image_size=(320, 320), n_noise_levels=10,
                              n_steps=50, device='cuda'):
    """
    Benchmark reconstruction speed for deployment readiness.

    Measure:
        - Single-slice reconstruction time
        - GPU memory usage
        - Throughput (slices per second)

    Target: < 5 seconds per slice on NVIDIA A100
    """
    # TODO: Implement speed benchmarking
    pass
```

```
def export_model_onnx(model, save_path, image_size=(320, 320)):
    """
    Export trained model to ONNX format for deployment.

    Args:
        model: trained UNetScoreNetwork
        save_path: output .onnx file path
        image_size: expected input image size
    """
    # TODO: Implement ONNX export
    # Hint: torch.onnx.export with dynamic axes for batch size
    pass
```

3.9 Ethics and Fairness

```
def audit_demographic_performance(model, test_dataset, sigmas):
    """
    Audit reconstruction quality across demographic groups.

    Check for disparities in:
        - Body habitus (BMI categories: underweight, normal, overweight, obese)
        - Age groups (pediatric, adult, geriatric)
        - Sex (male, female)
        - Scan type (knee, brain, cardiac)

    For each subgroup:
```

```
        - Compute mean and std of SSIM, PSNR
        - Flag any subgroup with mean SSIM < 0.90
        - Generate fairness report

    FDA requires demonstration of consistent performance across patient populations.
    """
    # TODO: Implement demographic audit
    pass
```

Section 4: Production and System Design Extension

System Architecture

The ReconPrior system integrates with the hospital's existing MRI workflow:

```
MRI Scanner --> DICOM Server --> ReconPrior Engine --> PACS
                                      |
                                      v
                               [NVIDIA A100 GPU]
                                      |
                        +-------------+-------------+
                        |             |             |
                   Score Model   Data Consistency  QA Module
                   (NCSN)        (k-space)         (metrics)
```

1. **DICOM Receiver:** Listens for incoming undersampled k-space data from the scanner
2. **Preprocessing:** Coil combination, normalization, mask extraction
3. **NCSN Reconstruction:** Annealed Langevin Dynamics with data consistency
4. **Quality Assurance:** Automated SSIM/PSNR check against expected ranges
5. **DICOM Packaging:** Reconstructed images wrapped in DICOM format with proper metadata
6. **PACS Upload:** Images sent to the hospital's Picture Archiving system for radiologist review

API Design

```
POST /api/v1/reconstruct
    Body: {
        "kspace": base64_encoded_complex_array,
        "mask": base64_encoded_binary_array,
        "acceleration": 4,
        "n_samples": 1,  // >1 for uncertainty estimation
        "priority": "normal"  // or "emergency"
    }
    Response: {
        "image": base64_encoded_magnitude_image,
        "uncertainty_map": base64_encoded_uncertainty,  // if n_samples > 1
        "metrics": {"ssim_estimate": 0.94, "snr": 35.2},
        "processing_time_ms": 3200,
        "model_version": "v2.1.0"
    }
```

Serving Infrastructure

- **GPU Cluster:** 4x NVIDIA A100 (80GB) with NVLink
- **Model Server:** NVIDIA Triton Inference Server with dynamic batching

- **Queue System:** Redis-based priority queue (emergency scans jump the queue)
- **Latency SLA:** p50 < 3s, p99 < 10s per slice
- **Throughput:** 200+ slices/minute across the cluster

Monitoring and Drift Detection

Real-time metrics (Prometheus + Grafana): - Reconstruction latency per slice - GPU utilization and memory - Queue depth and wait times - Automated SSIM estimates (using the noise-free center of k-space as reference)

Drift detection: - Track distribution of predicted score norms over time - Monitor SSIM distribution (rolling 7-day window) - Alert if mean SSIM drops below 0.90 on a scanner-by-scanner basis - Potential drift sources: scanner hardware upgrades, new pulse sequences, coil changes

Retraining trigger: If the 7-day rolling mean SSIM drops below 0.91 on any scanner, flag for retraining with recent data from that scanner.

A/B Testing Framework

- **Randomization:** 50% of non-emergency scans processed with both current and candidate models
- **Primary metric:** Radiologist preference in blinded side-by-side comparison (N=200 per test)
- **Secondary metrics:** SSIM, PSNR, NMSE against fully-sampled reference (subset of scans)
- **Minimum detectable effect:** 0.5 point improvement on Likert scale with 80% power
- **Safety guardrail:** Automatic rollback if any reconstruction fails QA metrics check

CI/CD Pipeline

```
Code Push --> Unit Tests --> Integration Tests --> Model Training
    |              |                |                    |
    |         Score function    k-space consistency   Train on subset
    |         correctness       verification          (1000 slices)
    |              |                |                    |
    v              v                v                    v
Linting     Reconstruction    DICOM formatting      Metric check
             quality check                          (SSIM > 0.90)
                                                         |
                                                         v
                                                   Staging deploy
                                                         |
                                                         v
                                                   Canary (5% traffic)
                                                         |
                                                         v
                                                   Full rollout
```

Cost Analysis

Component	Monthly Cost
4x A100 GPU instances (on-prem amortized)	\$8,000
Storage (DICOM + models)	\$500
Monitoring infrastructure	\$200
Engineering support (0.5 FTE)	\$7,500
Total monthly	**\$16,200**

ROI: At \$12M annual savings for the hospital network, the \$194K annual system cost yields a 61x return on investment. Even accounting for MedScanAI's licensing fees (projected at \$500K/year), the hospital network achieves a 24x ROI.