Goal of the project: Use stochastic differential equations to remove high level multiplicative noise from texture rich images.

Week-1: Related Works

Key Ideas (that can be implemented):

1. Multiplicative Noise as GBM in Log domain:

   -> Perception based multiplicative noise removal

2. Converting Multiplicative Noise into an Additive Noise using Log:

   -> log(a*b) = log(a) + log(b)
   -> A new two-step variational model for multiplicative noise removal with applications to texture images

Order + Main Methodology Used:

1. **A new two-step variational model for multiplicative noise removal with applications to texture images:**

   Date: 1 September 2024

   Proposes a **two-step variational model**:

   Step 1: Use a **logarithmic transform** to convert multiplicative noise into an additive form, followed by **tangential field smoothing** to propagate isophote directions.

   Step 2: Restore the image by fitting it to the constructed isophote directions.

2. **Perception based multiplicative noise removal:**

   Date: 19 Aug 2024

Novel mathematical modeling: The paper innovatively treats multiplicative noise as a *Geometric Brownian Motion* in the log domain, and leverages the Fokker-Planck equation to derive a principled reverse process for denoising.

Strong empirical validation: The approach is benchmarked against both classical signal processing methods and modern CNN-based models, showing superior results on *perception-based metrics* (FID, LPIPS) while staying competitive on traditional ones (PSNR, SSIM).

https://en.wikipedia.org/wiki/Fokker%E2%80%93Planck_equation

3. **Image Restoration with Mean-Reverting SDE:**

   Date: 2023

   Introduces a **mean-reverting SDE** that degrades high-quality images with fixed Gaussian noise, and then leverages the **reverse-time SDE** (learned via neural networks and maximum likelihood objective) to restore images without task-specific priors.

   Provides a **closed-form solution for the forward SDE**, enabling exact computation of the time-dependent score and a stable training procedure for image restoration tasks.

   The degradation process relies on **fixed Gaussian noise** in the mean-reverting SDE framework.

4. **Stimulating the Diffusion Model for Image Denoising via Adaptive Embedding and Ensembling:**

   Date: 2023

   Introduction of a novel strategy called Diffusion Model

   Works for Gaussian as well as real-world denoising problems

5. **Multiplicative Gaussian Noise Removal using Partial Differential Equations and Activation Functions: A Robust and Stable Approach:**

   Date: 2023

   Introduces a PDE-based denoising framework enhanced with activation functions and a gray-level indicator matrix, focusing on learning the kernels explicitly for effective restoration.

Explores different activation functions within PDE schemes to optimize performance, demonstrating improvements in PSNR and SSIM over existing methods.

Involves Multiplicative Gaussian Noise (non-linear and signal-dependent)

6. **How Does Noise Help Robustness? Explanation and Exploration under the Neural SDE Framework: (Not that related)**

   Date: 2020

   Extends **Neural ODEs into Neural SDEs**, introducing **stochasticity** via random noise injection to provide a continuous-time analog of regularization mechanisms like dropout and noise-based methods.

   Incorporates **multiple noise patterns** (dropout, additive, multiplicative) within the Neural SDE framework, supported by theoretical analysis showing improved robustness and generalization.

7. **Fourth-order partial differential equations for noise removal:**

   Date: 31 October 2000

   Proposes a **fourth-order PDE framework** that minimizes a cost functional based on the Laplacian of image intensity, enabling a balance between **noise removal and edge preservation**.

   Unlike second-order anisotropic diffusion (which produces step-like, blocky effects), the method approximates images as **piecewise planar**, leading to more natural reconstructions while still suppressing noise.

   Focuses on **speckle noise** (a form of multiplicative noise commonly affecting images).

8. **Multistep methods for SDEs and their application to problems with small noise:**

   Develops **stochastic linear multistep methods** for SDEs, with a detailed analysis of **mean-square consistency, stability, and convergence**.

   Provides specialized results for **two-step Maruyama schemes**, including conditions for consistency, and derives **local error expansions** in the small-noise regime.

Derivative Works + Related Works:

1. **MaRS: A Fast Sampler for Mean Reverting Diffusion based on ODE and SDE Solvers:**

   Date: 24 March 2025

   MaRS **(MR Sampler)**, which accelerates **Mean Reverting (MR) Diffusion** by solving the reverse-time SDE and associated PF-ODE with **semi-analytical solutions** (analytical part + neural network–parameterized integral).

   Achieves **training-free fast sampling** (10–20× speedup) while supporting multiple parameterizations (noise, data, velocity prediction), making controllable diffusion generation more practical.

2. **Diffusion Normalizing Flow:**

   Date: 14 Oct 2021

   Uses **two neural SDEs** — a forward SDE (adding noise: Gaussian) and a backward SDE (removing noise: data distribution), trained jointly with a common cost function.

   Gaussian Noise involved

3. **Score-Based Generative Modeling through SDE:**

   Date: 26 Nov 2020

   Proposes a forward SDE (data: prior via gradual noise injection) and a reverse-time SDE (prior: data via noise removal), with the reverse process guided by the score function (gradient of log-density) estimated using neural networks.
   Introduces a predictor-corrector framework to refine discretized reverse SDE sampling, and derives an equivalent neural ODE that enables exact likelihood estimation and improved efficiency.

   Gaussian Noise involved

4. **Exponential ergodicity of the solutions to SDE's with a jump noise**

Chronological Order:

- **2000:** PDE foundations
- **2010s:** Multistep SDE methods
- **2020–2021:** Neural SDEs, score-based generative modeling, diffusion normalizing flows
- **2023:** Mean-reverting SDEs, diffusion ensembles, PDE+activations
- **2024:** Variational & perception-based (GBM + Fokker–Planck)
- **2025:** MaRS fast sampler
- **Side branch:** Exponential ergodicity of SDEs with jump noise

Thematic Order:

1. Core Theory

    - Fokker–Planck Equation (statistical evolution of SDEs)
    - Exponential ergodicity of SDEs with jump noise

2. PDE & SDE Methods

    - Early PDE foundations (2000s)
    - Multistep SDE numerical methods (2010s)
    - Activation-based PDEs (2023)

3. Generative Models

    - Neural SDEs (2020)
    - Score-based generative modeling (2021)
    - Diffusion normalizing flows (2021)
    - MaRS fast sampler (2025)

4. Variational & Statistical

    - Variational approaches (2024)
    - GBM + Fokker–Planck for perception modeling (2024)

5. Specialized Applications

- Mean-reverting SDEs (2023)
- Diffusion ensembles (2023)

Week-2: Limitations

1. SDE-based Multiplicative Noise Removal:

Iterative Denoising Requirement

- Because it is score-based, the method cannot directly clean the image.
- It requires iterative denoising after adding noise during training, which makes it slower than direct restoration approaches.

High Computational Cost

- This means the method remains computationally expensive due to many required diffusion steps.

Bias in noise statistics

- After the log transform, the noise becomes approximately additive Gaussian, but not perfectly.
- This “approximation” may cause a mismatch between the assumed model and the true noise distribution.

Distribution vs. Exact Recovery

- The method produces results that are close to the clean image in distribution, but not the exact ground-truth image.
- This limitation is tied to the current diffusion loss, and the authors suggest that conditioning approaches like Doob's h-transform may improve exact recovery in future work.

Week-3: Importance of Each Layer

DataLoading + Processing:

-> SquarePad -> Grayscale -> Resize and Center Crop -> Transform Noise

1. Overview

- **Forward Process** (ddpm_utils/sde.py):
  The clean image is gradually corrupted with multiplicative noise, simulated as an SDE. This step defines how images evolve from clean: noisy.
- **Reverse Process** (denoise_sde.py + train_denoise_sde.py):
  A neural network (UNet backbone) is trained to learn the reverse dynamics of the SDE, i.e., reconstructing the clean image from noisy samples.

Unlike classical Gaussian additive noise models (e.g., DDPM), this approach is explicitly adapted for multiplicative noise, making it suitable for SAR imagery and low-light imaging where noise is signal-dependent.

2. UNet Backbone Architecture

The network backbone is defined in models/unet.py and instantiated in train_denoise_sde.py.

It follows a UNet-style encoder-decoder architecture with skip connections:

- Input Layer (unet-in-ch):
  First convolutional layer, directly processes noisy input images.
  - Location: models/unet.py: UNetModel.forward()
  - Role: Preserves raw pixel-level features before downsampling.
- Downsampling Blocks (unet-num-res):
  Multiple convolutional + residual blocks reduce image resolution while expanding feature depth.
  - Location: models/unet.py: DownBlock
  - Role: Capture local textures (edges, small details). Residual connections help stabilize training.
- Bottleneck:
  Deepest representation of the image at lowest resolution.
  - Location: models/unet.py: MiddleBlock
  - Role: Encodes global context and noise patterns across the entire image.
- Upsampling Blocks:
  Mirror of downsampling, gradually restoring spatial resolution.
  - Location: models/unet.py: UpBlock
  - Role: Merge global features from the bottleneck with local details from skip connections. Restores high-frequency details.
- Output Layer (unet-out-ch):
  Final convolution maps features back to the original number of channels (RGB or grayscale).
  - Location: models/unet.py: UNetModel.forward() (final conv)
  - Role: Produces the denoised image.

3. Noise & Time Embedding Layers

Because the model is trained in a diffusion-like process, it requires time-step embeddings that encode how much noise is present at a given step.

- Location: models/unet.py: timestep_embedding()
- Role: Provides the network with explicit information about the current noise level / SDE step, guiding the UNet on how aggressive the denoising should be.

4. Training-Specific Layers and Loss Functions

The training pipeline is managed in train_denoise_sde.py.

The network output is compared with clean images using several losses:

- L2 / MSE Loss (pixel-wise accuracy).
- Perceptual Loss (LPIPS) (visual similarity, using pretrained feature extractors).
- FID Metric (realism of generated images, implemented externally via pytorch-fid).

These losses ensure that the network learns both pixel-level reconstruction accuracy and perceptual quality.

5. Denoising Process

During testing (see denoise_sde.py), the model applies the reverse SDE to gradually remove noise.

Variants supported:

- Stochastic Reverse SDE – follows the probabilistic reverse dynamics.
- Deterministic ODE (--deterministic) – faster, removes randomness.
- DDIM Variant (--ddim) – even fewer steps required, trades off between speed and quality.

6. Why UNet?

The repository uses UNet because it effectively combines local detail preservation and global structure modeling:

- Local detail layers (downsampling + skip connections) capture fine-grained structures such as edges and textures.
- Global context layers (bottleneck + upsampling) capture large-scale structures.

Summary

Layer	Importance

Input Conv	Captures raw pixel data
Downsampling	Extracts local textures, reduces resolution
Bottleneck	Encodes global structure & noise patterns
Upsampling	Restores resolution, merges fine details
Output Conv	Generates final clean image
Skip Connections	Preserve spatial details lost in downsampling
Time Embeddings	Inform network of noise level
Residual Blocks	Improve stability & allow deeper networks

General Training Setup

- python train_denoise_sde.py: runs your training script.
- img-size 224: input images are resized to 224×224 pixels.
- batch-size 32: number of images per training batch = 32.
- img-dir ./datasets/img_celeba: dataset location, here it's the CelebA dataset (celebrity faces).
- img-data-amount large: indicates you are using the large version of the dataset (maybe high-res subset).
- device cuda:0: run on GPU 0. If unavailable, it may fall back to CPU.
- cpu-workers 5: number of parallel CPU workers for data loading = 5.
- exp-name noise-removal-celeba-large-RGB-224-500: custom experiment name for logging and saved checkpoints.
- epochs 100: train for 100 full passes through the dataset.
- save-every 1: save a checkpoint after every epoch.
- unet-channels 128: base number of feature channels in UNet layers = 128.
- unet-in-ch 3: input channels = 3 (RGB images).
- unet-out-ch 3: output channels = 3 (predicts RGB denoised images).
- unet-num-res 2: number of residual blocks per UNet level = 2.
- ddpm-num-steps 500: number of diffusion time steps = 500.
  (This controls how gradually noise is added/removed during training.)
- lr 1e-6: learning rate = 0.000001 (very small, stable for diffusion models).

Generating Testing Data:

- python generate_test_data.py: runs your test data generation script.
- img-size 224: input images are resized to 224×224 pixels.
- batch-size 16: number of images per batch during test data generation = 16.
- dataset celeba: specifies the dataset = CelebA (celebrity faces).
- img-dir ./datasets/img_celeba: dataset location, path to CelebA images.
- ddpm-num-steps 500: total number of diffusion steps used in the DDPM process = 500.
- ddpm-target-steps 150: reduces the full 500-step process to 150 steps when preparing the test data.
  (This is used for faster sampling or generating intermediate noisy images at step 150).
- img-data-amount test: specifies that you're preparing the test split of the dataset.
- img-out ./datasets/celeba_prepared/img_celeba_test_t150: output path where the prepared test imageswill be saved, with noise corresponding to 150 diffusion steps.

Testing SDE (Stochastic):

- python denoise_sde.py: runs your denoising script using a trained SDE model.
- img-size 224: input images are resized to 224×224 pixels before denoising.
- -batch-size 8: number of images processed in one batch = 8.
- dataset landuse: dataset being used is LandUse (satellite/remote-sensing style dataset).
- img-dir ./datasets/landuse_prepared/img_landuse_all_t100: path to the input dataset; here the LandUse images have already been diffused up to 100 steps (so they are partially noisy).
- ddpm-num-steps 500: total diffusion schedule = 500 steps (same as training).
- ddpm-target-steps 100: tells the model to start denoising from step 100, since your input images are already noisy up to 100 steps.
- img-out ./test_images/landuse_all/denoised_images_landuse: folder where the denoised output images will be saved.
- model-path ./models/sde_model.pkl: path to the trained SDE denoising model (UNet) that will perform the denoising.

Week-4: Training on SAR Dataset + Tried some improvement

Dataset Used:
https://www.kaggle.com/datasets/requiemonk/sentinel12-image-pairs-segregated-by-terrain

UNet Conditioning

What I added: a small extra input (a short vector of numbers) to the UNet in unet_ref.py so the network can "know" simple properties of the image it's denoising.
Why it helps: instead of treating every image the same, the network can adapt its behavior depending on image characteristics (brightness, how noisy or textured it is, etc.), which improves denoising across different image types.

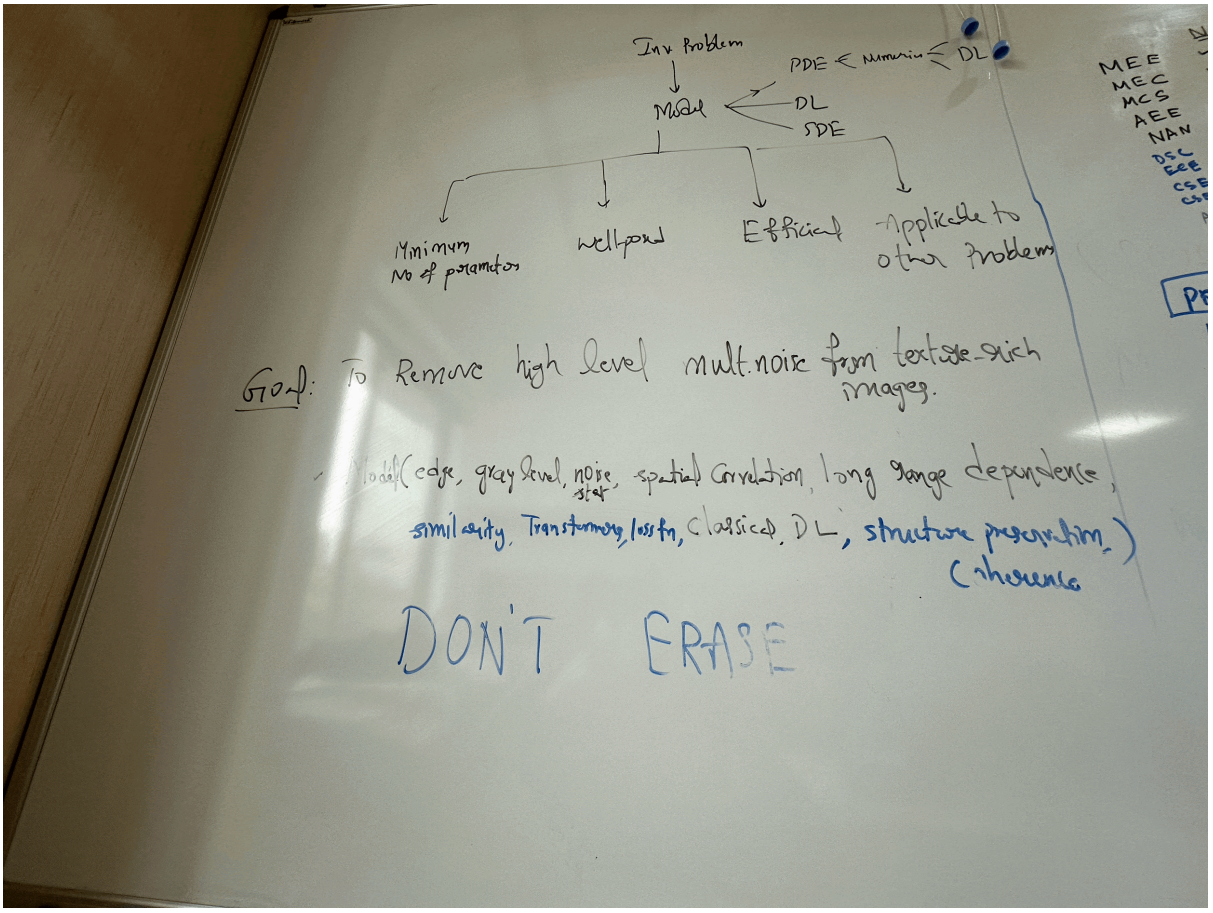
What the vector contains: four simple image statistics computed from the (clean) image:

- **Mean gray level:** overall brightness.
- **Global variance:** how much intensity varies.

- **Downsample difference:** how much fine detail vs. smooth area (a proxy for spatial correlation).
- **Laplacian energy:** a measure of the amount of edges / fine structure.

How it's used: the 4 numbers are converted into a short internal signal (via a tiny neural network) and added to the same "time" signal the model already uses. So the model sees "when" in the diffusion process and "what kind of image" at the same time.

These numbers can be normalized! (for better accuracy !?)



Week 5-6: Incorporating Long Term Dependencies:

- It's a multi-head spatial self-attention layer (called **SpatialSelfAttention**) added to the UNet.
- It helps the model look at relationships between all pixels in a feature map, not just nearby ones.
- It turns the bottleneck features into queries, keys, and values (like in Transformer attention).
- It computes attention across all spatial positions (H×W), meaning every location can "see" every other one.
- The result is then combined back (added residually) with the original bottleneck features.
- Adds a way for the UNet to capture global context and long-range dependencies.
- Improves coherence and structure in outputs — especially useful for restoring large patterns or textures.

Functionally, adding and calling this layer gives the network an explicit mechanism to capture long-range dependencies and propagate global context into reconstruction, which often improves restoration of large-scale structure and consistency across distant patches.

Results:

Loss for 100 images training: (Epochs: 10, Diffusion steps: 150)

W/o improvement: 0.77
With Improvement: 0.67

Loss Functions:

PSNR (Peak Signal-to-Noise Ratio):
PSNR measures the pixel-level fidelity between a restored (or generated) image and its reference. It is derived from the Mean Squared Error (MSE) — higher PSNR indicates smaller pixel differences and thus better reconstruction quality. However, PSNR mainly captures numerical similarity and may not always align with perceived visual quality, especially when structural or perceptual distortions are present.

SSIM (Structural Similarity Index Measure):
SSIM evaluates image quality based on perceptual similarity, considering luminance, contrast, and structural consistency between two images. It is designed to mimic the human visual system's sensitivity to structural information rather than raw pixel errors. SSIM values range from 0 to 1, with 1 indicating perfect similarity. It is more perceptually meaningful than PSNR for assessing image restoration and denoising results.

LPIPS (Learned Perceptual Image Patch Similarity):
LPIPS measures perceptual similarity using deep neural network features rather than raw pixels. It computes distances between feature activations of two images extracted from pretrained networks (like AlexNet or VGG). Lower LPIPS values mean the images look more similar to human observers. Unlike PSNR and SSIM, LPIPS better correlates with human judgments of image realism and perceptual quality.

Week 5-6: More Literature Review:

Paper 1:

There are two key ingredients in modern restoration:

1. Likelihood (data term) : how the observed noisy image relates to the true clean image.
2. Prior (knowledge term) : what a “realistic image” looks like, usually learned by a neural network.

Balance?

The 1st paper tries to find this balance using Bayesian sampling with diffusion models, and that’s where Langevin sampling comes in.

Langevin Sampling ~ Doob’s h-transform

Langevin sampling is a method to sample (generate) images that fit both:

- the observed data (likelihood), and
- what looks realistic (prior).

Instead of directly finding one “best” image, it walks through the space of possible images — like taking a noisy but guided walk that gradually moves toward high-probability (good) solutions.

That process — gradient ascent + random noise -> is what Langevin sampling does.

- The “gradient” term pushes the sample toward where the image is more likely to be true (like following the slope of the probability landscape).
- The “random noise” term keeps the process exploring (avoiding local traps).
- Repeating this many times lets you draw realistic conclusions images from the *posterior distribution* (the set of all possible clean images that could have produced your observed image).

Here's what's happening conceptually in the first paper:

1. Start with a pre-trained diffusion model that knows how to generate natural-looking images.
2. Add the measurement model (the blur, noise, or missing pixels).
3. Use Langevin sampling to slowly adjust a random initial image:
    - Each step moves it a bit closer to fitting the observed noisy data (using the likelihood gradient).
    - Simultaneously keeps it looking natural (using the diffusion prior).
4. Automatically tune how strongly to trust the prior vs. data using Bayesian parameter estimation.
5. End result: a clean image that looks real *and* is consistent with what was measured.

This is similar to how diffusion models generate clean images, but with an added Bayesian correction step.

How Can We Borrow the Idea:

- Add a short Langevin refinement stage after your diffusion/ODE process.
    - Take your denoised image, compute how far it deviates from the observed speckled image, and slightly adjust it in that direction while adding tiny random noise.
    - This helps enforce consistency with the input (just like the first paper does).

Why not Langevin Sampling?: Our reverse process is already a stochastic sampler

Shortcomings of 1st Paper:

i) Only handles additive Gaussian noise

ii) High Computational Cost

iii) Needs a pre-trained large diffusion model

iv) Smoothened images

Week 7: Presentation

Week 8: SAR_DDPM

What the Paper Does (High-Level Idea)

The paper introduces SAR-DDPM, the *first* diffusion-model–based method for removing speckle noise from SAR (Synthetic Aperture Radar) images.

It adapts a denoising diffusion probabilistic model (DDPM) — originally designed for image generation — and conditions it on the speckled SAR image to iteratively reconstruct a clean image. It also introduces a neat enhancement using cycle spinning to further improve despeckling.

Approach Explained (Step-by-Step)

1. The SAR Noise Problem

- SAR images contain multiplicative speckle noise:
  $Y = X \cdot N$
  where YY=speckled, XX=clean, NN=Gamma-distributed speckle.
- Speckle hurts downstream tasks, so despeckling is critical.

2. Forward Diffusion Process (Like DDPM)

The clean image xCxC is gradually destroyed by adding Gaussian noise over TT steps

This gives the network supervised pairs of:

- noisy version xtxt
- the true noise $\epsilon\epsilon$
- the speckled image xSxS, which will be used as a condition.

3. Reverse Diffusion

Standard DDPM predicts noise conditioned only on xtxt.
SAR-DDPM predicts noise conditioned on both:

- the intermediate noisy image xtxt
- the *actual SAR speckled image* xSxS

That means:

> The model uses the observed speckled image as a guide to reconstruct a plausible clean version.

Network architecture

- A U-Net with:
    - residual blocks
    - attention blocks
    - positional time embeddings
    - up/downsampling via BigGAN-style layers

So at each step:

ϵθ(xt,xS,t)ϵθ(xt,xS,t)

predicts the noise added, enabling reconstruction of xt−1xt−1.

4. Training Data

- There is no clean SAR ground truth available.
- So the authors create synthetic data:
    - Take optical images
    - Convert them to synthetic SAR by adding Gamma-distributed multiplicative speckle.

This allows supervised training: (synthetic speckled, clean optical) pairs.

5. Cycle Spinning (Ensemble Trick)

Cycle spinning = cyclically shift the image, run the denoiser, unshift, and average.

Why?

- Diffusion models sometimes introduce spatial artifacts.
- Shifting reduces these, giving smoother homogeneous regions and sharper edges.

Paper uses 3×3 grid of shifts:
u, v ∈ {0, 100, 200}.

The improvement in PSNR and SSIM is big.

6. Results

- Synthetic images: SAR-DDPM + cycle spinning achieves highest PSNR (29.42 dB) and SSIM (0.81).
- Real SAR: Evaluated using ENL → still best or competitive.
- Qualitative results: preserves structure better than methods like PPB, BM3D, SAR-CAM, Trans-SAR.

Limitations

1. Training on *synthetic* speckle, not real SAR noise

- Real SAR speckle is not perfectly Gamma distributed, and real images have:
    - spatial correlations
    - sensor-specific artifacts
    - calibration inconsistencies
- Training on optical images + fake speckle creates domain mismatch.

This may limit generalization to real SAR data.

2. Very computationally expensive

- DDPM requires T = 1000 denoising steps.
- For each cycle-spun version, they run all 1000 steps.
- With 9 shifts (3×3), runtime becomes huge.
  This is impractical for real-time SAR applications.

3. No physics model incorporated

- SAR imaging has coherent scattering physics, but the diffusion model:
    - does not use prior SAR statistics
    - ignores structural constraints like Gamma distribution in the reverse process
    - doesn't use multilook models
      Thus performance depends solely on the network's learning ability.

4. Slight distortions in homogeneous areas (before cycle spinning)

The base SAR-DDPM (without cycle spinning):

- adds small artifacts in flat regions
- ENL is actually worse than basic filters unless cycle spinning is added.

Cycle spinning solves it, but adds

5. Not tested varied SAR datasets

Only tested on:

- DSIFN (optical→synthetic)
- Sentinel-1 (real SAR)
  This limits claims of robustness.