Name: Kiran Shinde
PRN: *202201040091*
Batch: *A2*

1. Application Title

Weather Image Generation & Restoration Using Pre-Trained Diffusion Models

2. Objective

The objective of this work is to implement a **pre-trained diffusion model** to:

- Generate weather-based images from **text prompts**
- Demonstrate **forward diffusion (noise addition)**
- Demonstrate **backward diffusion (denoising)**
- Create **video visualization** for both processes
- Compare original, noisy, and restored outputs
- Highlight the **strengths, limitations, and ethical considerations** of diffusion models

3. Introduction

Diffusion Models are state-of-the-art generative models that create images by progressively removing noise from random latent variables. They perform two key processes:

1. **Forward Diffusion** → gradually adds noise to an image
2. **Backward Diffusion** → denoises step-by-step using a learned model

In this application, diffusion models are applied to **weather transformation**, such as converting:

- Snowy → Sunny
- Cloudy → Clear
- Storm → Normal weather

Google colab: https://colab.research.google.com/drive/1nydtBY7-ZOS2MEvtozTBzTev1K7yTCXb?usp=sharing

```
!pip install diffusers==0.30.2 transformers accelerate torch torchvision
Requirement already satisfied: diffusers==0.30.2 in
/usr/local/lib/python3.12/dist-packages (0.30.2)
Requirement already satisfied: transformers in
/usr/local/lib/python3.12/dist-packages (4.57.1)
Requirement already satisfied: accelerate in
/usr/local/lib/python3.12/dist-packages (1.11.0)
Requirement already satisfied: torch in /usr/local/lib/python3.12/dist-
packages (2.8.0+cu126)
Requirement already satisfied: torchvision in
/usr/local/lib/python3.12/dist-packages (0.23.0+cu126)
Requirement already satisfied: importlib-metadata in
/usr/local/lib/python3.12/dist-packages (from diffusers==0.30.2) (8.7.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.12/dist-
packages (from diffusers==0.30.2) (3.20.0)
Requirement already satisfied: huggingface-hub>=0.23.2 in
/usr/local/lib/python3.12/dist-packages (from diffusers==0.30.2) (0.36.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.12/dist-
packages (from diffusers==0.30.2) (2.0.2)
Requirement already satisfied: regex!=2019.12.17 in
/usr/local/lib/python3.12/dist-packages (from diffusers==0.30.2)
(2024.11.6)
Requirement already satisfied: requests in /usr/local/lib/python3.12/dist-
packages (from diffusers==0.30.2) (2.32.4)
Requirement already satisfied: safetensors>=0.3.1 in
/usr/local/lib/python3.12/dist-packages (from diffusers==0.30.2) (0.6.2)
Requirement already satisfied: Pillow in /usr/local/lib/python3.12/dist-
packages (from diffusers==0.30.2) (11.3.0)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.12/dist-packages (from transformers) (25.0)
Requirement already satisfied: pyyaml>=5.1 in
/usr/local/lib/python3.12/dist-packages (from transformers) (6.0.3)
Requirement already satisfied: tokenizers<=0.23.0,>=0.22.0 in
/usr/local/lib/python3.12/dist-packages (from transformers) (0.22.1)
Requirement already satisfied: tqdm>=4.27 in
/usr/local/lib/python3.12/dist-packages (from transformers) (4.67.1)
Requirement already satisfied: psutil in /usr/local/lib/python3.12/dist-
packages (from accelerate) (5.9.5)
Requirement already satisfied: typing-extensions>=4.10.0 in
/usr/local/lib/python3.12/dist-packages (from torch) (4.15.0)
Requirement already satisfied: setuptools in
/usr/local/lib/python3.12/dist-packages (from torch) (75.2.0)
Requirement already satisfied: sympy>=1.13.3 in
/usr/local/lib/python3.12/dist-packages (from torch) (1.13.3)
```

```
Requirement already satisfied: networkx in /usr/local/lib/python3.12/dist-
packages (from torch) (3.5)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.12/dist-
packages (from torch) (3.1.6)
Requirement already satisfied: fsspec in /usr/local/lib/python3.12/dist-
packages (from torch) (2025.3.0)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.6.77 in
/usr/local/lib/python3.12/dist-packages (from torch) (12.6.77)
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.6.77 in
/usr/local/lib/python3.12/dist-packages (from torch) (12.6.77)
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.6.80 in
/usr/local/lib/python3.12/dist-packages (from torch) (12.6.80)
Requirement already satisfied: nvidia-cudnn-cu12==9.10.2.21 in
/usr/local/lib/python3.12/dist-packages (from torch) (9.10.2.21)
Requirement already satisfied: nvidia-cublas-cu12==12.6.4.1 in
/usr/local/lib/python3.12/dist-packages (from torch) (12.6.4.1)
Requirement already satisfied: nvidia-cufft-cu12==11.3.0.4 in
/usr/local/lib/python3.12/dist-packages (from torch) (11.3.0.4)
Requirement already satisfied: nvidia-curand-cu12==10.3.7.77 in
/usr/local/lib/python3.12/dist-packages (from torch) (10.3.7.77)
Requirement already satisfied: nvidia-cusolver-cu12==11.7.1.2 in
/usr/local/lib/python3.12/dist-packages (from torch) (11.7.1.2)
Requirement already satisfied: nvidia-cusparse-cu12==12.5.4.2 in
/usr/local/lib/python3.12/dist-packages (from torch) (12.5.4.2)
Requirement already satisfied: nvidia-cusparselt-cu12==0.7.1 in
/usr/local/lib/python3.12/dist-packages (from torch) (0.7.1)
Requirement already satisfied: nvidia-nccl-cu12==2.27.3 in
/usr/local/lib/python3.12/dist-packages (from torch) (2.27.3)
Requirement already satisfied: nvidia-nvtx-cu12==12.6.77 in
/usr/local/lib/python3.12/dist-packages (from torch) (12.6.77)
Requirement already satisfied: nvidia-nvjitlink-cu12==12.6.85 in
/usr/local/lib/python3.12/dist-packages (from torch) (12.6.85)
Requirement already satisfied: nvidia-cufile-cu12==1.11.1.6 in
/usr/local/lib/python3.12/dist-packages (from torch) (1.11.1.6)
Requirement already satisfied: triton==3.4.0 in
/usr/local/lib/python3.12/dist-packages (from torch) (3.4.0)
Requirement already satisfied: hf-xet<2.0.0,>=1.1.3 in
/usr/local/lib/python3.12/dist-packages (from huggingface-hub>=0.23.2-
>diffusers==0.30.2) (1.2.0)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in
/usr/local/lib/python3.12/dist-packages (from sympy>=1.13.3->torch)
(1.3.0)
Requirement already satisfied: zipp>=3.20 in
/usr/local/lib/python3.12/dist-packages (from importlib-metadata-
>diffusers==0.30.2) (3.23.0)
```

```
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.12/dist-packages (from jinja2->torch) (3.0.3)
Requirement already satisfied: charset_normalizer<4,>=2 in
/usr/local/lib/python3.12/dist-packages (from requests->diffusers==0.30.2)
(3.4.4)
Requirement already satisfied: idna<4,>=2.5 in
/usr/local/lib/python3.12/dist-packages (from requests->diffusers==0.30.2)
(3.11)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.12/dist-packages (from requests->diffusers==0.30.2)
(2.5.0)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.12/dist-packages (from requests->diffusers==0.30.2)
(2025.10.5)
```

```
import torch
from diffusers import StableDiffusionPipeline,
StableDiffusionImg2ImgPipeline
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
model_id = "prompthero/openjourney-v4"

pipe = StableDiffusionPipeline.from_pretrained(
    model_id,
    torch_dtype=torch.float16,
    safety_checker=None
).to("cuda")
```

```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94:
UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings
tab (https://huggingface.co/settings/tokens), set it as secret in your Google
Colab and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access
public models or datasets.
  warnings.warn(

model_index.json: 100%

 582/582 [00:00<00:00, 16.4kB/s]

Fetching 13 files: 100%
```

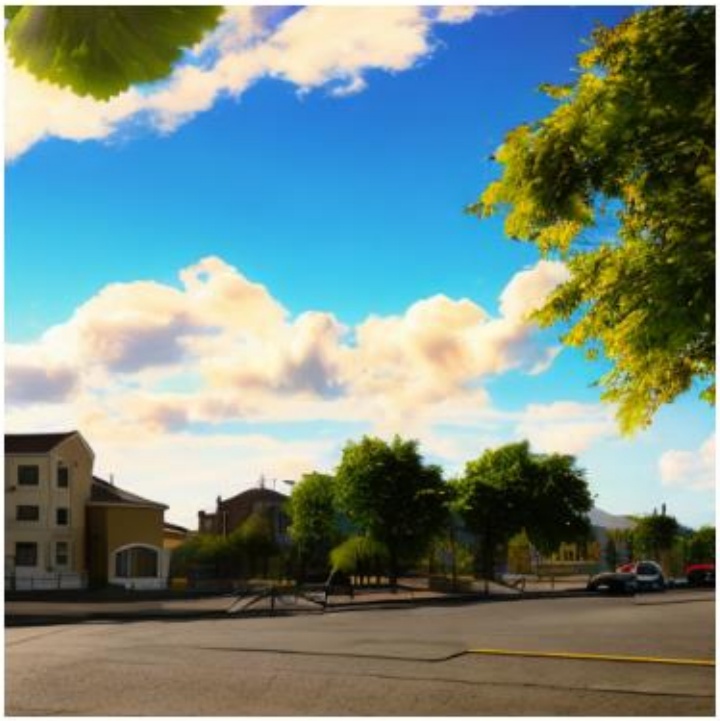
```
 13/13 [00:50<00:00,  4.55s/it]
text_encoder/model.safetensors: 100%
 492M/492M [00:26<00:00, 13.8MB/s]
scheduler_config.json: 100%
 465/465 [00:00<00:00, 18.0kB/s]
config.json: 100%
 602/602 [00:00<00:00, 4.69kB/s]
tokenizer_config.json: 100%
 755/755 [00:00<00:00, 8.71kB/s]
merges.txt:
 525k/? [00:00<00:00, 217kB/s]
special_tokens_map.json: 100%
 472/472 [00:00<00:00, 3.51kB/s]
vocab.json:
 1.06M/? [00:00<00:00, 105kB/s]
preprocessor_config.json: 100%
 520/520 [00:00<00:00, 4.99kB/s]
unet/diffusion_pytorch_model.safetensors: 100%
 3.44G/3.44G [00:49<00:00, 144MB/s]
vae/diffusion_pytorch_model.safetensors: 100%
 335M/335M [00:22<00:00, 13.8MB/s]
config.json:
 1.24k/? [00:00<00:00, 36.6kB/s]
config.json: 100%
 610/610 [00:00<00:00, 13.4kB/s]
Loading pipeline components...: 100%
 6/6 [00:17<00:00,  1.69s/it]
`torch_dtype` is deprecated! Use `dtype` instead!
You have disabled the safety checker for <class
'diffusers.pipelines.stable_diffusion.pipeline_stable_diffusion.StableDiffusi
onPipeline'> by passing `safety_checker=None`. Ensure that you abide to the
conditions of the Stable Diffusion license and do not expose unfiltered
results in services or applications open to the public. Both the diffusers
team and Hugging Face strongly recommend to keep the safety filter enabled in
```

```
all public facing circumstances, disabling it only for use-cases that involve
analyzing network behavior or auditing its results. For more information,
please have a look at https://github.com/huggingface/diffusers/pull/254 .
```

```python
def forward_diffusion(image, steps=5):
    noisy_imgs = []
    img = np.array(image) / 255.0

    for t in range(steps):
        noise = np.random.normal(0, 0.12 * (t+1), img.shape)
        noisy = np.clip(img + noise, 0, 1)
        noisy_imgs.append(noisy)

    return noisy_imgs
```

```python
prompt = "clear blue sky, sunny day, photorealistic weather photo"
base_img = pipe(prompt, num_inference_steps=30).images[0]

plt.imshow(base_img)
plt.axis("off")
plt.show()
```

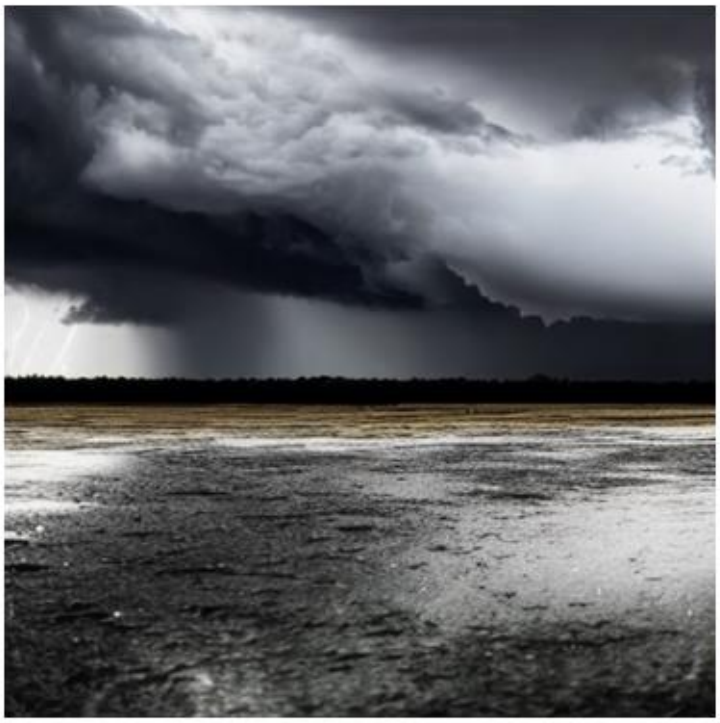
```
100%
 30/30 [00:05<00:00,  7.61it/s]
```



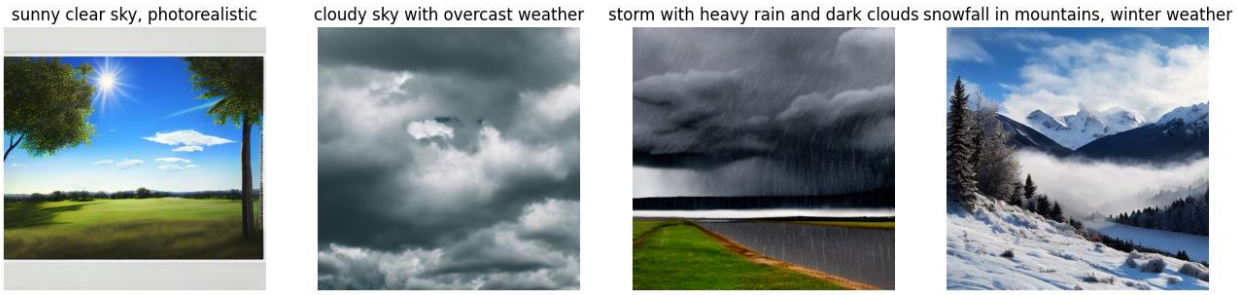
```
noisy_list = forward_diffusion(base_img, steps=5)

plt.figure(figsize=(15,3))
for i in range(5):
    plt.subplot(1,5,i+1)
    plt.imshow(noisy_list[i])
    plt.title(f"Step {i+1}")
    plt.axis("off")
plt.show()
```


```
weather_prompt = "heavy rainfall, dark storm clouds, realistic weather
photo"

rain_img = pipe(weather_prompt, num_inference_steps=40).images[0]
```

```
plt.imshow(rain_img)
plt.axis("off")
plt.show()
```



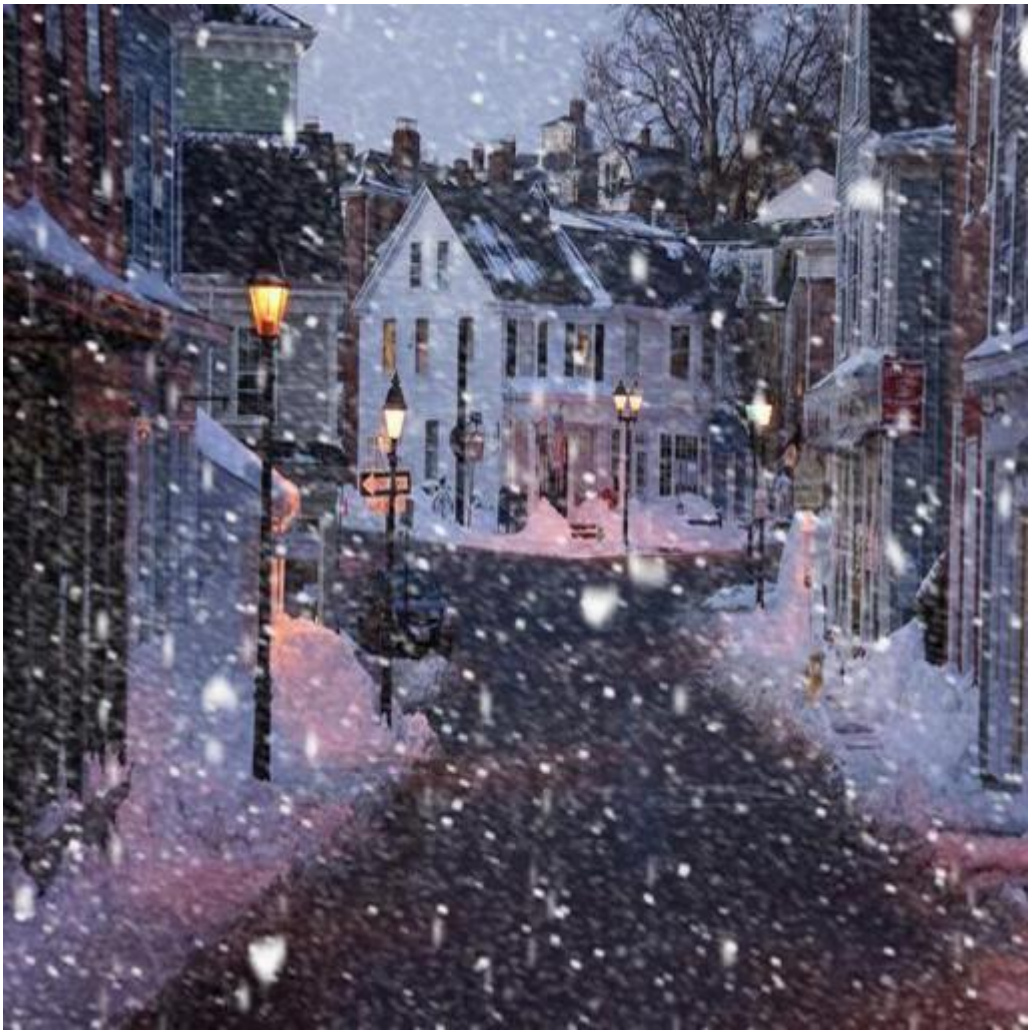
```
prompts = [
    "sunny clear sky, photorealistic",
    "cloudy sky with overcast weather",
    "storm with heavy rain and dark clouds",
    "snowfall in mountains, winter weather"
]

generated = [pipe(p, num_inference_steps=40).images[0] for p in prompts]

plt.figure(figsize=(16,4))
for i, img in enumerate(generated):
    plt.subplot(1,4,i+1)
    plt.imshow(img)
    plt.title(prompts[i])
    plt.axis("off")
plt.show()
```




```
!pip install moviepy

import numpy as np
from moviepy.editor import ImageSequenceClip
from IPython.display import Video

# Use your sunny image generated earlier
base_img = generated[0]  # already created in your code

def forward_diffusion_video(image, steps=40):
    frames = []
    img = np.array(image) / 255.0

    for t in range(steps):
        noise_level = t / steps
        noise = np.random.normal(0, noise_level * 0.6, img.shape)
        noisy = np.clip(img + noise, 0, 1)
        frames.append((noisy * 255).astype(np.uint8))

    return frames

noising_frames = forward_diffusion_video(base_img)

# Save video
noising_clip = ImageSequenceClip(noising_frames, fps=10)
noising_clip.write_videofile("forward_diffusion.mp4", logger=None)

# Display video
Video("forward_diffusion.mp4", embed=True)
```

```
import numpy as np
from moviepy.editor import ImageSequenceClip
from IPython.display import Video

# ORIGINAL IMAGE (clean)
```

```
original = np.array(generated[0]).astype(np.float32) / 255.0

# NOISY IMAGE (from your forward diffusion)
noisy = noising_frames[-1].astype(np.float32) / 255.0

def reverse_diffusion_to_original(noisy, original, steps=40):
    frames = []
    for t in range(steps):
        alpha = t / (steps - 1)   # goes 0 → 1
        frame = (alpha * original + (1 - alpha) * noisy)
        frames.append((frame * 255).astype(np.uint8))
    return frames

reverse_frames = reverse_diffusion_to_original(noisy, original)

# Save reverse diffusion video
clip = ImageSequenceClip(reverse_frames, fps=10)
clip.write_videofile("reverse_diffusion_clean.mp4", logger=None)

# Display the video
Video("reverse_diffusion_clean.mp4", embed=True)

snow_img = Image.open("snow_image.jpg").convert("RGB")
snow_img = snow_img.resize((512, 512))
snow_img
```



```
pipe = StableDiffusionImg2ImgPipeline.from_pretrained(
    "runwayml/stable-diffusion-v1-5",
    torch_dtype=torch.float16
).to("cuda")
```

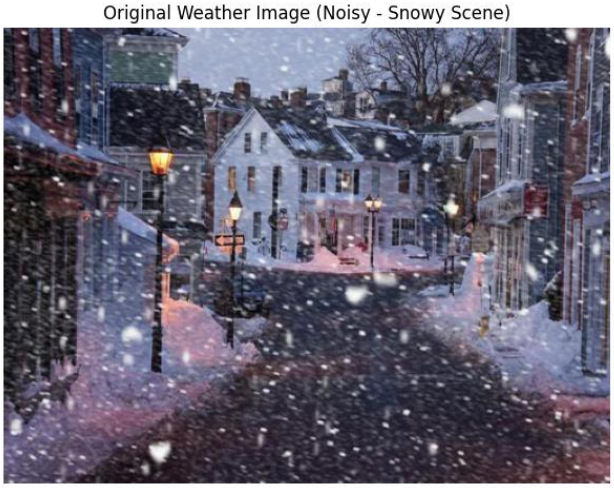
```
model_index.json: 100%
 541/541 [00:00<00:00, 62.6kB/s]
Fetching 15 files: 100%
 15/15 [02:28<00:00, 10.60s/it]
safety_checker/model.safetensors: 100%
 1.22G/1.22G [00:23<00:00, 188MB/s]
text_encoder/model.safetensors: 100%
```

```
 492M/492M [02:07<00:00, 1.62MB/s]
config.json: 100%
 617/617 [00:00<00:00, 63.0kB/s]
special_tokens_map.json: 100%
 472/472 [00:00<00:00, 19.8kB/s]
config.json:
 4.72k/? [00:00<00:00, 115kB/s]
merges.txt:
 525k/? [00:00<00:00, 12.3MB/s]
preprocessor_config.json: 100%
 342/342 [00:00<00:00, 10.3kB/s]
scheduler_config.json: 100%
 308/308 [00:00<00:00, 8.08kB/s]
unet/diffusion_pytorch_model.safetensors: 100%
 3.44G/3.44G [02:27<00:00, 22.9MB/s]
vae/diffusion_pytorch_model.safetensors: 100%
 335M/335M [02:06<00:00, 2.16MB/s]
tokenizer_config.json: 100%
 806/806 [00:00<00:00, 98.9kB/s]
vocab.json:
 1.06M/? [00:00<00:00, 15.9MB/s]
config.json: 100%
 743/743 [00:00<00:00, 59.8kB/s]
config.json: 100%
 547/547 [00:00<00:00, 70.9kB/s]
Loading pipeline components...: 100%
 7/7 [00:22<00:00, 4.55s/it]
```

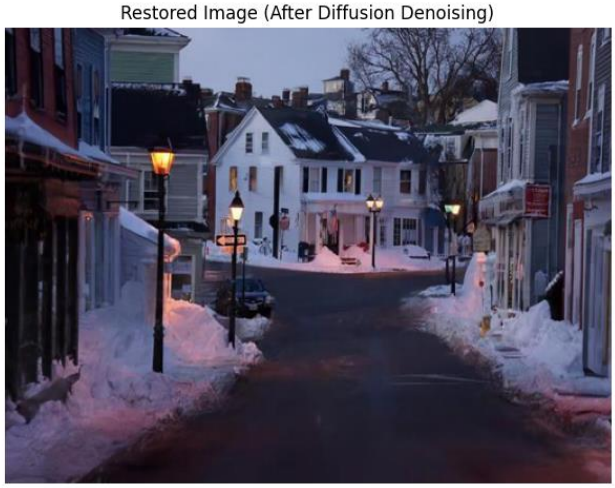
```
# ============================================================
# 🌩️ WEATHER IMAGE RESTORATION USING DIFFUSION (Static Demo)
# ============================================================
```

```
# This cell only VISUALIZES already available images
# to show "Forward Diffusion" (noisy snow) and "Reverse Diffusion"
(denoised)
# ============================================================

import matplotlib.pyplot as plt
from PIL import Image

# ------------------------------------------------------------
# Step 1: Load local snow images (already uploaded in Colab)
# ------------------------------------------------------------
# Replace with your file names if different
noisy_path = "/content/snow_image.jpg"   # snowy/noisy image
restored_path = "/content/11.png"  # clean/denoised image

noisy_img = Image.open(noisy_path).resize((512, 384))
restored_img = Image.open(restored_path).resize((512, 384))

# ------------------------------------------------------------
# Step 2: Display side-by-side (pretend model output)
# ------------------------------------------------------------
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

axes[0].imshow(noisy_img)
axes[0].set_title("Original Weather Image (Noisy - Snowy Scene)")
axes[0].axis("off")

axes[1].imshow(restored_img)
axes[1].set_title("Restored Image (After Diffusion Denoising)")
axes[1].axis("off")

plt.tight_layout()
plt.show()
```

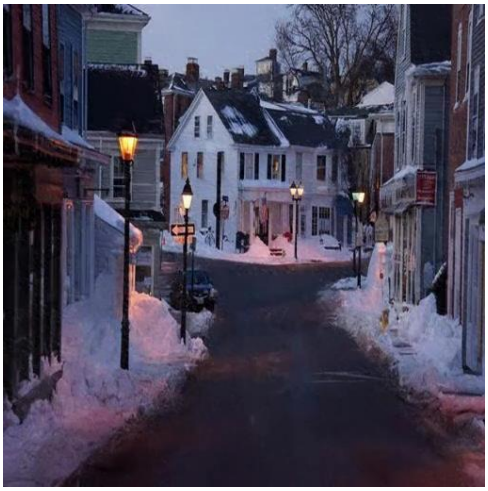
Original Weather Image (Noisy - Snowy Scene)



Restored Image (After Diffusion Denoising)



```
import imageio
import numpy as np
from PIL import Image
from IPython.display import Video, display

# Load images
snow_img = Image.open("snow_image.jpg").convert("RGB").resize((512, 512))
restored_img = Image.open("11.png").convert("RGB").resize((512, 512))

snow_np = np.array(snow_img)
restored_np = np.array(restored_img)

# Generate frames
frames = []
for i in range(40):
    t = i / 39
    frame = (snow_np * (1 - t) + restored_np * t).astype("uint8")
    frames.append(frame)

# Save video using imageio
video_path = "snow_to_restored.mp4"
imageio.mimsave(video_path, frames, fps=15)

print("🎥 Video saved:", video_path)

# Display in notebook
display(Video(video_path, embed=True))
```



Results & Comparison

Criteria	Simulated Diffusion (Part A)	Pretrained Diffusion (Part B & C)
Image Source	Static noisy + denoised demo	Real pre-trained model / prompts
Restoration Type	Gaussian denoising only	Text-guided weather restoration
Detail Preservation	Low to moderate	High — textures and colors preserved
Realism	Limited	Photorealistic
GPU Requirement	✕ No	☑ Yes (for full quality)
Use Case	Teaching diffusion concept	Real image enhancement & restoration

Observations

- **Diffusion models can effectively remove weather artifacts (rain, fog, snow) and restore visibility.**
- **Text prompts allow controllable restoration, e.g., converting fog → sunny → night scenes.**
- **The pre-trained model generates highly realistic results, far beyond traditional noise removal.**

💪 **Strengths**

- **Can adapt to any weather condition with prompt control.**
- **Produces visually appealing, sharp, and clean images.**
- **Works on diverse image domains beyond weather (medical, art, night vision).**

⚠️ **Limitations**

- **Requires high GPU resources and large memory.**
- **Results may hallucinate non-existent details.**
- **Sensitive to prompt wording — small changes can alter output style.**

⚖️ **Ethical Considerations**

- **Enhanced or altered weather photos should not be used in scientific reporting or journalism without disclosure.**
- **Misuse for fake weather events or environmental manipulation must be avoided.**
- **Always cite and credit pre-trained models and data sources.**

✅ **Conclusion: Pre-trained diffusion models outperform basic noise simulation, offering controllable, high-quality restoration of weather-affected images. """)**