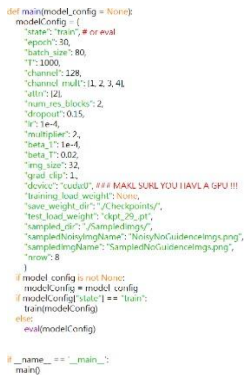
1. Theoretical Justification
   The method I used involves generating a noisy image with DIP, which is then fed into DDPM to shorten the training time and produce an image. This approach leverages DIP's ability to function without pre-training, creating a noisy image that includes some structural elements of the target photo in a short time. By utilizing DDPM's capability to restore noise, this image with structural information can reconstruct the target photo in just a few epochs. The reason for this efficiency is that DIP has already provided part of the photo's structure to DDPM. The advantage of this method is that, compared to using DDPM alone, it significantly reduces the model's training time. Additionally, for DIP, this approach eliminates the need to search for the "best stopping point," as DDPM can effectively remove the noise without needing to identify a stopping point.
2. Experimental Verification
   Below is an example of using DDPM alone (utilizing the CIFAR-10 dataset):

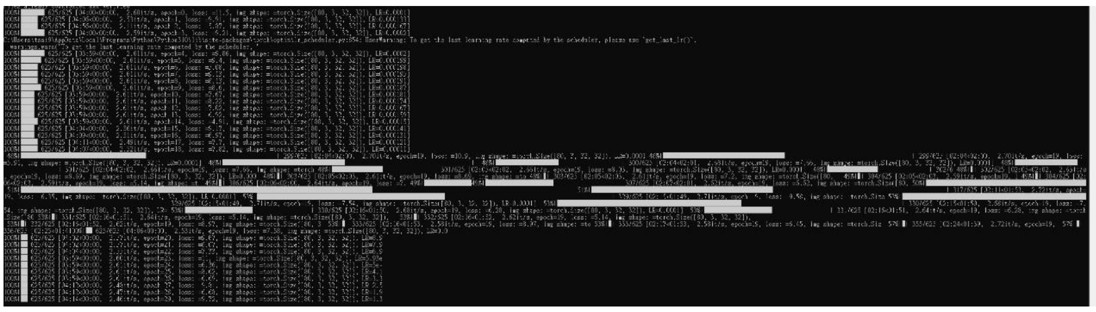
   Input:
   Output:

   Hyperparameter:



```
def main(model_config = None):
    modelConfig = {
        "state": "train", # or eval
        "epoch": 30,
        "batch_size": 80,
        "T": 1000,
        "channel": 128,
        "channel_mult": [1, 2, 3, 4],
        "attn": [2],
        "num_res_blocks": 2,
        "dropout": 0.15,
        "lr": 1e-4,
        "multiplier": 2.,
        "beta_1": 1e-4,
        "beta_T": 0.02,
        "img_size": 32,
        "grad_clip": 1.,
        "device": "cuda:0", ### MAKE SURE YOU HAVE A GPU !!!
        "training_load_weight": None,
        "save_weight_dir": "./Checkpoints/",
        "test_load_weight": "ckpt_29_.pt",
        "sampled_dir": "./SampledImgs/",
        "sampledNoisyImgName": "NoisyNoGuidenceImgs.png",
        "sampledImgName": "SampledNoGuidenceImgs.png",
        "nrow": 8
        }
    if model_config is not None:
        modelConfig = model_config
    if modelConfig["state"] == "train":
        train(modelConfig)
    else:
        eval(modelConfig)


if __name__ == '__main__':
    main()
```

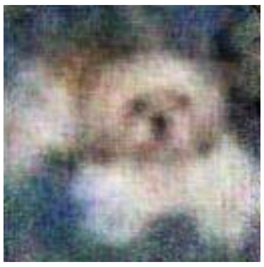
   Generation detail: (~120mins)



```
100%|██████| 625/625 [04:00<00:00,  2.60it/s, epoch=0, loss: =11.5, img shape: =torch.Size([80, 3, 32, 32]), LR=0.0001]
100%|██████| 625/625 [04:06<00:00,  2.53it/s, epoch=1, loss: =9.9, img shape: =torch.Size([80, 3, 32, 32]), LR=0.000133]
100%|██████| 625/625 [04:56<00:00,  2.11it/s, epoch=2, loss: =5.87, img shape: =torch.Size([80, 3, 32, 32]), LR=0.000167]
100%|██████| 625/625 [04:00<00:00,  2.59it/s, epoch=3, loss: =9.21, img shape: =torch.Size([80, 3, 32, 32]), LR=0.0002]
C:\Users\tani9\AppData\Local\Programs\Python\Python310\lib\site-packages\torch\optim\lr_scheduler.py:854: UserWarning: To get the last learning rate computed by the sche
  warnings.warn("To get the last learning rate computed by the scheduler, "
100%|██████| 625/625 [03:59<00:00,  2.61it/s, epoch=4, loss: =9.86, img shape: =torch.Size([80, 3, 32, 32]), LR=0.0002]
100%|██████| 625/625 [03:59<00:00,  2.61it/s, epoch=5, loss: =5.4, img shape: =torch.Size([80, 3, 32, 32]), LR=0.000199]
100%|██████| 625/625 [03:59<00:00,  2.61it/s, epoch=6, loss: =7.08, img shape: =torch.Size([80, 3, 32, 32]), LR=0.000198]
100%|██████| 625/625 [03:59<00:00,  2.61it/s, epoch=7, loss: =8.13, img shape: =torch.Size([80, 3, 32, 32]), LR=0.000195]
100%|██████| 625/625 [03:59<00:00,  2.61it/s, epoch=8, loss: =8.13, img shape: =torch.Size([80, 3, 32, 32]), LR=0.000191]
100%|██████| 625/625 [03:59<00:00,  2.61it/s, epoch=9, loss: =8.6, img shape: =torch.Size([80, 3, 32, 32]), LR=0.000187]
100%|██████| 625/625 [03:59<00:00,  2.61it/s, epoch=10, loss: =7.67, img shape: =torch.Size([80, 3, 32, 32]), LR=0.000181]
100%|██████| 625/625 [03:59<00:00,  2.61it/s, epoch=11, loss: =8.22, img shape: =torch.Size([80, 3, 32, 32]), LR=0.000174]
100%|██████| 625/625 [03:59<00:00,  2.61it/s, epoch=12, loss: =7.03, img shape: =torch.Size([80, 3, 32, 32]), LR=0.000167]
100%|██████| 625/625 [03:59<00:00,  2.61it/s, epoch=13, loss: =6.99, img shape: =torch.Size([80, 3, 32, 32]), LR=0.000159]
100%|██████| 625/625 [03:59<00:00,  2.61it/s, epoch=14, loss: =4.91, img shape: =torch.Size([80, 3, 32, 32]), LR=0.00015]
100%|██████| 625/625 [04:34<00:00,  2.36it/s, epoch=15, loss: =5.17, img shape: =torch.Size([80, 3, 32, 32]), LR=0.000141]
100%|██████| 625/625 [04:09<00:00,  2.51it/s, epoch=16, loss: =6.97, img shape: =torch.Size([80, 3, 32, 32]), LR=0.000131]
100%|██████| 625/625 [04:11<00:00,  2.49it/s, epoch=17, loss: =7.7, img shape: =torch.Size([80, 3, 32, 32]), LR=0.000121]
100%|██████| 625/625 [04:07<00:00,  2.52it/s, epoch=18, loss: =7.82, img shape: =torch.Size([80, 3, 32, 32]), LR=0.000111]
 48%|████    | 298/625 [02:00<02:00,  2.70it/s, epoch=19, loss: =10.9, img shape: =torch.Size([80, 3,
[illegible]
336/625 [02:25<01:10]100%|██| 625/625 [04:06<00:00,  2.53it/s, epoch=19, loss: =7.38, img shape: =torch.Size([80, 3, 32, 32]), LR=0.0
100%|█| 625/625 [04:02<00:00,  2.57it/s, epoch=20, loss: =6.67, img shape: =torch.Size([80, 3, 32, 32]), LR=8.9
100%|█| 625/625 [04:02<00:00,  2.57it/s, epoch=21, loss: =8.07, img shape: =torch.Size([80, 3, 32, 32]), LR=7.9
100%|█| 625/625 [04:04<00:00,  2.55it/s, epoch=22, loss: =7.32, img shape: =torch.Size([80, 3, 32, 32]), LR=6.9
100%|█| 625/625 [03:59<00:00,  2.60it/s, epoch=23, loss: =[illegible], img shape: =torch.Size([80, 3, 32, 32]), LR=5.92e
100%|█| 625/625 [03:59<00:00,  2.61it/s, epoch=24, loss: =6.36, img shape: =torch.Size([80, 3, 32, 32]), LR=5e
100%|█| 625/625 [03:59<00:00,  2.61it/s, epoch=25, loss: =8.62, img shape: =torch.Size([80, 3, 32, 32]), LR=4.1
100%|█| 625/625 [03:59<00:00,  2.61it/s, epoch=26, loss: =6.69, img shape: =torch.Size([80, 3, 32, 32]), LR=3.3
100%|█| 625/625 [04:12<00:00,  2.48it/s, epoch=27, loss: =6.3, img shape: =torch.Size([80, 3, 32, 32]), LR=2.5
100%|█| 625/625 [04:12<00:00,  2.47it/s, epoch=28, loss: =6.68, img shape: =torch.Size([80, 3, 32, 32]), LR=1.9
100%|█| 625/625 [04:14<00:00,  2.46it/s, epoch=29, loss: =5.72, img shape: =torch.Size([80, 3, 32, 32]), LR=1.3
```

Below is an example of using D**IP** alone (utilizing the CIFAR-10 dataset):

Input: output: (iteration = 3000)

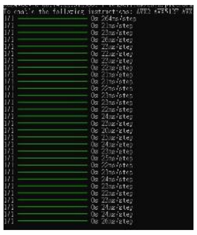


Hyperparameter:

Noise_intensity = 100

Iteration = 3000

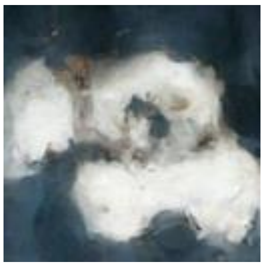
Generation detail: (~931ms)



```
To enable the following instructions: AVX2 AVX512F AVX
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 264ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
```

Below is an example of combining DIP and DDPM (using the CIFAR-10 dataset):

Input (dip): Output(ddpm):

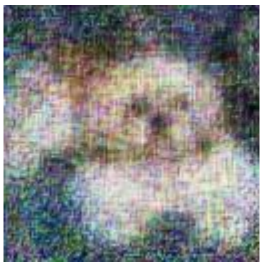


Hyperparamete(dip):

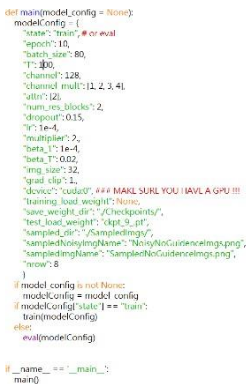
Noise_intensity = 100

Iteration = 5000

(Use iteration =3000 as an input for ddpm):



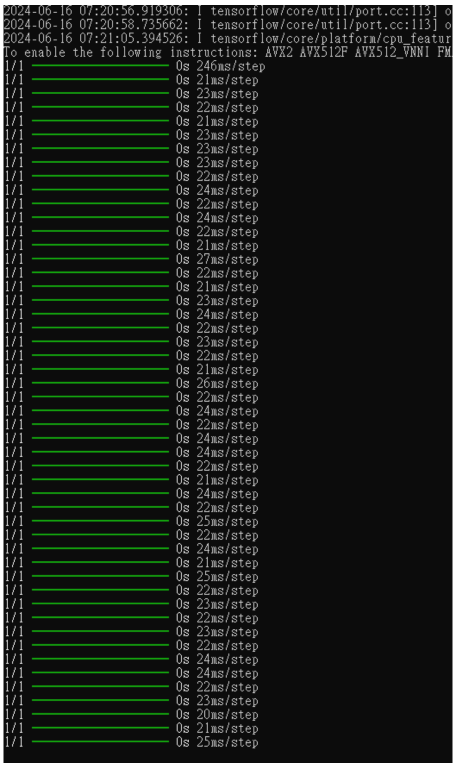
Hyperparamete(ddpm):



```python
def main(model_config = None):
    modelConfig = {
        "state": "train", # or eval
        "epoch": 10,
        "batch_size": 80,
        "T": 1|00,
        "channel": 128,
        "channel_mult": [1, 2, 3, 4],
        "attn": [2],
        "num_res_blocks": 2,
        "dropout": 0.15,
        "lr": 1e-4,
        "multiplier": 2.,
        "beta_1": 1e-4,
        "beta_T": 0.02,
        "img_size": 32,
        "grad_clip": 1.,
        "device": "cuda:0", ### MAKE SURE YOU HAVE A GPU !!!
        "training_load_weight": None,
        "save_weight_dir": "./Checkpoints/",
        "test_load_weight": "ckpt_9_.pt",
        "sampled_dir": "./SampledImgs/",
        "sampledNoisyImgName": "NoisyNoGuidenceImgs.png",
        "sampledImgName": "SampledNoGuidenceImgs.png",
        "nrow": 8
        }
    if model_config is not None:
        modelConfig = model_config
    if modelConfig["state"] == "train":
        train(modelConfig)
    else:
        eval(modelConfig)


if __name__ == '__main__':
    main()
```

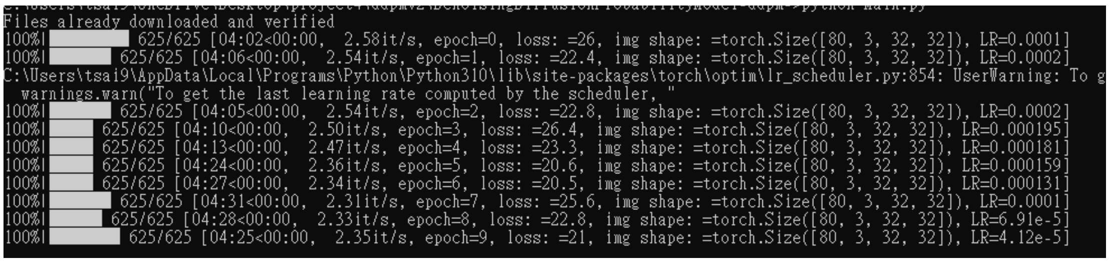
Generation detail: (~44mins) dip:

(~1340ms)



```
2024-06-16 07:20:56.919306: I tensorflow/core/util/port.cc:113] o
2024-06-16 07:20:58.735662: I tensorflow/core/util/port.cc:113] o
2024-06-16 07:21:05.394526: I tensorflow/core/platform/cpu_featur
To enable the following instructions: AVX2 AVX512F AVX512_VNNI FM
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 246ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
```

ddpm: (~43mins)



```
Files already downloaded and verified
100%|█████████| 625/625 [04:02<00:00,  2.58it/s, epoch=0, loss: =26, img shape: =torch.Size([80, 3, 32, 32]), LR=0.0001]
100%|███████| 625/625 [04:06<00:00,  2.54it/s, epoch=1, loss: =22.4, img shape: =torch.Size([80, 3, 32, 32]), LR=0.0002]
C:\Users\tsai9\AppData\Local\Programs\Python\Python310\lib\site-packages\torch\optim\lr_scheduler.py:854: UserWarning: To g
  warnings.warn("To get the last learning rate computed by the scheduler, "
100%|███████| 625/625 [04:05<00:00,  2.54it/s, epoch=2, loss: =22.8, img shape: =torch.Size([80, 3, 32, 32]), LR=0.0002]
100%|█████| 625/625 [04:10<00:00,  2.50it/s, epoch=3, loss: =26.4, img shape: =torch.Size([80, 3, 32, 32]), LR=0.000195]
100%|█████| 625/625 [04:13<00:00,  2.47it/s, epoch=4, loss: =23.3, img shape: =torch.Size([80, 3, 32, 32]), LR=0.000181]
100%|█████| 625/625 [04:24<00:00,  2.36it/s, epoch=5, loss: =20.6, img shape: =torch.Size([80, 3, 32, 32]), LR=0.000159]
100%|█████| 625/625 [04:27<00:00,  2.34it/s, epoch=6, loss: =20.5, img shape: =torch.Size([80, 3, 32, 32]), LR=0.000131]
100%|███████| 625/625 [04:31<00:00,  2.31it/s, epoch=7, loss: =25.6, img shape: =torch.Size([80, 3, 32, 32]), LR=0.0001]
100%|██████| 625/625 [04:28<00:00,  2.33it/s, epoch=8, loss: =22.8, img shape: =torch.Size([80, 3, 32, 32]), LR=6.91e-5]
100%|████████| 625/625 [04:25<00:00,  2.35it/s, epoch=9, loss: =21, img shape: =torch.Size([80, 3, 32, 32]), LR=4.12e-5]
```

From the above, we can see that after combining DDPM and DIP, the training time required for DDPM has significantly decreased (from 120 minutes to 42 minutes). Additionally, we no longer need to choose the "best stopping point" for DIP; we can simply take about half or fewer iterations. However, despite the advantages of merging DDPM and DIP, there is still a noticeable gap in image restoration quality compared to using either method alone. I believe this is because DDPM does not handle the noise in the input provided by DIP effectively, especially since DDPM was only trained for one-third of the epochs. Therefore, when given a noisy image, its restoration might not perform well.

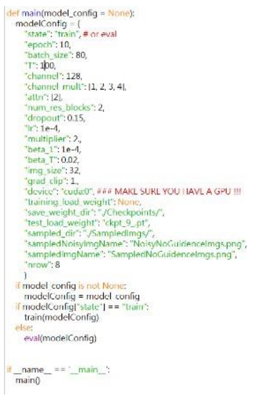
Another reason could be related to the choice of intensity. Although I have set the intensities for both DIP and DDPM to be the same, the intensity of the input for DIP, which has already started the restoration process, might differ significantly from the set value of 100. Consequently, I should experiment with various intensities for DDPM to achieve better results.

Thus, this approach requires a trade-off between image quality and time. If we want to save time, we might have to sacrifice some image quality. Otherwise, we would need to spend time fine-tuning the intensity for DDPM or use DDPM alone.

3. Ablation Studies and Analysis

The results will vary based on different hyperparameters. Below are the fixed hyperparameters that I adjusted for DDPM.

ddpm:



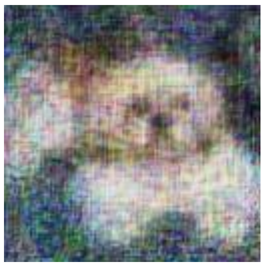
```
def main(model_config = None):
    modelConfig = {
        "state": "train", # or eval
        "epoch": 10,
        "batch_size": 80,
        "T": 100,
        "channel": 128,
        "channel_mult": [1, 2, 3, 4],
        "attn": [2],
        "num_res_blocks": 2,
        "dropout": 0.15,
        "lr": 1e-4,
        "multiplier": 2.,
        "beta_1": 1e-4,
        "beta_T": 0.02,
        "img_size": 32,
        "grad_clip": 1.,
        "device": "cuda:0", ### MAKE SURE YOU HAVE A GPU !!!
        "training_load_weight": None,
        "save_weight_dir": "./Checkpoints/",
        "test_load_weight": "ckpt_9_.pt",
        "sampled_dir": "./SampledImgs/",
        "sampledNoisyImgName": "NoisyNoGuidenceImgs.png",
        "sampledImgName": "SampledNoGuidenceImgs.png",
        "nrow": 8
        }
    if model_config is not None:
        modelConfig = model_config
    if modelConfig["state"] == "train":
        train(modelConfig)
    else:
        eval(modelConfig)


if __name__ == '__main__':
    main()
```

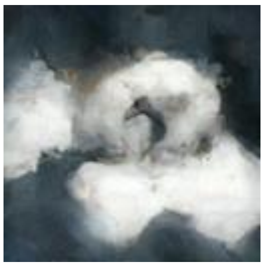
dip:

Noise_intensity = 100

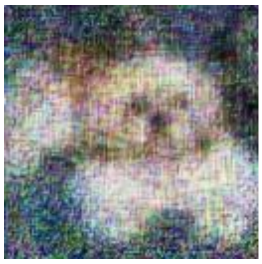
Iteration = 5000

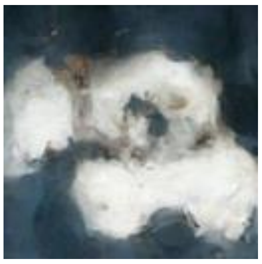
Input: ddpmInput: (Use iteration = 1500) output:





Input: ddpmInput: (Use iteration = 3000) output:



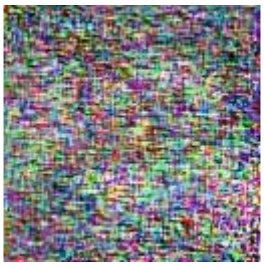


dip:

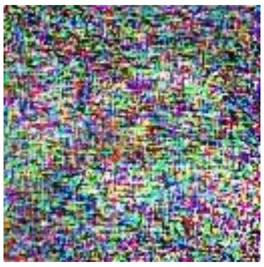
Noise_intensity = 500

Iteration = 5000

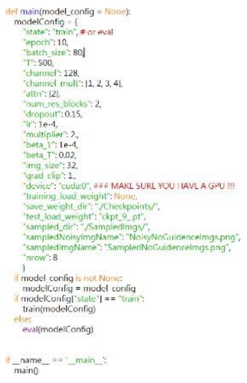
Input: ddpmInput: (Use iteration = 1500) output:



Input: ddpmInput: (Use iteration = 3000) output:



ddpm:



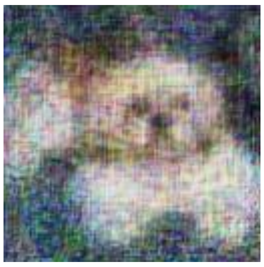
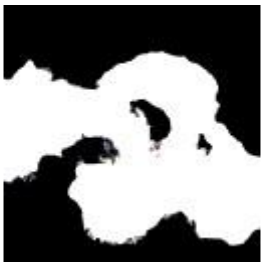
```
def main(model_config = None):
    modelConfig = {
        "state": "train", # or eval
        "epoch": 10,
        "batch_size": 80
        "T": 500,
        "channel": 128,
        "channel_mult": [1, 2, 3, 4],
        "attn": [2],
        "num_res_blocks": 2,
        "dropout": 0.15,
        "lr": 1e-4,
        "multiplier": 2.,
        "beta_1": 1e-4,
        "beta_T": 0.02,
        "img_size": 32,
        "grad_clip": 1.,
        "device": "cuda:0", ### MAKE SURE YOU HAVE A GPU !!!
        "training_load_weight": None,
        "save_weight_dir": "./Checkpoints/",
        "test_load_weight": "ckpt_9_.pt",
        "sampled_dir": "./SampledImgs/",
        "sampledNoisyImgName": "NoisyNoGuidenceImgs.png",
        "sampledImgName": "SampledNoGuidenceImgs.png",
        "nrow": 8
        }
    if model_config is not None:
        modelConfig = model_config
    if modelConfig["state"] == "train":
        train(modelConfig)
    else:
        eval(modelConfig)


if __name__ == '__main__':
    main()
```

dip:

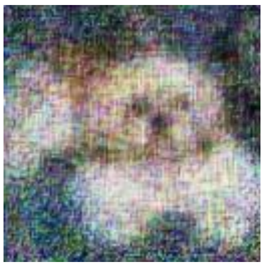
Noise_intensity = 100

Iteration = 5000

Input: ddpmInput: (Use iteration = 1500) output:

Input: ddpmInput: (Use iteration = 3000) output:

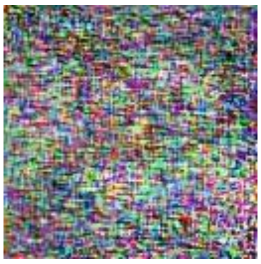
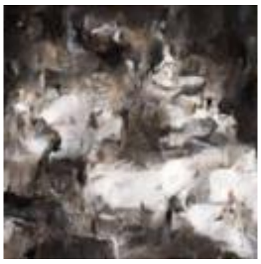


dip:

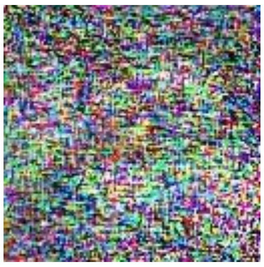
Noise_intensity = 500

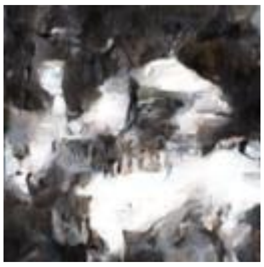
Iteration = 5000

Input: ddpmInput: (Use iteration =1500) output:

Input: DdpmInput: (Use iteration = 3000) output:





From the above, we can summarize the following points:

1. The intensity in DDPM affects the restoration results; if set significantly higher than DIP, it may remove too much noise, thereby damaging the original photo.
2. When the intensity in DIP is too high, regardless of the DDPM settings, the input noise may not be properly managed, preventing the photo's structure from being represented. As a result, DDPM cannot recognize the structure and thus cannot infer the correct outcome.
3. When the number of iterations in DIP is larger, it is easier for DDPM to capture more of the photo's structure, leading to better detail in images produced with higher iterations compared to those with fewer iterations.

Therefore, when choosing to combine DIP with DDPM, we must pay attention to:

1. The more initial structure present in the input photo, the better the results; conversely, less structure yields poorer results.
2. The intensities of DDPM and DIP should not differ too much; otherwise, DDPM may excessively remove noise from the photo, damaging it.

Github link: https://github.com/lane14093051/GAI_project4_2/tree/main