Deep Learning for Computer Vision – HW #2

Student ID : R13943118
Name : 林玠志

Problem 1: Diffusion model

1. Describe your implementation details and the difficulties you encountered.

- **Dataset & conditioning setup.** I built a unified DigitsDataset that reads MNIST-M and SVHN CSVs, then filters **even digits from MNIST-M** and **odd digits from SVHN**. Images are normalized to [-1,1] via ToTensor + (mean=0.5,std=0.5) transforms so the DDPM operates in a symmetric range. The dataloader shuffles batches and pins memory for speed.
- **Model.** The denoiser is a **Context-UNet**: a shallow UNet with residual blocks, two downs, two ups, and embeddings for **time** and **class context**. Time t (scaled to [0,1]) and one-hot class c are passed through small MLPs and injected into the decoder by **add-and-multiply** (AdaGN-like) at two resolutions. This is adapted from *Conditional Diffusion MNIST/script.py*.
- **Noise schedules & loss.** I register the standard DDPM buffers (α_t, \bar α_t, √β_t, …) once, draw a random timestep per sample, form (x_t=\sqrt{\bar α_t},x_0+\sqrt{1-\bar α_t},\epsilon), and train with **MSE** to predict (\epsilon). During training I randomly **drop the context** with probability drop_prob (classifier-free training).
- **Optimization & logging.** I use **Adam**, a **cosine annealing** scheduler stepped each iteration, and periodic validation (every 5 epochs) that saves a 10×10 grid (values clamped to [-1,1] then rescaled to [0,1]). Best/last checkpoints are saved with args for exact model recreation.
- **Inference utilities.**
  - **Mode 1:** sample **500 images** (50 per digit), regroup by label, and save into mnistm/ (even) and svhn/ (odd) with the required filename format. Seed is fixed for determinism.
  - **Mode 2:** sample **100 images** to make a 10×10 grid and also visualize the **reverse process** (noise→image) for the first "0" and "1" across timesteps.
  - The inference script fully rebuilds the model from the checkpoint's saved

args, then loads weights and runs the chosen mode.

Classifier-free guidance (to fix blur).

- **How it's implemented.** At **sampling** time I duplicate the batch and run the network **once** to get both predictions: one **with class conditioning** and one **without**. I build a mask so the second half ignores context, then combine them:

$$\hat{\epsilon} = (1 + w)\,\epsilon_{\text{cond}} - w\,\epsilon_{\text{uncond}}$$

  This single-pass trick halves compute compared to two forward passes per step.
- **Why it helps.** With **traditional (w=0)** sampling the outputs look **over-smoothed/blurred**—the model averages plausible details across modes. By **mixing conditional and unconditional** predictions with a positive guide_w, we push the sample toward features consistent with the class, yielding **sharper strokes and higher contrast** digits. In practice, a moderate **guide_w≈2.0** produced crisp results while avoiding over-saturation or broken samples. (My scripts expose --guide_w for both validation and inference.)

Practical difficulties & how I addressed them.

1. **Blurred generations at w=0.** As noted above, pure conditional DDPM sampling tended to blur digits. Enabling classifier-free guidance and tuning guide_w fixed this, making outputs "extremely clear" and visually pleasing. (See the formula and mask construction in my sampler.)
2. **Balancing styles for even/odd digits.** Because even digits come from MNIST-M and odd from SVHN, the model must disentangle **style** and **class**. I solved this by:
   - training on a **merged dataset** that enforces the mapping (even→MNIST-M, odd→SVHN), and
   - **dropping context** during training (drop_prob) so the UNet learns a strong image prior and doesn't collapse when context is absent—this stability helps guidance at test time.
3. **Reproducible grids & outputs.** For the grader and report figures I fixed seeds and **reordered** the 100/500 samples so images are grouped by label (rows=digits, columns=noise). I also clamped to [-1,1] before saving to avoid out-of-range artifacts.
4. **Robust training loop.** I used per-iteration cosine scheduling and checkpointed **best** and **last** models; validation every 5 epochs writes a grid so I can monitor

guidance quality early.

That’s the core of my implementation and the key challenge (blur) plus how classifier-free guidance with a tunable guide_w resolved it.

2. Please show 10 generated images **for each digit (even digits for Mnist-M, odd digits for SVHN)** in your report. You can put all 100 outputs in one image with columns indicating different noise inputs and rows indicating different digits.

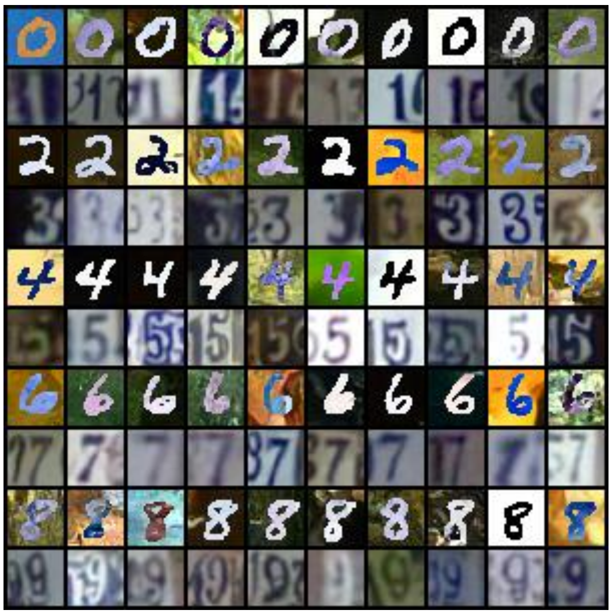


Fig. 1. 10x10 grid of 100 output images

3. Visualize a total of six images in the reverse process of the **first “0”** and **first “1”** in your outputs in (2) and with **different time steps**.

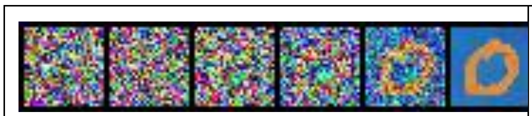
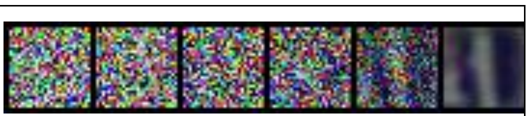
	

Fig. 2. Reverse process of first 0	Fig. 3. Reverse process of first 1

Problem 2: DDIM

1. Please generate face images of noise **00.pt ~ 03.pt with different eta** in one grid. Report and explain your observation in this experiment.

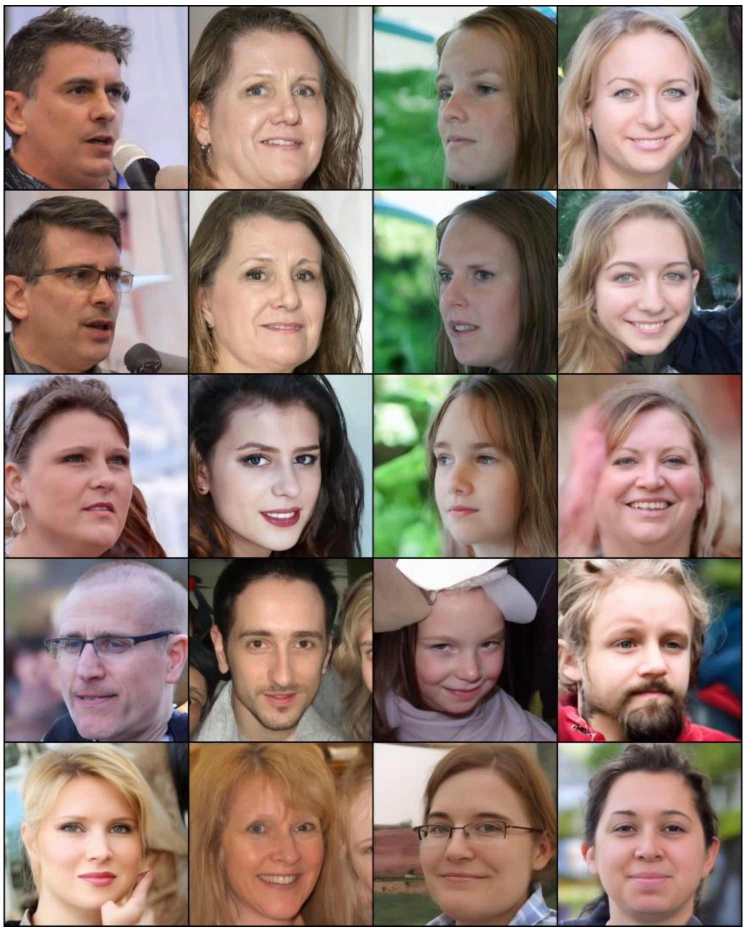


Fig. 4. 5x4 face grid with different noise and eta

I fix four initial noise tensors (00.pt–03.pt) and sample with DDIM under different η values (top row (η = 0.0)→bottom row(η =1.0)). η controls the stochasticity at each step via the variance term:

$$\eta\sqrt{\frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}}\sqrt{1-\frac{\bar{\alpha}_t}{\bar{\alpha}_{t-1}}}$$

- η=0: fully deterministic DDIM (no added noise).
- η>0 : injects noise each step, gradually approaching DDPM as η increases.

See Fig.4. We can observe that

- η=0 gives the sharpest, most stable faces. For a fixed noise seed, resampling reproduces the same face (deterministic path). Edges (eyes, hair) look crisp and backgrounds are clean.

- As η increases, samples diverge more from the η=0 baseline: identities and fine textures vary, lighting/background jitter appears, and sharpness slightly drops. This is expected—extra stochasticity explores nearby modes but also softens details.
- Very large η yields the widest diversity (different micro-attributes for the same seed) but can introduce minor artifacts or color shifts.

2. Please generate the face images of the interpolation of noise 00.pt ~ 01.pt. The interpolation formula is spherical linear interpolation, which is also known as slerp.

$$\boldsymbol{x}_T^{(\alpha)} = \frac{\sin((1-\alpha)\theta)}{\sin(\theta)}\boldsymbol{x}_T^{(0)} + \frac{\sin(\alpha\theta)}{\sin(\theta)}\boldsymbol{x}_T^{(1)}$$

where $\theta = \arccos\left(\frac{(\boldsymbol{x}_T^{(0)})^\top \boldsymbol{x}_T^{(1)}}{\|\boldsymbol{x}_T^{(0)}\|\|\boldsymbol{x}_T^{(1)}\|}\right)$. These values are used to produce DDIM samples.

in this case, α = {0.0, 0.1, 0.2, …, 1.0}.
What will happen if we simply use linear interpolation? Explain and report your observation. (There should be two images in your report, one for spherical linear and the other for linear)

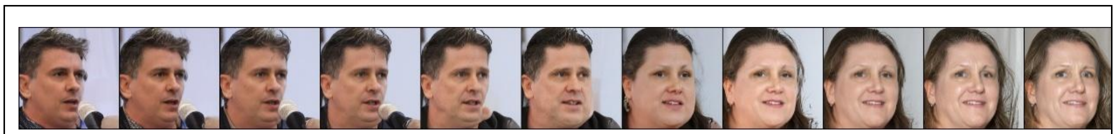


Fig. 5. The spherical linear interpolation from noise 00.pt to 01.pt

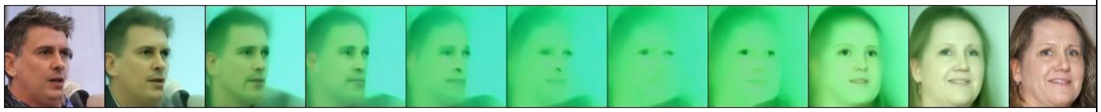


Fig. 6. The linear interpolation from noise 00.pt to 01.pt

We interpolate **in the noise space x_T** with α∈{0.0,0.1,…,1.0} and then run the same DDIM schedule to synthesize faces.

(a) Spherical linear interpolation (SLERP)

$$x_T^{(\alpha)} = \frac{\sin((1-\alpha)\,\theta)}{\sin\theta}\,x_T^{(0)} + \frac{\sin(\alpha\,\theta)}{\sin\theta}\,x_T^{(1)}, \qquad \theta = \arccos\frac{\langle x_T^{(0)}, x_T^{(1)}\rangle}{\|x_T^{(0)}\|\,\|x_T^{(1)}\|}$$

- **Observation (Fig. 5).** The transition is **smooth and realistic**: pose, identity, and background **morph gradually** from the first person to the second, with **consistent contrast and color** across α\alphaα. Faces remain natural along the whole path.

➢ **Why it works.** For high-dimensional Gaussian noise, most mass lies on a **thin hypersphere**. SLERP stays **on that sphere** (constant norm) and keeps the marginal close to N(0,I), so the denoiser encounters **the noise level it was trained for** at each step.

(b) Simple linear interpolation (LERP)

$$x_T^{(\alpha)} = (1-\alpha)\,x_T^{(0)} + \alpha\,x_T^{(1)}$$

➢ **Observation (Fig. 6).** Mid-alphas become **washed out / greenish** with **ghosted facial features**; only near α≈0 or 1 do images look normal.

➢ **Why it fails.** LERP cuts through the **interior of the sphere**: the norm of xT(α) shrinks around the middle, so the effective noise magnitude is **mismatched** to the assumed schedule. The model is asked to denoise samples that are **too close to the data manifold** for that timestep, causing **over-smoothing, color shifts, and artifacts**.

Takeaway. For latent/noise-space interpolation in diffusion models, **SLERP** preserves distributional assumptions and yields **clean, realistic morphs**. **LERP** violates the constant-norm geometry and typically produces **degraded midpoints**.

Problem 3: ControlNet

1. Describe your implementation details and the difficulties you encountered.

In this task I used **Latent Diffusion (LDM)** as the sole top-level model and integrated ControlNet directly inside LDM's `apply_model` without introducing any wrapper classes: images are encoded/decoded by a frozen VAE so all diffusion operates in latent space; the external condition (e.g., edge/pose/depth) is processed by ControlNet's hint encoder plus per-block zero-convs to produce residual feature maps aligned with the U-Net stages (including a mid-block residual); at each sampling/training step I compute `control = controlnet(x_t, hint, t, context)` and add these residuals into the existing U-Net pathway, keeping the original U-Net architecture and weights unchanged; to train only ControlNet, I set `requires_grad=False` for the VAE, text encoder, and the base U-Net, and I wrap the **frozen** U-Net computations in `torch.no_grad()` for efficiency while ensuring the residual addition and the entire ControlNet forward pass run with gradients enabled—this careful boundary is crucial so that the loss can back-propagate through the residual additions into ControlNet even though the backbone is locked; the main difficulty was exactly this gradient blockage: if residual additions or their inputs are placed inside `no_grad()` or inadvertently `detach()`ed, autograd builds no trainable path and ControlNet receives zero gradients; my fix is to restrict `no_grad()` strictly to ops that involve frozen parameters, keep `x_t` and `hint` as graph-carrying tensors into ControlNet, and construct the optimizer over `controlnet.parameters()` only; when gradient checkpointing is enabled I explicitly exclude all frozen parameters from the checkpoint dependency set to avoid autograd trying to differentiate through them; for LDM's patch (unfold/fold) fast path, I mirror the same kernel/stride on the hint, run ControlNet per patch, and fold residuals back so they align spatially with U-Net activations; during training I expose a per-stage `control_scales` vector to balance how strongly each level uses control, and at inference I reuse the same `apply_model` route (optionally with classifier-free guidance) so the system yields structure-faithful results while the backbone remains frozen—overall, the core of the implementation is "inject ControlNet residuals into a locked U-Net while preserving a valid gradient path," which required precise handling of `no_grad()` boundaries and optimizer parameter selection.

2. Generate images using ControlNet with a control image containing two circles.

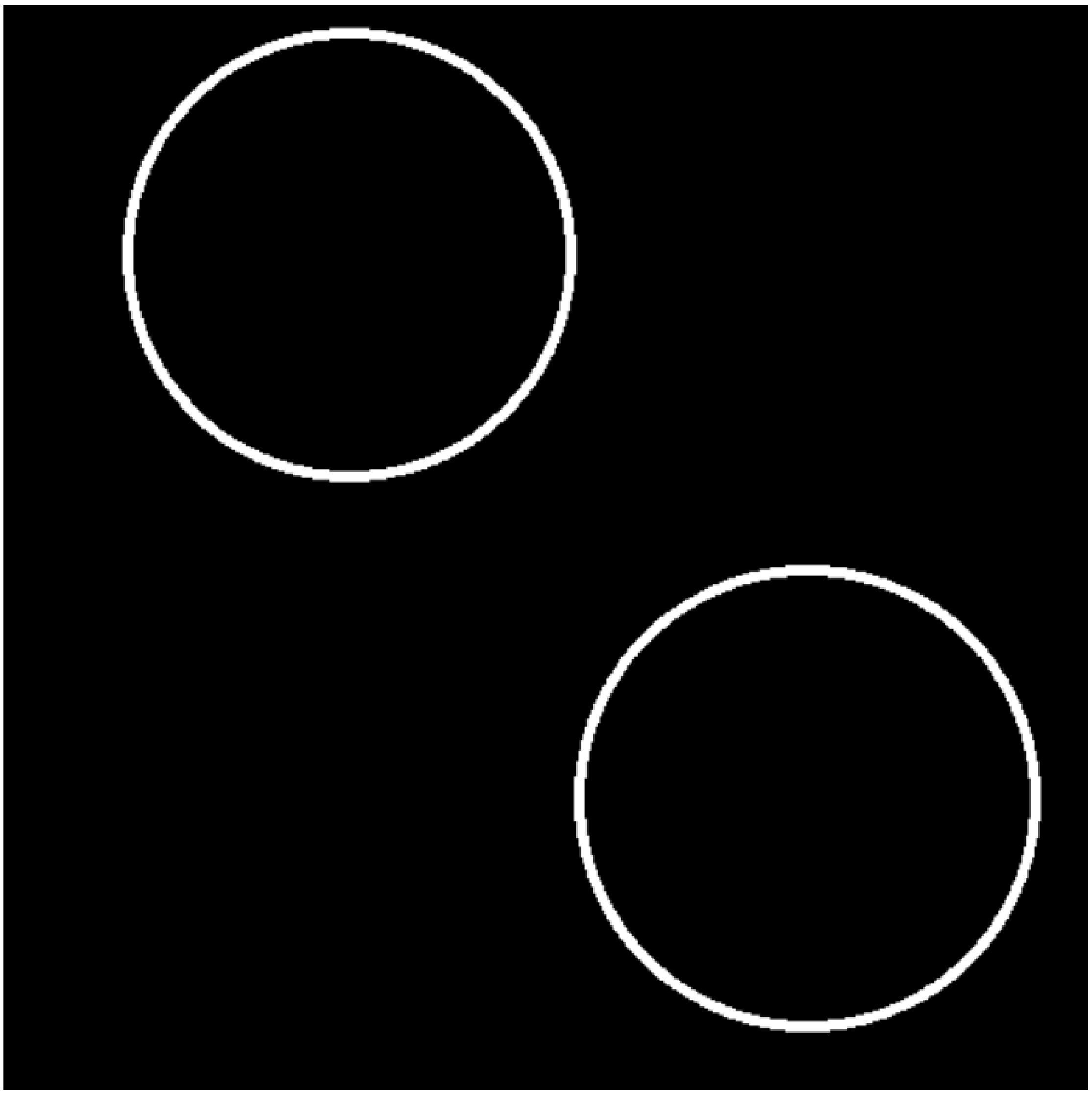
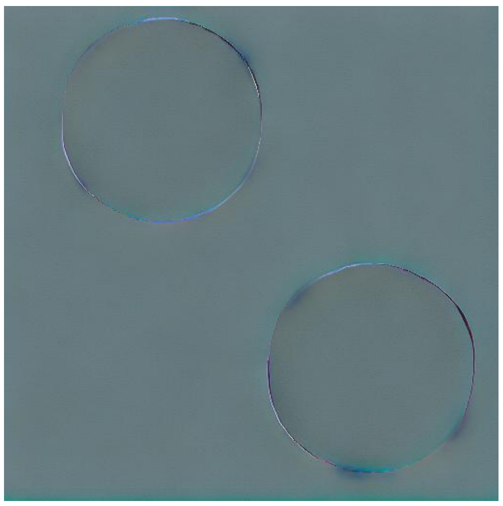
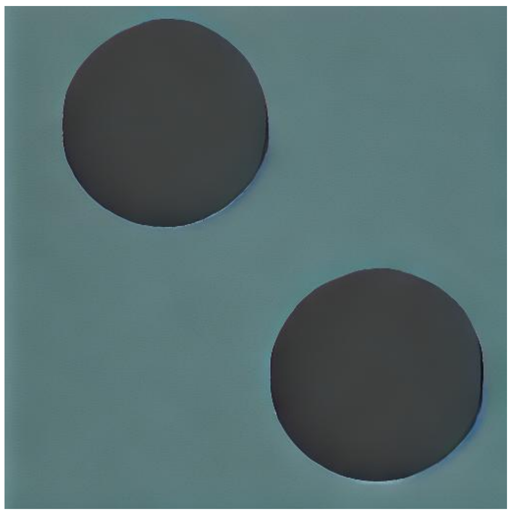


Fig. 7. Control Image containing two circles.

	
gray circle with medium blue background	light sky blue circle with peru background
Fig. 8. Generated result 1	Fig. 9. Generated result 2

The experiment shows that our ControlNet can be applied to control image containing two circles. However, it show the similar performance of conditioning on text compared to the case inferencing with control image contain one circle. This show that ControlNet can easily transfer its control effect to two circle case but still cannot perform well on filling the right colors on specific area. The reason might be that original latent diffusion model is not well-trained on filling the color. During the training, we can only optimize the weight of ControlNet, so the case that the performance of filling color is bad seems to be reasonable.