# EE 8520 - Homework 4

## Conditional Generation for Inverse Problems and Class/Text-Based Conditioning

**Due:** November 26, 2025 **Template:** Any LaTeX template is fine

! Please follow the "rules for every HW" listed in the syllabus

**Programming Exercise:** In this assignment, you will explore and implement several different posterior sampling techniques for solving inverse problems (*e.g.,* inpainting or super-resolution), along with methods for class-conditional and text-to-image (T2I) generation. You are provided with skeleton codes for step 1 and step 2 to help you get started.

Figure 1: Sample images from (a) CelebA-HQ (Liu et al., 2015; Karras et al., 2018) and (b) ImageNet (Deng et al., 2009) datasets. CelebA-HQ is a high-quality celebrity face image dataset widely used for evaluating generative and facial analysis models, while ImageNet serves as a large-scale benchmark for visual recognition and representation learning.

## Step 1 (35 points): Posterior Samplers for Inverse Problems

Your first task is to implement and compare different posterior sampling methods for solving inverse problems using a pre-trained ADM model[1] (Dhariwal & Nichol, 2021) from HW3. However, unlike unconditional generation in HW3, posterior sampling aims to reconstruct an image that is both consistent with a given measurement (*e.g.*, noisy masked or low resolution image) and plausible under the learned data prior.

To help you get started, we have provided the skeleton code for restoration (`hw4_step1_main.py`) along with the ADM model codes and some helper functions (`utils.py`) under the folder `step1_utils`. Make sure to check the lines with the `#TODO` flag inside the `hw4_step1_main.py` file in order to complete the code. Some pointers about the model and code:

† We have provided the CelebA-HQ and ImageNet pretrained ADM checkpoints (click). Please download these `.pt` files and place them under `"./step1_utils/models/"`.

[1] https://github.com/openai/guided-diffusion

† This code builds upon the unconditional generation framework from HW3 and uses the same ADM backbone, but trained on a different dataset. In this step, you will extend that framework from pure image generation to posterior sampling, where the diffusion model is conditioned on measurement consistency. Therefore, most of the information from the ADM setup in HW3 remains applicable.

**Algorithm 1** Posterior Sampling Strategies

**Require:** $T$, $\mathbf{y}$, $\{\tilde{\sigma}_i\}_{i=1}^{T}$, $\eta$

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

2: **for** $t = T, ..., 1$ **do**

3: $\quad \hat{\mathbf{x}}_{0|t} \leftarrow \frac{1}{\sqrt{\bar{\alpha}_t}} \cdot (\mathbf{x}_t - \hat{\boldsymbol{\epsilon}}_\theta(\mathbf{x}_t, t)\sqrt{1-\bar{\alpha}_t})$ ▷ *Tweedie denoised estimate*

4: $\quad \mathbf{x}'_{t-1} \sim p(\mathbf{x}_{t-1}|\mathbf{x}_t, \hat{\mathbf{x}}_{0|t})$ ▷ *DDIM sampling*

5: $\quad \mathbf{x}_{t-1} \leftarrow$ Projection or gradient update via $\mathbf{x}'_{t-1}$ and $\hat{\mathbf{x}}_{0|t}$ to get closer to $\{\mathbf{x}|\mathbf{y} = \mathbf{A}\mathbf{x}\}$

6: **end for**

7: **return** $\mathbf{x}_0$

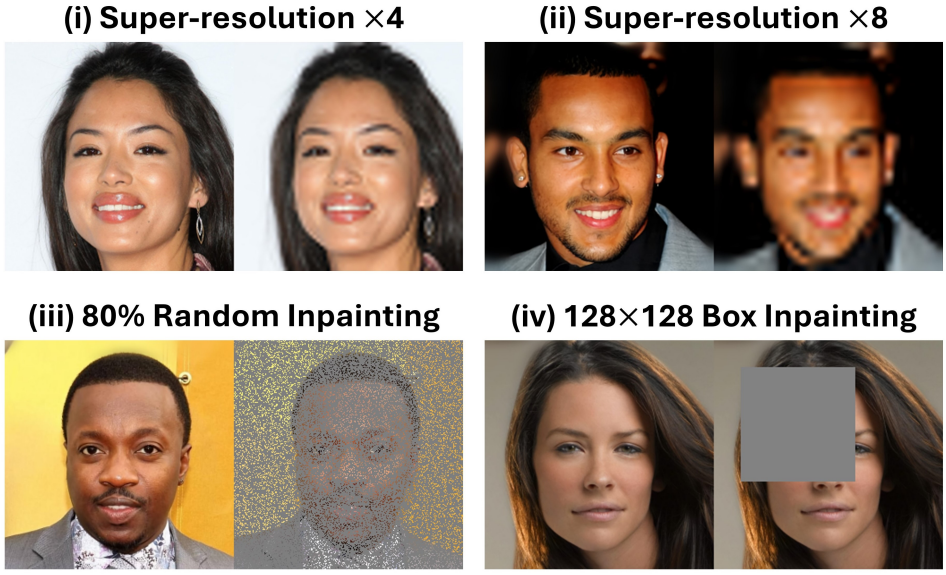



Figure 2: Four degradations that will be covered in this HW. **You can change the box indices as long as you keep it 128×128 and at least 10 pixels away from the edges.**

For this step, your goals are:

(a) **[4 pts]** Implement `predict_x0_hat()` and `sample_ddim()` functions, effectively reproducing Alg. 1 excluding line 5, to ensure you can perform unconditional sampling from both pretrained ADM models. Use $\eta = 1.0$ throughout this assignment, thereby implementing the DDPM variant. Run the sampling process for 1000 steps and present five different samples from each network in a $2 \times 5$ grid.

(b) **[8 pts]** As discussed in the class, earlier attempts focused on direct projections onto the constraint set. For example, Iterative Latent Variable Refinement (ILVR) (Choi et al., 2021) which is initially proposed for super-resolution (SR) tasks considers the following update for line 5 (Alg. 1):

$$\mathbf{x}_{t-1} = \mathbf{x}'_{t-1} + \zeta_{\text{ILVR}} \cdot \mathbf{A}^{\dagger}(\mathbf{y}_{t-1} - \mathbf{A}\mathbf{x}'_{t-1}),$$

where $\zeta_{\text{ILVR}}$ is the weighting parameter, $\mathbf{A}^{\dagger}$ is the left-pseudoinverse of the forward operator, and $\mathbf{y}_{t-1} \sim q(\mathbf{y}_{t-1}|\mathbf{y}_0)$.

- Select 3 images from each dataset's validation set provided to you (click) and place them in their corresponding folders under `"./step1_utils/data/"`. *Note: Their pre-trained networks are different so do not forget to change the* `--dataset` *parser for the correct network prior. Also do not forget to have some fun so choose the celebrities you know! Let's see if you can recognize them after reconstruction!*
- Implement the `q_sample()` function to perform the forward noising process defined as $q(\mathbf{x}_{t-1} \mid \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t-1}; \sqrt{\bar{\alpha}_{t-1}}\,\mathbf{x}_0, (1-\bar{\alpha}_{t-1})\mathbf{I})$.
- Implement the `ilvr()` function to perform ILVR's $\nabla_{\mathbf{x}_t} p(\mathbf{y}|\mathbf{x})$ update. Use the implemented `q_sample()` function to obtain the noisy measurements $\mathbf{y}_{t-1}$.
- Perform image restoration for the following inverse problem tasks: (i) SR×4, (ii) SR×8, (iii) 80% random inpainting, and (iv) 128×128 box inpainting. Assume $\sigma_{\mathbf{y}} = 0$ and **use 1000 sampling steps**. Tune $\zeta_{\text{ILVR}}$ heuristically for best performance, and report reference, measurement, and reconstruction for each image along with their corresponding **PSNR, SSIM, and LPIPS metrics**. *Note: These degradations are already defined for you. You only need to change them from parser operations.*
- **Discuss** the observed performance and report the restoration time per image. Do you obtain similar results for inpainting tasks as for super-resolution tasks? If not, why might that be the case?

(c) **[6 pts]** Manifold Constrained Gradient (MCG) (Chung et al., 2022) improves upon ILVR by first taking a gradient step along the manifold followed by a projection step similar to ILVR (but with $\mathbf{A}^{\top}$ instead of $\mathbf{A}^{\dagger}$):

$$\tilde{\mathbf{x}}_{t-1} = \mathbf{x}'_{t-1} - \zeta_{\text{MCG}} \cdot \nabla_{\mathbf{x}_t} ||\mathbf{y} - \mathbf{A}\hat{\mathbf{x}}_{0|t}||_2$$

$$\mathbf{x}_{t-1} = \mathbf{x}'_{t-1} + \mathbf{A}^{\top}(\mathbf{y}_{t-1} - \mathbf{A}\tilde{\mathbf{x}}_{t-1})$$

- Implement the `mcg()` function and perform image restoration for the following inverse problem tasks: (i) SR×4, (ii) SR×8, (iii) 80% random inpainting, and (iv) 128×128 box inpainting. Assume $\sigma_{\mathbf{y}} = 0$ and **use 1000 sampling steps**. Tune $\zeta_{\text{MCG}}$ heuristically for best performance, and report reference, measurement, and reconstruction for each image along with their corresponding **PSNR, SSIM, and LPIPS metrics**. *Note: Use the same 3 images from part (b) and obtain* $\mathbf{y}_{t-1}$ *similar to ILVR using* `q_sample()` *function.*

- **Discuss** the observed performance and report the restoration time per image. Compare the speed and quality to ILVR. Do you see consistent performance across restoration tasks this time? Although MCG also requires 1000 sampling (denoising) steps, do you notice any slowness compared to ILVR? **Discuss**.

(d) **[6 pts]** Denoising Diffusion Null-Space Model (DDNM) (Wang et al., 2023) replaces the line 5 update in Alg. 1 with an update from a *range-null-space decomposition of the forward operator*. Specifically, at each step, it forms the Tweedie estimate $\hat{\mathbf{x}}_{0|t}$ and projects it onto the affine constraint set as:

$$\tilde{\mathbf{x}}_{0|t} = \hat{\mathbf{x}}_{0|t} + \zeta_{\text{DDNM}} \cdot \mathbf{A}^{\dagger}(\mathbf{y} - \mathbf{A}\hat{\mathbf{x}}_{0|t}).$$

The goal here is to enforce exact data consistency in the range of $\mathbf{A}$, while preserving the null-space component favored by the prior. Once this refined denoised estimate is obtained, DDNM gets the final sample as $\mathbf{x}'_{t-1} \sim p(\mathbf{x}_{t-1}|\mathbf{x}_t, \tilde{\mathbf{x}}_{0|t})$. Given this new formulation, DDNM improves upon past methods in terms of required sampling steps.

- Implement the `ddnm()` function and perform image restoration for the following inverse problem tasks: (i) SR×4, (ii) SR×8, (iii) 80% random inpainting, and (iv) 128×128 box inpainting. Assume $\sigma_{\mathbf{y}} = 0$ and **use 100 sampling steps**. Tune $\zeta_{\text{DDNM}}$ heuristically for best performance, and report reference, measurement, and reconstruction for each image along with their corresponding **PSNR, SSIM, and LPIPS metrics**. *Note: Use the same 3 images from part (b).*
- **Discuss** the observed performance and report the restoration time per image. Compare the speed and quality to ILVR and MCG.

(e) **[5 pts]** Add noise to the measurements with $\sigma_y = 0.05$ (use `--sigma_y`). Repeat steps (b), (c), and (d) *only* for the SR×4 and box inpainting tasks on a single dataset (either CelebA-HQ or ImageNet). Report reference, measurement, and reconstruction for each image. Do you observe any performance degradation for ILVR, MCG, or DDNM? Is the drop in performance more pronounced for one inverse task compared to the other (SR×4 `vs` box inpainting)? **Explain your reasoning for each case.**

(f) **[6 pts]** Diffusion Posterior Sampling (DPS) (Chung et al., 2023) applies MCG update for the most part but it removes the final projection step for stability to noisy conditions. Therefore, its update is given as:

$$\mathbf{x}_{t-1} = \mathbf{x}'_{t-1} - \zeta_{\text{DPS}} \cdot \nabla_{\mathbf{x}_t} ||\mathbf{y} - \mathbf{A}\hat{\mathbf{x}}_{0|t}||_2$$

- Implement the `dps()` function and perform image restoration for the following inverse problem tasks: (i) SR×4, (ii) SR×8, (iii) 80% random inpainting, and (iv) 128×128 box inpainting. Assume $\sigma_{\mathbf{y}} = 0.05$ and **use 1000 sampling steps**. Tune $\zeta_{\text{DPS}}$ heuristically for best performance, and report reference, measurement, and reconstruction for each image along with their corresponding **PSNR, SSIM, and LPIPS metrics**. *Note: Use the same 3 images from part (b).*
- **Discuss** the observed performance and report the restoration time per image. Compare the speed and quality to ILVR, MCG and DDNM. Do you see improved performance across noisy conditions? **Discuss**.

## Step 2 (20 points): Classifier & Classifier-Free Guidance

Your next task is to explore two key conditioning mechanisms used in modern diffusion models: classifier guidance (CG) (Dhariwal & Nichol, 2021) and classifier-free guidance (CFG) (Ho & Salimans, 2021). Both techniques allow diffusion models to steer the generation process toward a desired class or concept, but they achieve this through fundamentally different principles.

To help you get started, we have provided the skeleton code for class-conditional sampling (`hw4_step2_main.py`) along with a lightweight diffusion model and a noise-aware classifier codes under the folder `step2_utils`. We further provided pre-trained networks for each but you can also train them from scratch if desired.

Your task is to complete the implementation of the two missing sampling methods. Make sure to check the lines with the `#TODO` flag inside the `hw4_step2_main.py` file in order to complete the code. Your goals for this step are as follows:

(a) **[7 pts]** Implement `sample_cg()` to perform Classifier Guidance (CG) sampling, where the guidance signal is obtained from the gradient of the noise-aware classifier:

$$\tilde{\boldsymbol{\epsilon}} = \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, \varnothing) - \omega_{\text{CG}} \cdot \sqrt{1-\bar{\alpha}_t}\, \nabla_{\mathbf{x}_t} \log p_\phi(\mathbf{y}|\mathbf{x}_t, t),$$

where $\omega_{\text{CG}}$ is the guidance scale (provided as `--cg_scale` in the parser). Use the reverse update rule of the DDPM parameterization to generate class-conditioned samples for digits 0–9. Present the generated digits in a $10 \times 10$ grid (each row represents 10 samples from a single digit), varying $\omega_{\text{CG}} \in \{0.0, 1.0, 3.0, 5.0, 10.0\}$ to visualize the strength of guidance.

(b) **[7 pts]** Implement `sample_cfg()` to perform Classifier-Free Guidance (CFG) sampling. Use the unconditional and conditional noise predictions from the diffusion model and combine them according to:

$$\tilde{\boldsymbol{\epsilon}} = \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, \varnothing) + \omega_{\text{CFG}} \cdot (\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, c) - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, \varnothing)),$$

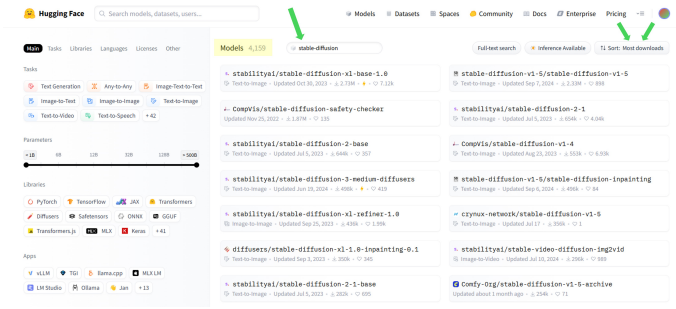
where $\omega_{\text{CFG}}$ is the guidance scale (provided as `--cfg_scale` in the parser). Follow the same reverse update as in the DDPM case, and visualize the generated digits in a $10 \times 10$ grid for $\omega_{\text{CG}} \in \{0.0, 1.0, 3.0, 5.0, 10.0\}$.

(c) **[6 pts]** Compare the outputs of CFG and CG both qualitatively and quantitatively. For each method, compute:

- Classification accuracy of the generated images using the provided noise-aware classifier.
- Intra-class diversity using the `intraclass_diversity_cosine()` metric (already implemented for you).

Reflect on your findings. How does the guidance scale influence sample quality and diversity in each method? Which method do you find more stable or visually consistent across classes? Based on your implementations and observations, explain the fundamental difference between CG and CFG. Why do you think CFG is more popular for text-to-image generation?

## Step 3 (20 points): Text-to-Image Generation & CLIP Similarity




(a)


(b)

Your final task is to implement a Stable Diffusion (SD) model (Rombach et al., 2022) of your choice. In this step, **no skeleton code is provided**, as the implementation is straightforward using the high-level APIs available in Hugging Face's 🤗 `Diffusers`[2] library, which offers well-documented and modular components for text-to-image generation.

To select a model, search for "stable-diffusion" under the *Models* section on Hugging Face 🤗. With over 4,000 models available, it is recommended to sort the results by "Most downloads", as illustrated in figure above (a). Once you find a model you like, review its model card to understand the implementation details.

*Note: In most cases, the provided code examples are highly compact and may offer limited flexibility (for example, lacking options to adjust the CFG scale, number of sampling steps, or $\eta$). You may need to write additional (but short) custom code to incorporate these parameters. In this case, additional documentations can be helpful.* **Make sure** *to set a fixed random seed for reproducibility (check this by running the pipeline with exact same conditions twice and ensure you get the same final output).*

❗ Please name your main python file as `"hw4_step3_main.py"` and (if necessary) put any other helper function inside a folder which you can name `"step3_utils"`.

For this step, your goals are:

(a) **[2 pts]** Run your pipeline using $\omega_{\text{CFG}} = 0.0$, $\eta = 0.0$ and 50 sampling steps without any positive or negative prompt to make sure everything works (this should give you a meaningful image generated unconditionally).

(b) **[6 pts]** Create 15 engaging prompts based on 5 unique and different topics (see some examples in (b) figure). For each topic, write three versions of the same idea:

1. **Simple and short:** a short, clear prompt (about one sentence).
2. **Medium:** a slightly longer prompt that adds some detail or context.
3. **Long and complex:** a detailed, imaginative version that expands on the idea with creative or descriptive elements.

[2] https://huggingface.co/docs/diffusers

You will end up with 5 topics × 3 versions = **15 prompts in total**. Do not rely on LLMs to generate you these prompts and use your creativity to generate interesting prompts that you want to see visualized as images. For each prompt, generate the corresponding SD output. Use $\omega_{\text{CFG}} = 10.0$ with 50 sampling steps and $\eta = 0.0$. Keep the negative prompt `NULL`. Finally, assign each generated image a similarity score (out of 10) based on how well it matches the intended prompt (*e.g.*, 5.4/10.0). **Share your thoughts**. For instance, which version of a prompt typically gave the best similarity or image quality? Does prompt length affect fidelity to the intended concept or the final image quality?

(c) **[8 pts]** Implement CLIP-based similarity scoring (Radford et al., 2021) to automatically evaluate how well each generated image aligns with its corresponding prompt. Conceptually, CLIP computes embeddings for both the image and the text prompt in a shared feature space, and the similarity between them is measured using the cosine of the angle between their embeddings.

Similar to SD, you can find different pre-trained CLIP models (and their corresponding model cards with implementation details) in Hugging Face 🤗. Compare the CLIP similarity scores with your own manually assigned similarity ratings from the previous step. Discuss any trends, differences, or insights you observe and reflect on how well you believe this metric aligns with human judgment.

(d) **[4 pts]** Now select *one* of these 15 prompts and come up with a meaningful negative prompt to feed to SD. Fix number of steps to 50 and $\eta = 0.0$ but select $\omega_{\text{CFG}} \in \{0.0, 2.0, 5.0, 8.0, 12.0, 15.0\}$. Report the generated images and their corresponding CLIP similarity scores. **Comment on them**.

# References


Jooyoung Choi, Sungwon Kim, Yonghyun Jeong, Youngjune Gwon, and Sungroh Yoon. ILVR: Conditioning method for denoising diffusion probabilistic models. In *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, pp. 14367–14376, 2021.

Hyungjin Chung, Byeongsu Sim, Dohoon Ryu, and Jong Chul Ye. Improving diffusion models for inverse problems using manifold constraints. In *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, pp. 25683–25696, 2022.

Hyungjin Chung, Jeongsol Kim, Michael T Mccann, Marc L Klasky, and Jong Chul Ye. Diffusion posterior sampling for general noisy inverse problems. In *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2023.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recog. (CVPR)*, pp. 248–255, 2009.

Prafulla Dhariwal and Alexander Nichol. Diffusion models beat GANs on image synthesis. In *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, pp. 8780–8794, 2021.

Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. In *Proc. NeurIPS Workshop DGMs Appl.*, 2021.

Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of GANs for improved quality, stability, and variation. In *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2018.

Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, pp. 3730–3738, 2015.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *Proc. Int. Conf. Mach. Learn. (ICML)*, pp. 8748–8763, 2021.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recog. (CVPR)*, pp. 10684–10695, 2022.

Yinhuai Wang, Jiwen Yu, and Jian Zhang. Zero-shot image restoration using denoising diffusion null-space model. In *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2023.