CS726: Assignment-2
Denoising Diffusion Probabilistic Models

Mrudul Jambhulkar - 21D070044
Saurav Raj - 21D070064

Contents

Abstract

This report details our work on CS726 Programming Assignment 2, focusing on Denoising Diffusion Probabilistic Models (DDPMs). We implemented an unconditional DDPM, analyzed hyperparameter effects across multiple datasets (‘albatross‘, ‘moons‘, ‘circles‘, ‘manycircles‘, ‘blobs‘, ‘helix‘), and generated samples for the ‘albatross‘ dataset. We also outline our approach to Classifier-Free Guidance (CFG) with experimental results . **All code adheres to the provided environment, and we acknowledge the use of AI tools like ChatGpt for conceptual guidance and coding . Additionally other online sources are mentioned in references as well as in comments in code .**

1 Introduction

This assignment explores DDPMs and their extensions. Part 1.1 implements an unconditional DDPM, studies hyperparameters, and generates samples across six datasets. Part 1.2 extends this to CFG . Negative log likelihood (NLL) was used as an important metric to check the quality of samples generated .

2 Part 1.1: Denoising Diffusion Probabilistic Models

2.1 Implementation Details

- **Datasets**: ‘albatross‘, ‘moons‘, ‘circles‘, ‘manycircles‘, ‘blobs‘, ‘helix‘.
- **Hyperparameters**: Fixed learning rate = 0.01, batch size = 64, epochs = 100; varied $T = 10, 50, 100, 150, 200$.
- **Noise Schedule**: Linear with $\beta_1 = 0.0001$, $\beta_T = 0.02$.

The noise prediction network for the Denoising Diffusion Probabilistic Model (DDPM) is implemented as a PyTorch module designed to estimate the noise added to data at each timestep. The model consists of two main components: the time embedding module and the main noise prediction network.

The **Time Embedding Module** (`time_embed`) encodes the timestep information using a small feedforward network. A scalar timestep is first mapped to a higher-dimensional space through a linear layer, followed by a SiLU (Swish) activation to introduce non-linearity. Another linear layer further refines the embedding. This learned representation allows the model to capture time-dependent noise patterns effectively.

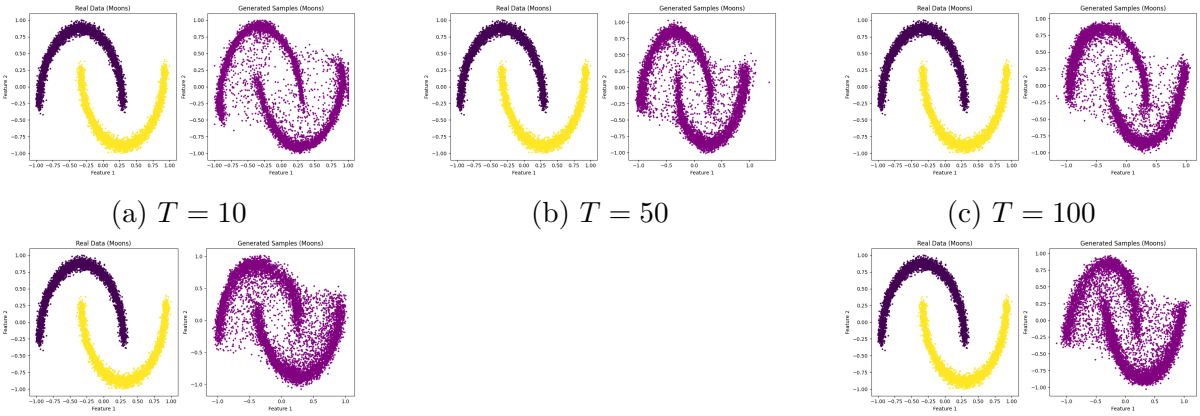
The **Main Noise Prediction Network** (`model`) is a three-layer Multi-Layer Perceptron (MLP) responsible for predicting the noise. The input data x and its corresponding timestep embedding t are concatenated before being passed through the MLP. Each hidden layer consists of 128 units with SiLU activation functions, ensuring smooth gradient flow. The final linear layer outputs a noise prediction of the same dimensionality as the input.

This structure enables the model to learn the evolution of noise over time, allowing it to effectively remove noise from corrupted samples during the reverse diffusion process.

2.2 Experiments and Results

2.2.1 Effect of Diffusion Steps T

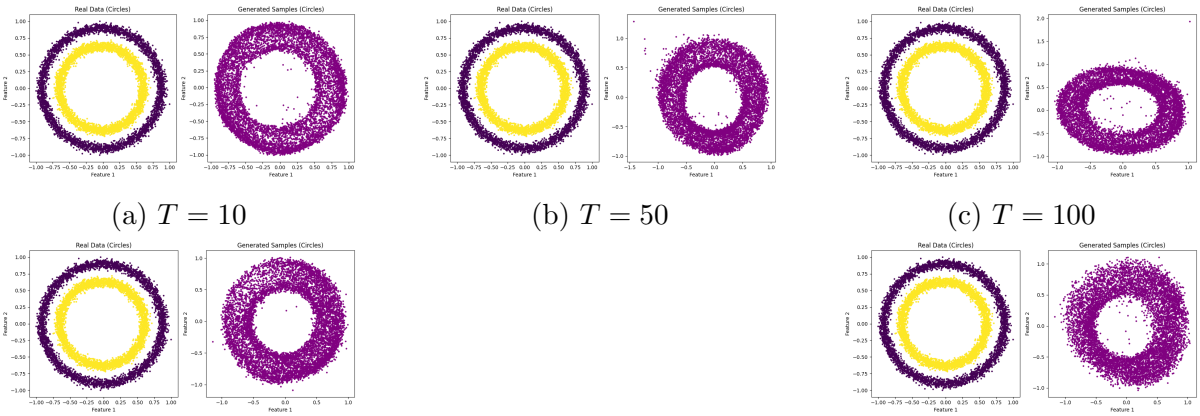
We trained DDPMs for $T = 10, 50, 100, 150, 200$ on all datasets. Sample images and NLL values are presented below:



(a) $T = 10$ (b) $T = 50$ (c) $T = 100$

(d) $T = 150$ (e) $T = 200$

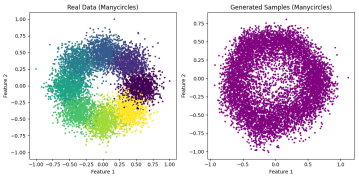
Figure 1: Samples for ‘moons‘ dataset across T values.



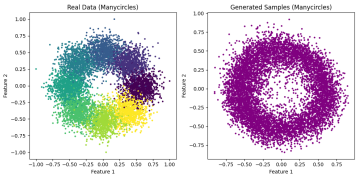
(a) $T = 10$ (b) $T = 50$ (c) $T = 100$

(d) $T = 150$ (e) $T = 200$

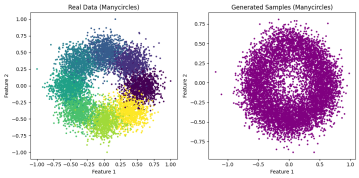
Figure 2: Samples for ‘circles‘ dataset across T values.



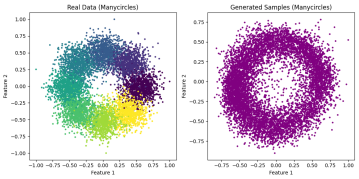
(a) $T = 10$



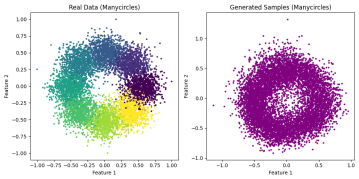
(b) $T = 50$



(c) $T = 100$

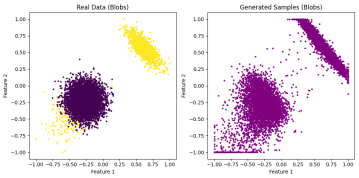


(d) $T = 150$

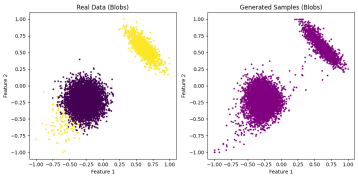


(e) $T = 200$

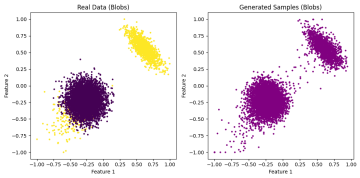
Figure 3: Samples for 'manycircles' dataset across T values.



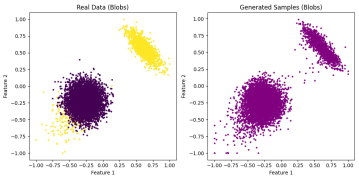
(a) $T = 10$



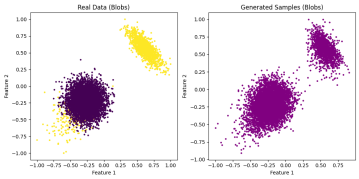
(b) $T = 50$



(c) $T = 100$

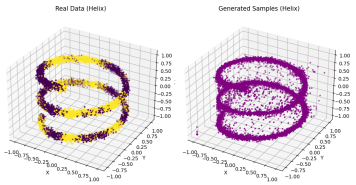


(d) $T = 150$

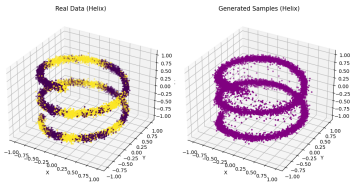


(e) $T = 200$

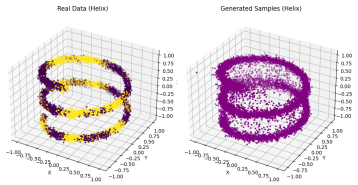
Figure 4: Samples for 'blobs' dataset across T values.



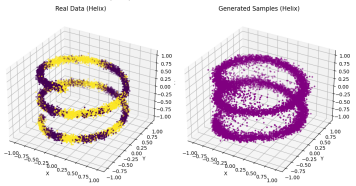
(a) $T = 10$



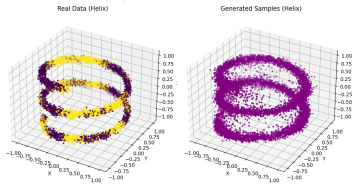
(b) $T = 50$



(c) $T = 100$



(d) $T = 150$



(e) $T = 200$

Figure 5: Samples for 'Helix' dataset across T values.

NLL Values Across Datasets and Timesteps:

Table 1: NLL values for different datasets at various T values.

Dataset	$T = 10$	$T = 50$	$T = 100$	$T = 150$	$T = 200$
moons	1.0068	0.9629	0.9551	0.9544	0.9361
circles	1.0209	0.9856	0.9924	0.9827	0.9794
manycircles	0.5993	0.5655	0.5194	0.5701	0.5402
blobs	0.3561	0.0588	0.0260	0.0181	0.0380
helix	1.5733	1.5335	1.5103	1.5257	1.5378

Table 2: Comparison of lbeta and ubeta values with corresponding NLL for "moons" dataset

lbeta	**ubeta**	**NLL**
0.0001	0.02	0.9361
0.0005	0.03	0.9189
0.00005	0.01	0.9553
0.001	0.05	0.9366
0.0002	0.025	0.9300

Observations: - NLL decreases as T increases, indicating better sample quality .

2.2.2 Noise Schedule

We tested the following noise schedules for the 'moons' dataset by running the `ddpm_noise.py` script: linear, cosine, sigmoid and quadratic. The linear, sigmoid, and quadratic schedule uses $lbeta = 0.0001$ and $ubeta = 0.02$, while cosine and schedules uses $s = 0.008$. Below, we present the NLL values obtained for each schedule at $T = 100$.

Linear Beta Schedule:

$$\beta_t = \beta_{\text{start}} + (\beta_{\text{end}} - \beta_{\text{start}}) \cdot \frac{t}{T-1}$$

Cosine Beta Schedule:

$$\alpha_t = \frac{f(t)}{f(0)} \quad \text{where} \quad f(t) = \cos\left(\frac{t/T + s}{1+s} \cdot \frac{\pi}{2}\right)^2$$

$$\beta_t = 1 - \frac{\bar{\alpha}_t}{\bar{\alpha}_{t-1}}$$

Quadratic Beta Schedule:

$$\beta_t = \beta_{\text{start}} + (\beta_{\text{end}} - \beta_{\text{start}}) \cdot \left(\frac{t}{T-1}\right)^2$$

Sigmoid Beta Schedule:

$$\beta_t = \text{sigmoid}(x_t) \cdot (\beta_{\text{end}} - \beta_{\text{start}}) + \beta_{\text{start}}$$

Table 3: NLL values for different noise schedules on the 'moons' dataset at $T = 100$.

Noise Schedule	NLL
Linear	0.9549
Cosine	1.3403
Sigmoid	0.9321
Quadratic	0.9511

2.2.3 Albatross Dataset

`ddpm_albatross.ipynb` contains the code for training , sampling and saving the `model_albatross.pth` model . Using following hyperparameters, we trained on 'albatross' with initial noise from

`albatross_priority_samples.npy`:

- $n\_steps = 10$
- $lbeta = 0.0001$
- $ubeta = 0.02$
- $epochs = 50$
- $batchsize = 64$
- $lr = 0.01$

NLL value for the generated samples was 10.3965 .

Samples are reproducible via `reproduce.py`, saved as `albatross_samples_reproduce.npy`.

2.3 Discussion

Higher T generally improves sample quality across datasets, with marginal gains past $T = 100$, reflecting sufficient denoising. NLL trends suggest dataset-specific optimal T values (e.g., 'helix' may plateau earlier due to its structure). As cumulative noise increases it may overcorrupt the data . But , higher number of time steps allow the model to learn and adapt better especially in "low density regions" .

3 Part 1.2: Classifier-Free Guidance

3.1 Training free method

CFG enhances conditional sampling without an explicit classifier. Instead of training a separate classifier model, we train and unconditional DDPM model $p_\theta(z)$ is parameterized through a score estimator $\epsilon_\theta(z_\lambda)$, together with the conditional model $p_\theta(z_j \mid c)$ parameterized through $\epsilon_\theta(z_\lambda; c)$. We will modify `ConditionalDDPM` to accept label inputs, training it to predict noise $\epsilon_\theta(x_t, t, y)$ (conditioned on class y) and $\epsilon_\theta(x_t, t, \emptyset)$ (unconditioned). Both the conditional and uncomditional models are parameterized by same neural network and are jointly trained by randomly setting c (class label) to the unconditional class identifier (null token- $\emptyset$) with some probability p_{uncond}, a hyperparameter we fixed as 0.1 . Sampling uses:

$$\tilde{\epsilon}_\theta(x_t, t, y) = (1 + w)\epsilon_\theta(x_t, t, y) - w\epsilon_\theta(x_t, t, \emptyset),$$

where w is the guidance scale. For classification, we propose a training-free method: run the reverse process for each class y, compute the likelihood $p(x|y)$ via NLL, and select the class with the highest likelihood.

3.2 Guided vs. Conditional Sampling

Guided sampling (CFG) adjusts the denoising direction using w, offering flexibility over conditional sampling's fixed $p(x|y)$. **Guidance Scale**: Higher w may sharpen class-specific features but risks overfitting. In conditional sampling , samples are generated provided they satisfy a given condition / target label . In guidance sampling , a classifier/guidance signal is used to identify the target condition and generate samples . Classifier-free guidance, our guidance method which avoids any classifier entirely. Rather than sampling in the direction of the gradient of an image classifier, classifier-free guidance instead mixes the score estimates of a conditional diffusion model and a jointly trained unconditional diffusion model .

3.3 Implementation Details

- **Datasets**: 'moons', 'circles', 'manycircles', 'blobs', 'helix'.
- **Hyperparameters**: Fixed learning rate = 0.01, batch size = 64, epochs = 100; $T = 200$.
- **Noise Schedule**: Linear with $\beta_1 = 0.0001$, $\beta_T = 0.02$, varied the guidance scale

The conditional Denoising Diffusion Probabilistic Model (Conditional DDPM) extends the standard DDPM by incorporating class conditioning. This is achieved by embedding class labels and concatenating them with the input data and time embeddings.

The model consists of three key components:

Time Embedding Module: A feedforward network maps the timestep to an embedding space using two linear layers with SiLU activations.

Class Embedding Module: A learnable embedding layer encodes class labels into a vector representation.

Noise Prediction Network: A three-layer Multi-Layer Perceptron (MLP) processes the concatenated input, timestep, and class embeddings to predict the noise.

The classifier, `ClassifierDDPM`, uses the trained Conditional DDPM to classify samples. It computes class probabilities by measuring the likelihood of noise predictions for different class labels. The training framework incorporates both conditional and unconditional training, where class labels are randomly masked with a probability p_{uncond} to improve generalization. The loss function is the Mean Squared Error (MSE) between the predicted and actual noise, optimized using stochastic gradient descent.

3.4 Experiments and Results

We trained a conditional DDPM on the 'moons' dataset and studied the effects of guidance scale w values ($w = 0.5, 1, 1.5, 2, 2.5$) on sample quality and classification accuracy. Sample quality is evaluated using Negative Log-Likelihood (NLL) from `utils.py`, while accuracy is assessed using a pre-trained classifier to determine if generated samples match the intended class. Results are summarized in the table below.

Table 4: NLL for the 'moons' dataset across different guidance scale values.

Guidance Scale (w)	NLL
0.5	0.8955
1.0	0.9522
1.5	0.9650
2.0	0.9681
2.5	0.9722

3.4.1 Classification results

The hyperparameters chosen for computing these results are :

- $n\_steps = 200$
- $lbeta = 0.0001$
- $ubeta = 0.02$
- $epochs = 100$

- $batchsize = 64$
- $lr = 0.01$

Table 5: Average classification accuracy over various datasets

Dataset	Classification accuracy
Moons	61.155%
Circles	53.610%
Blobs	53.580%
Helix	50.635%

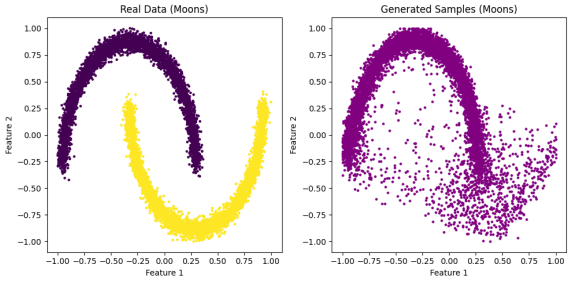


Figure 6: Moons dataset class 0. Classification accuracy for predicted class 0: 60.02% (based on 4407 samples predicted as class 0 with guidance scale 0.5 during generation).

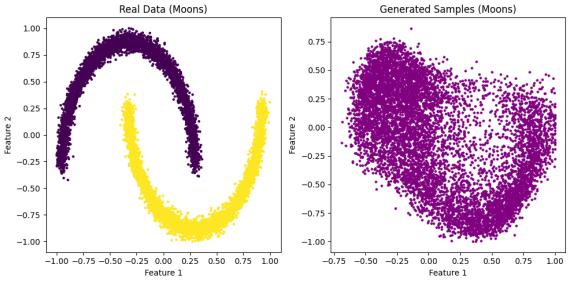


Figure 7: Moons dataset class 1. Classification accuracy for predicted class 1: 62.29% (based on 3593 samples predicted as class 1 with guidance scale 0.5 during generation).

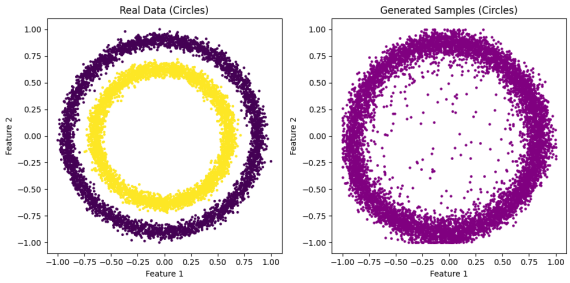


Figure 8: Circles dataset class 0. Classification accuracy for predicted class 0: 53.34% (based on 4299 samples predicted as class 0 with guidance scale 0.5 during generation).

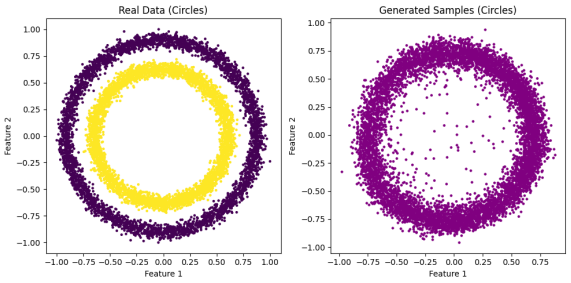


Figure 9: Circles dataset class 1. Classification accuracy for predicted class 1: 53.88% (based on 3701 samples predicted as class 1 with guidance scale 0.5 during generation).

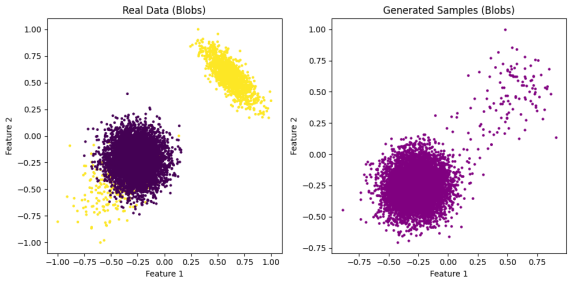


Figure 10: Blobs dataset class 0. Classification accuracy for predicted class 0: 84.01% (based on 3983 samples predicted as class 0 with guidance scale 0.5 during generation).

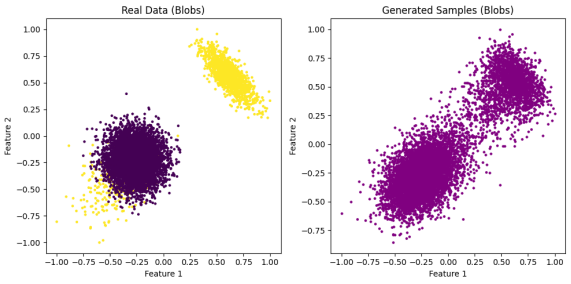


Figure 11: Blobs dataset class 1. Classification accuracy for predicted class 1: 23.15% (based on 4017 samples predicted as class 1 with guidance scale 0.5 during generation).

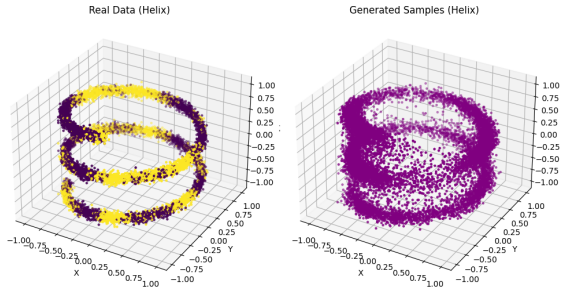


Figure 12: Helix dataset class 0. Classification accuracy for predicted class 0: 52.04% (based on 5027 samples predicted as class 0 with guidance scale 0.5 during generation).

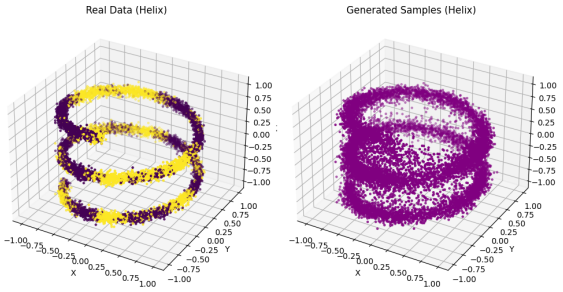


Figure 13: Helix dataset class 1. lassification accuracy for predicted class 1: 49.23% (based on 4973 samples predicted as class 1 with guidance scale 0.5 during generation).

4 References

- ChatGPT
- `https://www.kaggle.com/code/vikramsandu/ddpm-from-scratch-in-pytorch`
- `https://colab.research.google.com/github/huggingface/notebooks/blob/main/examples/annotated_diffusion.ipynb#scrollTo=5d751df2`
- `https://github.com/TeaPearce/Conditional_Diffusion_MNIST/tree/main`

5 Individual Contribution

Both team members actively discussed and collaboratively wrote the entire `ddpm.py` file, along with most sections of the code. However, some specific tasks were handled individually like:

- **DDPM:**
  - Training , sampling and reproduce on the albatross dataset and experimenting with different values of β_{low} and β_{high} – **Mrudul**
  - Implementing different noise schedules – **Saurav**
- **CFG:**
  - Implementing classifier and accuracy prediction – **Mrudul**
  - Experimenting with different guidance scales – **Saurav**

6 README

```
ddpm.py : main code
ddpm_noise.py : ddpm code with different noise schedules
ddpm_albatross.ipynb : code to train/sample albatross dataset
run_code_for_ddpm_noise.txt : contains cmd prompt script to run ddpm_noise for
various noise schedules and hyperparameters
run_code_for_ddpmCFG.txt : contains cmd prompt script to run for CFG model
(runs DDPM by default) and select hyperparameters
```