Lab6 - Generative Models

He-Bi Yang

- **Deadline: May 6, 2025, 11:59 p.m.**
- **Format: Experimental report (.pdf) , source code (.py) and images folder.**
  Zip all files in one file and name it DL_LAB6_YourStudentID_YourName.zip.

1. Lab Description

In this lab, you need to implement a conditional Denoising Diffusion Probabilistic Model (DDPM) to generate synthetic images according to multi-label conditions: Given a specific set of label conditions, your DDPM should generate the corresponding synthetic images (see Fig. 1). For example, “red sphere,” “yellow cube” and “gray cylinder” are given, your model should generate the synthetic images containing those objects. Meanwhile, you also need to input your generated images into a pre-trained classifier for evaluation.

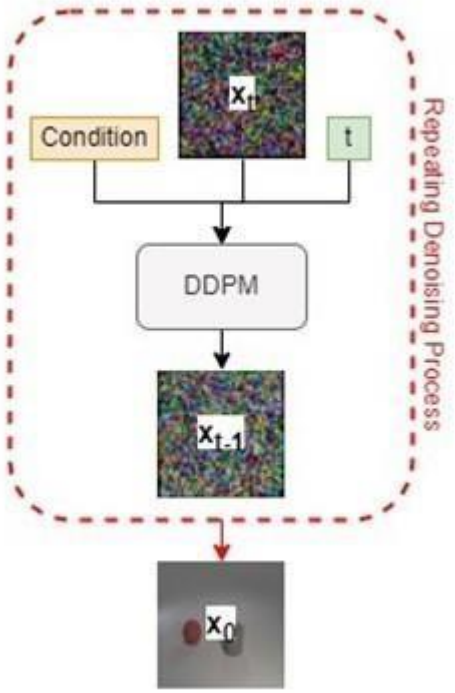

Fig. 1: The overview of conditional DDPM.

2. Requirements

- Prepare dataset

- Implement data loader
- Define your multi-label condition embedding method

- Implement a conditional DDPM

- Choose your conditional DDPM setting, design your model architecture
- Design your noise schedule, time embeddings, sampling method, etc. You can also include the pretrained evaluator as a “classifier guidance” in sampling
- Choose your loss functions (e.g. reparameterization type)
- Implement the training function, testing function **- Evaluate the results** ● Evaluate the accuracy of test.json and new test.json, input the output images of each model to the pretrained evaluator to get the accuracies
- Show the synthetic images in grids for two testing files
- Show the denoising process in a grid for sampling an image from your DDPM, **with the label set ["red sphere", "cyan cylinder", "cyan cube"]**

3. Implementation Details

3.1 Basic Rules

- For each model, **you can choose any architecture you like**. Also, feel free to use any library that helps your work, but you need to mention the implementation details and include the reference in your report.
- Except for the provided files, you cannot use other training data. (e.g. background image)
- Use the function of a pre-trained evaluator to compute the accuracy of your synthetic images.
- The evaluator is based on ResNet18. By inputting synthesized images and one-hot GT labels, the evaluator will return accuracy. Its class script and pretrained weight are given. **DO NOT modify any of them, BUT involving the classification result in your discriminator or classifier guidance is allowed. Please inherit the class if you need extra functionalities.**
- The normalization of input images is ***transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))***. **You must input the normalized image into the pretrained classifier**. You should apply denormalization to generate RGB images.
- The resolution of input for the pretrained evaluator is 64x64. You can design your own output resolution and resize it for evaluation
- Use ***make_grid(images)*** from torchvision.utils to generate grids for your testing image (8 images a row, 4 rows for each JSON file) and denoising process (at least 8 images in 1 row from noisy to clear).

- If you have no idea how to design your DDPM, follow 3.2 for simple suggestions.

3.2 DDPM

- Denoising Diffusion Probabilistic Models
- Hugging Face Diffusion Models Course

You can refer to the tutorial linked above to design your diffusion model for this lab.

4. File Descriptions

You can download file.zip from e3 or ntu cool, except iclevr.zip, from the open-source Google Drive (link). There are 7 files in the file.zip: readme.txt, train.json, test.json, new_test.json, object.json, evaluator.py, and checkpoint.pth. All the details of the dataset are in the readme.txt.

5. Scoring Criteria

- Report (60%)

- Introduction (5%)
- Implementation details (25%)
  - Describe how you implement your model, what is this step, including your choice of DDPM, noise schedule. (Please write in detail.)
- Results and discussion (30%)
  - Show your synthetic image grids (total 16%: 8% * 2 testing data) and a denoising process image **with the label set ["red sphere", "cyan cylinder", "cyan cube"]** (4%)
  - Discussion of your extra implementations or experiments (10%)

- Experimental results (40%) (based on results shown in your report)

- Classification accuracy on test.json and new test.json (40%)
- Show your accuracy screenshots (20% * 2 testing data)

Accuracy	Grade
score ≥ 0.8	100%
$0.8 >$ score ≥ 0.7	90%
$0.7 >$ score ≥ 0.6	80%
$0.6 >$ score ≥ 0.5	70%
$0.5 >$ score ≥ 0.4	60%
score < 0.4	0%

6. Save Images

- Save your synthetic images for the 2 testing data (test.json , new_test.json)
  Save in png files !!!
- The data structure will be organized as follows:

```
images/
├── test/
│   ├── 0.png
│   ├── 1.png
│   ├── ...
│
├── new_test/
│   ├── 0.png
│   ├── 1.png
│   ├── ...
```

- The image name should follow the order in the test.json and new_test.json.

7. Output Examples

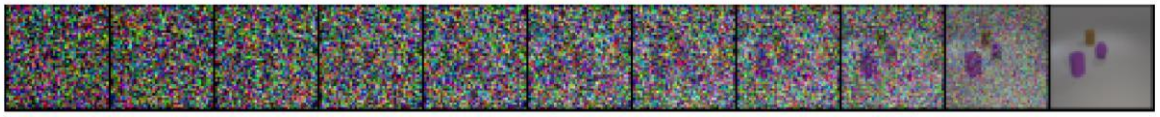

Fig. 2: Example of the denoising process image.

Fig. 3: The synthetic image grid on new_test.json. (F1-score 0.821)