# Lab6 Report

Deep Learning

Generative Models

姓名：陳科融

學號：314551010

Aug 7, 2025

# 1. INTRODUCTION

這次的 Lab 實作一個基於 Denoising Diffusion Probabilistic Models（DDPM）的條件式圖像生成模型，其能夠根據給定的文字描述（多標籤）生成對應的圖像。我們將說明模型的架構設計、訓練策略與實作細節，並以 iCLEVR 資料集為基礎，展示其在多物體複雜場景下的生成效果與性能評估。

擴散模型是一種透過逐步去噪的反向過程，從純高斯噪聲中還原出清晰圖像的方法。本次 Lab 標是實現一個能夠理解多標籤語意條件，並生成符合描述的複雜場景圖像的生成模型，驗證其在條件式生成任務上的表現與潛力。

# 2. IMPLEMENTATION DETAILS

## 2.1 噪聲排程器（Noise Scheduler）

本專案採用了 DDPM 的核心架構，並在其中實作了自訂的噪聲排程器（Noise Scheduler）來控制前向加噪過程中每個時間步 t 所加入的噪聲強度。

### Squared Cosine Schedule

我們選擇了 squaredcos_cap_v2（平方餘弦）作為噪聲排程策略。相較於傳統的線性排程，平方餘弦排程在時間步 t 較小與較大時的噪聲增量 $\beta_t$ 變化較平緩，而在中間時段變化較快。在初始化階段，排程器會根據所選的策略預先計算所有時間步的參數，包括：

- $\beta_t$：每一步加入的噪聲比率
- $\alpha_t = 1 - \beta_t$：每一步保留的訊號比例
- $\overline{\alpha_t} = \prod_{s=1}^{t} \alpha_s$ ：從第 1 步到第 t 步的累積訊號比例

加噪函數 add_noise(x_0, t, noise)：

在訓練階段將乾淨圖像$x0$加入噪聲，依據以下公式生成加噪後的圖像 $xt$：

$$x_t = \sqrt{\overline{\alpha_t}} \cdot x_0 + \sqrt{1 - \overline{\alpha_t}} \cdot \epsilon$$

$\overline{\alpha_t}$ 是從第 1 步到第 t 步所有 α的累積乘積

$\epsilon \sim \mathcal{N}(0, I)$為標準高斯噪聲

去噪步驟：step(x_t, t, eps_pred)

在推論階段，我們從純噪聲開始，透過多步去噪逐步恢復圖像。根據模型預測的噪聲 $\epsilon_\theta$，我們計算$x_{t-1}$：

$$x_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left( x_t - \frac{1 - \alpha_t}{\sqrt{1 - \overline{\alpha_t}}} \cdot \epsilon_\theta \right) + \sigma_t \cdot z$$

$\epsilon_\theta$為模型預測的噪聲

$\sigma_t$為該步驟的標準差（由排程器決定）

$z \sim \mathcal{N}(0, I)$ 為隨機噪聲，僅在 t > 1 時加入

```
noise_scheduler = DDPMScheduler(
    num_train_timesteps=args.num_train_timesteps,
    beta_schedule=args.beta_schedule
)
```

程式碼 1. 使用 diffuser 提供的 DDPMScheduler

## 2.2 Model Architecture

我們的核心模型 DiffusionModel 由兩部分組成：一個負責處理條件的 class_embedding 層，以及一個作為骨幹網路的 U-Net 模型。我們參照 diffusers 的 UNet2DModel，從零開始實現了一個功能完全對應的 UNet2DModel。

### ResBlock2D

U-Net 的基本建構單元，包含兩個卷積層、GroupNorm、SiLU 激活函數和殘差連接，它是條件注入的主要位置。

```python
def __init__(self, in_channels, out_channels, time_emb_dim, dr
    super().__init__()

    # 時間嵌入的線性變換層
    self.time_mlp = nn.Linear(time_emb_dim, out_channels)

    # 第一個卷積塊
    self.conv1 = nn.Sequential(
        nn.GroupNorm(8, in_channels),
        nn.SiLU(),
        nn.Conv2d(in_channels, out_channels, 3, padding=1)
    )

    # 第二個卷積塊
    self.conv2 = nn.Sequential(
        nn.GroupNorm(8, out_channels),
        nn.SiLU(),
        nn.Dropout(dropout),
        nn.Conv2d(out_channels, out_channels, 3, padding=1)
    )

    # 殘差連接的通道匹配層
    if in_channels != out_channels:
        self.residual_conv = nn.Conv2d(in_channels, out_channe
    else:
        self.residual_conv = nn.Identity()
```

程式碼 2. ResBlock2D 程式碼

## SelfAttention2D

自注意力模組，用於在較低解析度的特徵圖上捕捉全局依賴關係，增強模型的空間理解能力。

```python
def forward(self, x):
    b, c, h, w = x.shape
    x_norm = self.norm(x)

    # 計算 Q, K, V
    qkv = self.to_qkv(x_norm)
    q, k, v = qkv.chunk(3, dim=1)

    # 重塑為 (batch, heads, seq_len, head_dim)
    q = q.view(b, self.num_heads, c // self.num_heads, h * w).transpose(2, 3)
    k = k.view(b, self.num_heads, c // self.num_heads, h * w).transpose(2, 3)
    v = v.view(b, self.num_heads, c // self.num_heads, h * w).transpose(2, 3)

    # 計算注意力權重
    attn = torch.softmax(torch.matmul(q, k.transpose(-2, -1)) * self.scale, dim=-1)

    # 應用注意力權重
    out = torch.matmul(attn, v)
    out = out.transpose(2, 3).reshape(b, c, h, w)

    # 輸出投影
    out = self.to_out(out)

    return x + out
```

程式碼 3. Attention Block 程式碼

## Encoder

輸入的圖片先經過一個卷積層處理，然後依序通過一連串的 DownBlock2D 和 AttnDownBlock2D。在每個 ResBlock2D 裡，我們把處理過的 cond_emb 條件向量，先經過一個線性層，再直接加到卷積輸出上，這樣就把條件資訊放進去，每一層在下採樣之前的輸出，都會存起來當作後面解碼時用的（Skip Connection）。

```
# 構建下採樣塊
self.down_blocks = nn.ModuleList()
for i, block_type in enumerate(down_block_types):
    in_ch = block_out_channels[i - 1] if i > 0 else block_out_channels[0]
    out_ch = block_out_channels[i]

    if block_type == "DownBlock2D":
        block = DownBlock2D(in_ch, out_ch, time_embed_dim, layers_per_block)
    elif block_type == "AttnDownBlock2D":
        block = AttnDownBlock2D(in_ch, out_ch, time_embed_dim, layers_per_block)
    else:
        raise ValueError(f"Unknown down_block_type: {block_type}")

    self.down_blocks.append(block)
```

程式碼 4. Encoder 程式碼

## Decoder

輸出的特徵圖會一路經過一系列 UpBlock2D 和 AttnUpBlock2D，在每個上採樣階段一開始，我們會把當前的特徵圖跟對應的 Skip Connection 拼在一起（用 torch.cat），為了可以把高解析度的細節帶回來，讓最後生成的圖更清楚。條件向量 cond_emb 也會在上採樣路徑的每個 ResBlock2D 裡注入，跟 Encoder 的做法一樣，最後特徵圖經過一個輸出卷積層，得到跟輸入圖片同尺寸的噪聲預測。

```
# 構建上採樣塊
self.up_blocks = nn.ModuleList()
reversed_block_out_channels = list(reversed(block_out_channels))

for i, block_type in enumerate(up_block_types):
    in_ch = reversed_block_out_channels[i]
    skip_ch = reversed_block_out_channels[i]  # 跳躍連接的通道數

    if i == len(up_block_types) - 1:
        out_ch = block_out_channels[0]  # 最後一層輸出到初始通道數
    else:
        out_ch = reversed_block_out_channels[i + 1]

    if block_type == "UpBlock2D":
        block = UpBlock2D(in_ch, skip_ch, out_ch, time_embed_dim, layers_per_blo
    elif block_type == "AttnUpBlock2D":
        block = AttnUpBlock2D(in_ch, skip_ch, out_ch, time_embed_dim, layers_per
    else:
        raise ValueError(f"Unknown up_block_type: {block_type}")

    self.up_blocks.append(block)

# 輸出層
self.conv_norm_out = nn.GroupNorm(8, block_out_channels[0])
self.conv_out = nn.Conv2d(block_out_channels[0], out_channels, 3, padding=1)
```

程式碼 5. Decoder 程式碼

## DDPM

本模型在每個時間步 t，根據加噪圖像 xt 和條件（例如 one-hot 標籤）預測原始高斯噪聲 $\epsilon_\theta(x_t, t)$。此步驟是逆向擴散過程的核心，因為重建乾淨圖像需要依靠該噪聲預測。

DDPM 的損失函數

$$L_{\text{simple}} = E_{x_0,t,\epsilon\sim\mathbb{N}(0,I)}[|\epsilon - \epsilon_\theta(x_t, t, y)|^2]$$

U-Net 編碼器-解碼器結構

| 類型 | 輸入尺寸 | 輸出尺寸 | 通道變化 |
|---|---|---|---|
| DownBlock2D | 64×64 | 32×32 | 3 → 128 |
| DownBlock2D | 32×32 | 16×16 | 128 → 128 |
| DownBlock2D | 16×16 | 8×8 | 128 → 256 |
| DownBlock2D | 8×8 | 4×4 | 256 → 256 |
| AttnDownBlock2D | 4×4 | 2×2 | 256 → 512 |
| DownBlock2D | 2×2 | 1×1 | 512 → 512 |
| UpBlock2D | 1×1 | 2×2 | 512 → 512 |
| AttnUpBlock2D | 2×2 | 4×4 | 512 → 512 |
| UpBlock2D | 4×4 | 8×8 | 512 → 256 |
| UpBlock2D | 8×8 | 16×16 | 256 → 256 |
| UpBlock2D | 16×16 | 32×32 | 256 → 128 |
| UpBlock2D | 32×32 | 64×64 | 128 → 3 |

時間嵌入（Time Embedding）

- 將時間步 t 編碼為高維向量（採用 sin/cos 位置編碼），再經 MLP 投影，注入至 ResBlock，讓模型知道當前擴散階段。

條件嵌入（Class Embedding）

- 將多標籤 one-hot 向量轉換為條件特徵，與時間嵌入結合，支援條件生成（Conditional Generation）。

```
# 核心組件：自定義實現的 UNet2DModel
# 這裡的結構參數完全對應 diffusers 的配置，旨在平衡模型容量與計算效率。
self.model = UNet2DModel(
    sample_size=image_size,  # 輸入圖像尺寸
    in_channels=3,  # 輸入通道數 (RGB)
    out_channels=3,  # 輸出通道數 (預測的噪聲也是 RGB)
    layers_per_block=2,  # 每個 U-Net 區塊的卷積層數
    block_out_channels=(128, 128, 256, 256, 512, 512),  # U-Net 各
    class_embed_type="identity",  # 條件嵌入類型, "identity" 表示我們
    down_block_types=(  # U-Net 編碼器（下採樣）的區塊類型
        "DownBlock2D", "DownBlock2D", "DownBlock2D", "DownBlock2D"
        "AttnDownBlock2D",  # 在中低解析度層加入注意力機制，捕捉全局特徵
        "DownBlock2D",
    ),
    up_block_types=(  # U-Net 解碼器（上採樣）的區塊類型
        "UpBlock2D",
        "AttnUpBlock2D",  # 同樣加入注意力機制
        "UpBlock2D", "UpBlock2D", "UpBlock2D", "UpBlock2D",
    ),
)

# 條件嵌入層：將 one-hot 標籤轉換為高維嵌入向量
# 使用一個小型的 MLP (多層感知機) 來增強其表達能力
self.class_embedding = nn.Sequential(
    nn.Linear(num_classes, class_embed_dim),
    nn.SiLU(),  # SiLU 是一個平滑且高效的非線性激活函數
    nn.Linear(class_embed_dim, class_embed_dim),
)
```

程式碼 6. DDPM 程式碼架構

## 2.3 Dataloader

### 讀取資料與標籤

- 訓練模式：從 train.json 載入圖像檔名與對應標籤，並讀取實際圖片。
- 測試模式：從 test.json 或 new_test.json 載入條件標籤。

### 標籤處理

- 將每個樣本的物件名稱轉換成 **one-hot 向量**，長度等於物件類別數

### 圖像處理與增強

- 圖像 Resize 到 64×64，並正規化到[−1,1]範圍。

- 訓練時額外做隨機水平翻轉，增加資料多樣性。

## 批次讀取與加速

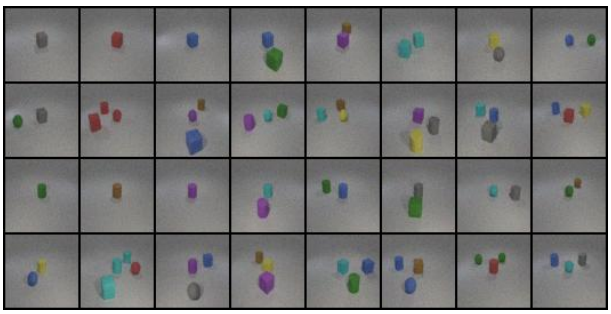
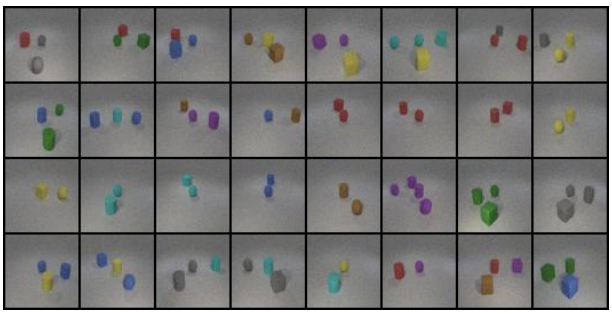
- 透過 **DataLoader** 將資料打包成批次，支援多線程讀取與 GPU 傳輸加速。
- 訓練時打亂順序（shuffle），測試時保持順序。

```python
def get_dataloader(data_dir: str, image_dir: str, mode: str, batch_size: i
                   num_workers: int = 4) -> DataLoader:
    # 定義圖像轉換流程
    transform_list = [transforms.Resize((64, 64))]
    if mode == 'train':
        # 只在訓練時進行數據增強（隨機水平翻轉）
        transform_list.append(transforms.RandomHorizontalFlip())

    transform_list.extend([
        transforms.ToTensor(),  # 將 PIL 圖像轉換為 PyTorch 張量
        transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])  #
    ])
    transform = transforms.Compose(transform_list)

    objects_json_path = os.path.join(data_dir, 'objects.json')

    dataset = IclevrDataset(
        data_dir=data_dir,
        image_dir=image_dir,
        mode=mode,
        objects_json_path=objects_json_path,
        transform=transform
    )

    # 如果未指定 shuffle，則根據模式自動決定 (訓練時打亂，測試時不打亂)
    if shuffle is None:
        shuffle = (mode == 'train')

    dataloader = DataLoader(
        dataset,
        batch_size=batch_size,
        shuffle=shuffle,
        num_workers=num_workers,
        pin_memory=True  # 如果記憶體充足，可以加速從 CPU 到 GPU 的數據傳輸
    )
    return dataloader
```

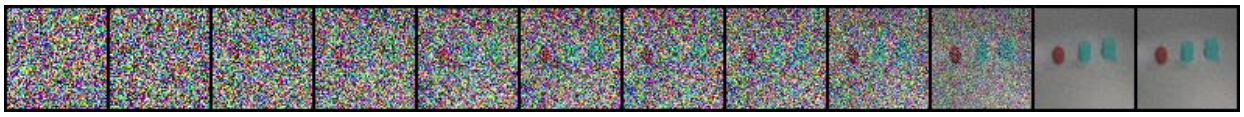
程式碼 7. 讀取 iCLEVR 資料集

# 3 RESULTS AND DISCUSSION

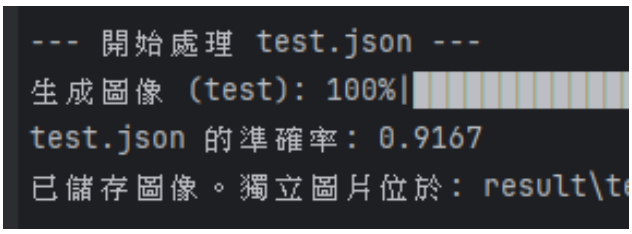
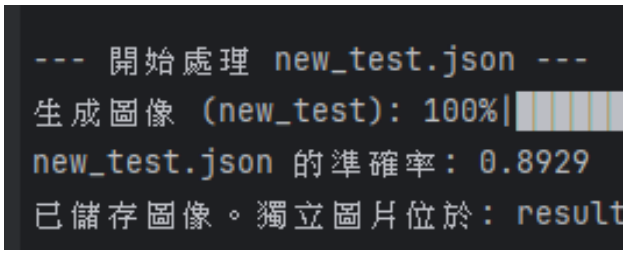
## 3.1 synthetic image grids

| test_grid | new_test_grid |
|---|---|
|  |  |

## 3.2 denosing process image

| denosing_process |
|---|
|  |

## 3.3 Experimental results

| | |
|---|---|
| test.json |  |
| new_test.json |  |

## 3.4 Instruction

| | |
|---|---|
| train | python src/train.py --epochs 500 |
| test | python src/test.py --ckpt_path "output\ddpm-cfg-v1\checkpoints\model_epoch_500.pth" |
| guidance scale | python src/guidance_scale.py --ckpt_path "output\ddpm-cfg-v1\checkpoints\model_epoch_500.pth" |

## 3.5 Parameter

| 參數 | 設定值 | 說明 |
|---|---|---|
| epochs | 500 | 訓練的總輪數。 |
| batch_size | 64 | 每一批次訓練的樣本數。 |
| lr | 1e-4 | 學習率，控制模型權重更新的幅度。 |
| lr_warmup_steps | 500 | 學習率從 0 增加到設定值 lr 所需的步數。 |
| num_train_timesteps | 500 | 擴散過程總時間步數，步數越少生成越快。 |
| beta_schedule | squaredcos_cap_v2 | 噪聲排程策略，決定如何隨時間添加噪聲。 |
| cfg_dropout_prob | 0.1 | CFG 中，隨機丟棄文字條件的機率 (10%)。 |

## 3.6 Extra implementations or experiments

### Wandb


圖 1. 訓練數據紀錄

## Classifier-Free Guidance

為了讓模型生成的圖像能精準地符合輸入的文字標籤，我們實現並採用了當前最先進的無分類器引導 (Classifier-Free Guidance, CFG) 技術。這項技術巧妙地讓單一模型同時具備有條件和無條件的生成能力，從而無需訓練一個額外的分類器來引導擴散過程。

```
# Classifier-Free Guidance 訓練: 以一定機率丟棄條件
if torch.rand(1).item() < args.cfg_dropout_prob:
    labels = torch.zeros_like(labels)  # 使用全零向量代表「無條件」
```

程式碼 8. CFG 程式碼

**Guidance Scale (w)** 控制了模型在生成過程中**對條件的依賴程度**。

- **小（~1.0）**：模型更自由，多樣性高，但對條件的遵守程度低。
- **中等 (3.0–7.5）**：平衡條件準確率與圖像多樣性，最常用範圍。
- **大（>10）**：條件強制效果強，結果高度符合條件，但圖像可能過飽和。

$$\tilde{\varepsilon} = \varepsilon_{\text{uncond}} + w \cdot (\varepsilon_{\text{cond}} - \varepsilon_{\text{uncond}})$$

$\tilde{\varepsilon}$：最終的噪聲預測

$\varepsilon_{\text{uncond}}$：無條件噪聲預測（忽略標籤）

$\varepsilon_{\text{cond}}$：有條件噪聲預測（使用標籤）

w：guidance scale（引導強度）

```
# 為了模型, 創建一個在 GPU 上的時間步張量
t_for_model = t.to(device).expand(latent_model_input.shape[0])
noise_pred = model(latent_model_input, t_for_model, combined_labels)
noise_pred_cond, noise_pred_uncond = noise_pred.chunk(2)
guided_noise = noise_pred_uncond + args.guidance_scale * (noise_pred_cond - noise_pred_uncond)

將 CPU 上的 t 傳遞給 scheduler
nage = noise_scheduler.step(guided_noise, t, image).prev_sample
```

程式碼 9.　Guidance noise 程式

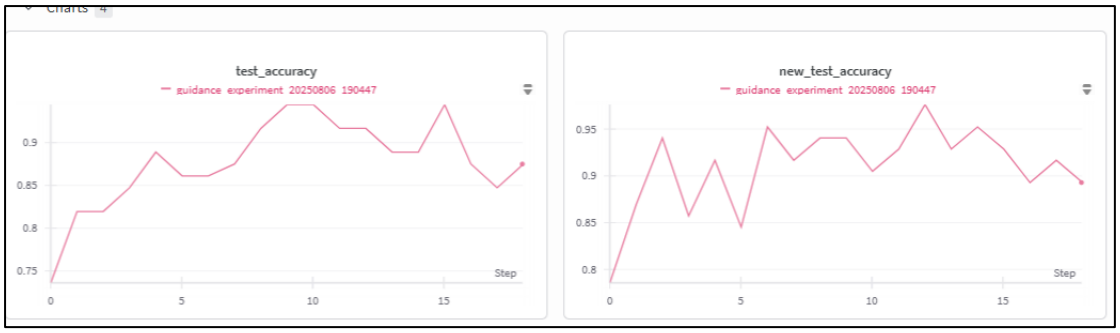



圖 2. 不同 Guidance Scale 圖表

```
============================================================
📊 實驗結果總結
============================================================
Guidance Scale  Test Acc     New Test Acc    Average
------------------------------------------------------------
1.0             0.7361       0.7857          0.7609
1.5             0.8194       0.8690          0.8442
2.0             0.8194       0.9405          0.8800
2.5             0.8472       0.8571          0.8522
3.0             0.8889       0.9167          0.9028
3.5             0.8611       0.8452          0.8532
4.0             0.8611       0.9524          0.9067
4.5             0.8750       0.9167          0.8958
5.0             0.9167       0.9405          0.9286
5.5             0.9444       0.9405          0.9425
6.0             0.9444       0.9048          0.9246
6.5             0.9167       0.9286          0.9226
7.0             0.9167       0.9762          0.9464
7.5             0.8889       0.9286          0.9087
8.0             0.8889       0.9524          0.9206
8.5             0.9444       0.9286          0.9365
9.0             0.8750       0.8929          0.8839
9.5             0.8472       0.9167          0.8819
10.0            0.8750       0.8929          0.8839

============================================================
📝 正在上傳總結報告至 W&B...
============================================================
✅總結報告已上傳！

🏆 最佳 Guidance Scale: 7.0
   平均準確率: 0.9464
```

圖 3. 不同 Guidance Scale 數據

## DDIM

DPM 的反向去噪過程是一個馬可夫鏈（Markov Chain），意思是每一步 $x_{t-1}$ 的生成都只依賴於前一步$x_t$，並且每一步都引入了微小的隨機性。它在不需重新訓練 DDPM 模型的前提下，提出了一種全新的、更快速且確定性的採樣方法。它將原本隨機的生成過程，變成了一個可以控制的、確定性的路徑。

DDPM 的問題：

- **速度慢：** 由於過程是隨機的馬可夫鏈，為了保證生成品質，必須走完非常多的步數（例如 1000 步），導致推論速度極慢。
- **不確定性：** 即使給定相同的初始噪聲，每次生成的結果都會因過程中的隨機項而略有不同，缺乏可重現性。

DDIM 的反向推理步驟如下：

$$x_{t-1} = \sqrt{\overline{\alpha_{t-1}}}\,\widehat{x_0} + \sqrt{1 - \overline{\alpha_{t-1}} - \sigma_t^2} \cdot \epsilon_t + \sigma_t z$$

其中：

$\widehat{x_0}$：模型預測的最終乾淨圖像（根據 $x_t$）

$\epsilon_t$：模型預測的噪聲方向

$z \sim \mathcal{N}(0, I)$：隨機噪聲

$\overline{\alpha_t}$：累積噪聲消除係數（由 scheduler 決定）

$\sigma_t$：控制隨機性的權重，與 DDIM 的$\eta$參數相關

此公式可以拆解為三個方向：

1. 指向最終圖像的方向：$\sqrt{\overline{\alpha_{t-1}}}\,\widehat{x_0}$
2. 指向噪聲圖像的方向：$\sqrt{1 - \overline{\alpha_{t-1}} - \sigma_t^2} \cdot \epsilon_t$
3. 可選的隨機項（當 $\eta > 0$ 時）：$\sigma_t z$

DDIM 引入一個超參數 $\eta$，控制 $\sigma_t$的大小，調整「生成過程的隨機性」：

- 當 $\eta = 0$：$\sigma_t = 0$，生成變為 deterministic，用於快速推理與精準重現。
- 當 $\eta = 1$：$\sigma_t$與 DDPM 相同，生成含隨機性，用於高品質隨機生成。

```python
# [修改] 使用 DDIMScheduler
noise_scheduler = DDIMScheduler(
    num_train_timesteps=args.num_train_timesteps,
    beta_schedule=args.beta_schedule,
    clip_sample=False,  # DDIM 通常建議不裁切樣本
)
```

程式碼 10. DDIM scheduler 程式

```python
# 設定推論步數，此處預覽與訓練步數一致
noise_scheduler.set_timesteps(args.num_train_timesteps)

for t in tqdm(noise_scheduler.timesteps, desc="Generating preview samples"):
    t_for_model = t.to(device).expand(preview_images.shape[0])
    noise_pred = model(preview_images, t_for_model, preview_labels)
    # 使用 DDIM 的 step 函數
    preview_images = noise_scheduler.step(noise_pred, t, preview_images).prev_sample

accuracy = evaluator.eval(preview_images, preview_labels)
gen_images_display = (preview_images.clamp(-1, 1) + 1) / 2
save_path = os.path.join(sample_dir, f"epoch_{epoch + 1}.png")
save_image(make_grid(gen_images_display), save_path)
```

程式碼 11. DDIM 透過 noise_pred 對每個時間步進行確定性更新

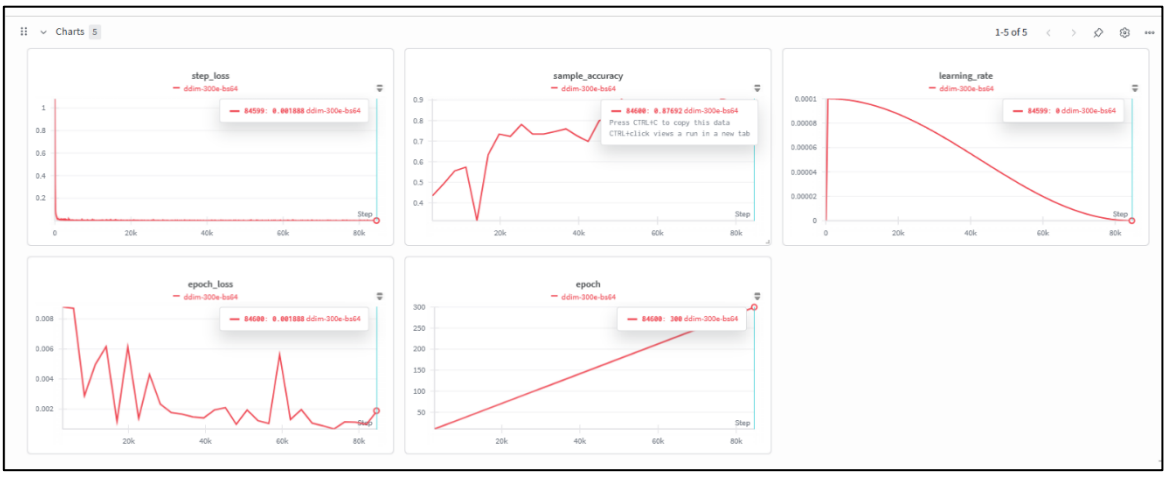



圖 4. DDIM 數據結果

# 4. 參考資料

[1] J. Yin, “NYCU Deep Learning Course Code,” *GitHub Repository*. [Online]. Available: https://github.com/jayin92/NYCU-deep-learning

[2] alu98753, “NYCU Deep Learning 2025,” *GitHub Repository*. [Online]. Available: https://github.com/alu98753/NYCU-Deep-Learning-2025

[3] Part-time-Ray, “DLP: Deep Learning Practice,” *GitHub Repository*. [Online]. Available: https://github.com/Part-time-Ray/DLP

[4] c1uc, “2025 Spring Deep Learning Labs,” *GitHub Repository*. [Online]. Available: https://github.com/c1uc/2025_Spring_Deep-Learning-Labs

[5] W. Shzd, “DDPM 擴散模型詳解,” *CSDN Blog*, Aug. 2023. [Online]. Available: https://blog.csdn.net/wshzd/article/details/132500829

[6] QQ_38423499, “擴散模型(DDPM)數學推導,” *CSDN Blog*, Apr. 2024. [Online]. Available: https://blog.csdn.net/qq_38423499/article/details/136845413

[7] HuggingFace, “diffusers,” *GitHub Repository*. [Online]. Available: https://github.com/huggingface/diffusers

[8] Zhihu 專欄, “DDPM 詳解,” *Zhihu*, 2024. [Online]. Available: https://zhuanlan.zhihu.com/p/679792481

[9] Weixin_44212848, “DDPM 程式實現與解析,” *CSDN Blog*, 2024. [Online]. Available: https://blog.csdn.net/weixin_44212848/article/details/141128157

[10] Jonathan Ho & Tim Salimans, CLASSIFIER-FREE DIFFUSION GUIDANCE

Available：https://arxiv.org/pdf/2207.12598

## 使用的 AI 工具：

✧ **ChatGPT-4o**（由 OpenAI 提供）
用於問答、報告草稿撰寫、英文轉中文及論文說明。

✧ **Google AI Studio**
作為補充問答工具，用於模型概念、架構或函式撰寫思路輔助。

✧ **VSCode + Google Gemini 2.5 Pro**
協助撰寫與補全 Python 程式碼，並支援簡易 debug。

✧ **VSCode GitHub Copilot（GPT-4o）**
用於實作過程中進行即時程式補全、語法建議與除錯輔助。