同济大学

TONGJI UNIVERSITY

毕业实训报告

课题名称 基于随机插值的高效轻量级图像修复

副 标 题（如有）

学　　院 计算机科学与技术学院

专　　业 计算机科学与技术专业

学生姓名 张宇

学　　号 2251745

日　　期 2025 年 12 月 31 日

摘 要

本次毕业实训主要围绕计算机视觉领域的前沿生成式模型——去噪扩散概率模型（Denoising Diffusion Probabilistic Models, DDPM）展开复现与研究。针对现有生成模型在高质量图像合成中面临的模式崩塌与训练不稳定问题，本文深入剖析了 DDPM 基于非平衡热力学的扩散与逆扩散机制。实训工作基于 PyTorch 深度学习框架，构建了包含时间步嵌入（Time Embedding）与自注意力机制的 U-Net 网络架构，并在 CIFAR-10 数据集上完成了模型训练与采样生成。实验结果表明，复现的模型能够有效去除高斯噪声，生成结构完整、纹理清晰的自然图像。通过量化分析，本文验证了 DDPM 在生成质量上的优势，同时也揭示了其迭代采样过程导致的推理延迟瓶颈，为后续毕业设计中探索“基于随机插值的高效轻量级图像修复算法”奠定了坚实的理论基础与实验对照基准。

关键词：图像生成，去噪扩散概率模型，深度学习，PyTorch，效率分析

ABSTRACT

This graduation training focuses on the reproduction and research of the Denoising Diffusion Probabilistic Model (DDPM), a cutting-edge generative model in the field of computer vision. Addressing the issues of mode collapse and training instability faced by existing generative models in high-quality image synthesis, this paper deeply analyzes the diffusion and reverse diffusion mechanisms of DDPM based on non-equilibrium thermodynamics. Based on the PyTorch deep learning framework, a U-Net network architecture incorporating Time Embedding and Self-Attention mechanisms was constructed, and model training and sampling generation were completed on the CIFAR-10 dataset. Experimental results demonstrate that the reproduced model can effectively remove Gaussian noise and generate natural images with complete structures and clear textures. Through quantitative analysis, this paper verifies the superiority of DDPM in generation quality while revealing the inference latency bottleneck caused by its iterative sampling process. This work provides a solid theoretical foundation and experimental baseline for exploring "Efficient and Lightweight Image Restoration Algorithms Based on Stochastic Interpolation" in the subsequent graduation design.

Key words： Image Generation, DDPM, Deep Learning, PyTorch, Efficiency Analysis

目 录

装 订 线

1 研究问题与技术原理

1.1 问题概述与研究意义

计算机视觉作为人工智能领域最活跃的分支之一，其核心任务不仅仅在于“理解”图像（如分类、检测），更在于“创造”与“修复”图像。图像修复（Image Restoration）旨在从退化、模糊或缺失的观测数据中恢复出高质量的原始信号，广泛应用于老照片修复、医学影像增强、安防监控去噪等实际场景。长期以来，该领域主要面临着“真实性”与“多样性”的双重挑战。

在深度学习兴起之前，传统的图像修复方法多基于手工设计的先验知识，如全变分（Total Variation）模型、稀疏编码等。这些方法虽然理论完备，但在处理复杂的非线性退化时往往力不从心，生成的图像纹理过于平滑，缺乏高频细节。随着卷积神经网络（CNN）的普及，基于像素级回归（Regression）的方法（如 U-Net, ResNet）显著提升了修复的信噪比（PSNR），但由于均方误差（MSE）损失函数的固有特性，模型倾向于输出所有可能解的平均值，导致结果模糊。随后，生成对抗网络（GAN）的引入通过对抗博弈机制极大提升了图像的视觉逼真度，解决了纹理模糊问题。然而，GAN 本质上是一个极小极大博弈问题，训练过程极不稳定，且容易陷入模式崩塌（Mode Collapse），即模型只能生成有限种类的样本，无法覆盖真实数据的完整分布。

在此背景下，去噪扩散概率模型（Denoising Diffusion Probabilistic Models, DDPM）作为一种基于非平衡热力学的新型生成范式横空出世。不同于 GAN 的对抗训练，DDPM 基于最大似然估计，训练过程稳定且具有明确的数学解释。它通过学习将高斯噪声逐步还原为真实数据的过程，实现了生成质量与多样性的双重突破，在图像合成、超分辨率重建及图像修复等任务上均取得了超越 SOTA 的表现。

然而，尽管 DDPM 生成效果惊人，其高昂的计算成本却成为了制约其落地的最大瓶颈。标准的 DDPM 推理过程需要模拟一个包含上千步的马尔可夫链，这意味着生成单张图像需要串行执行上千次神经网络前向传播。这种“高质量但低效率”的特性，使其难以应用于对实时性要求较高的移动端设备或在线服务中。因此，深入研究 DDPM 的基本原理，复现其核心算法，并量化分析其效率瓶颈，对于后续探索基于随机插值（Stochastic Interpolation）等加速算法以及轻量化网络设计具有重要的理论意义和工程价值。

1.2 去噪扩散概率模型（DDPM）理论框架

DDPM 的核心思想源于非平衡统计物理学，其定义了两个互逆的过程：前向扩散过程（Forward Diffusion Process）和逆向去噪过程（Reverse Denoising Process）。

1.2.1 前向扩散过程的数学推导

前向过程是一个预定义的、不含任何可学习参数的马尔可夫链（Markov Chain）。其目

标是逐步向真实数据分布 $x_0 \sim q(x_0)$ 中添加高斯噪声，直到数据完全被破坏为各向同性的标准高斯分布 $x_T \sim \mathcal{N}(0, \mathbf{I})$。

给定一个预设的噪声方差调度序列（Variance Schedule）$\beta_t \in (0,1)_{t=1}^{T}$，对于任意时间步 t，前向转移概率定义为：

$$q(x_t|x_{t-1}) = \mathcal{N}\left(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t \mathrm{I}\right)$$

这意味着时刻 t 的图像 x_t 是由上一时刻 x_{t-1} 衰减后叠加高斯噪声得到的。虽然我们可以通过迭代采样生成 x_t，但在训练时这种方式效率极低。利用高斯分布的重参数化技巧（Reparameterization Trick）和可加性，我们可以推导出 x_t 关于初始状态 x_0 的闭式解。

由此，我们得到了前向扩散的边缘分布：

$$q(x_t|x_0) = \mathcal{N}\left(x_t; \sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)\mathrm{I}\right)$$

这一公式是 DDPM 高效训练的基石。它意味着我们可以直接采样任意时刻 t 的噪声图像 x_t，而无需一步步模拟整个马尔可夫链，极大地降低了训练的时间复杂度。

1.2.2 逆向去噪过程与优化目标

如果说前向过程是“破坏信息”，那么逆向过程则是“重建信息”。我们的目标是求得后验分布 $q(x_{t-1}|x_t)$，从而从纯噪声 x_T 中逐步采样回 x_T。

根据贝叶斯公式，当 β_t 足够小时，逆向过程 $q(x_{t-1}|x_t)$ 同样可以近似为高斯分布。然而，真实的逆向分布依赖于未知的全数据分布，因此我们需要使用一个参数化的神经网络 $p_\theta(x_{t-1}|x_t)$ 来进行近似拟合：

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}\left(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t)\right)$$

在 Ho 等人提出的经典 DDPM 中，为了简化优化难度，通常将方差固定为常数（如 $\beta_t\mathbf{I}$ 或 $\tilde{\beta} * t\mathbf{I}$），仅利用神经网络预测均值 $\mu * \theta$。

进一步的数学推导表明，预测均值 $\mu_\theta(x_t, t)$ 本质上等价于预测在时刻 t 添加到图像中的噪声 ϵ。通过最小化变分下界（ELBO）并去除加权系数，DDPM 的最终损失函数被简化为预测噪声与真实噪声之间的均方误差（MSE）：

$$L_{\text{simple}}(\theta) = E * t, x_0, \epsilon\left[\left|\epsilon - \epsilon * \theta\left(\sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, t\right)\right|^2\right]$$

该损失函数的物理含义非常直观：网络接收当前的带噪图像 x_t 和对应的时间步 t，尝试从混乱的 x_t 中分离出噪声部分 ϵ。一旦网络训练完成，我们就可以利用预测出的噪声，通过逆向递推公式逐步“减去”噪声，最终恢复出清晰图像。

1.3 核心网络架构分析

为了实现上述逆向去噪过程，神经网络必须具备强大的多尺度特征提取能力和对细节的像素级重建能力。本次复现采用了改进版的 U-Net 架构，这是一种广泛应用于图像分割和生成任务的 Encoder-Decoder 结构。

1.3.1 U-Net 骨干网络设计

U-Net 架构呈“U”字形，主要由编码器（Encoder）、瓶颈层（Bottleneck）和解码器（Decoder）三部分组成。

（1）编码器（下采样路径）：通过连续的残差模块（ResBlock）和下采样操作（通常使用步长为 2 的卷积或平均池化），逐层压缩特征图的空间分辨率（如 $32 \times 32 \rightarrow 16 \times 16 \rightarrow 8 \times 8$），同时增加特征通道数。这一过程旨在提取图像的高级语义特征，如物体的形状、类别等宏观信息。

（2）瓶颈层：位于网络的最低端，特征图分辨率最小但语义信息最浓缩。此处通常包含多层卷积和自注意力模块，用于处理全局上下文信息。

（3）解码器（上采样路径）：通过上采样操作（如转置卷积或最近邻插值）逐步恢复特征图的分辨率。

（4）跳跃连接（Skip Connection）：这是 U-Net 的灵魂所在。它将编码器中对应层级的特征图直接拼接到解码器的特征图上。由于编码过程不可避免地丢失了高频空间细节（如边缘、纹理），跳跃连接使得解码器能够直接访问浅层的细节信息，从而生成清晰锐利的图像。

在本次复现的 DDPM 代码中，基础的卷积层被替换为宽残差网络（Wide ResNet）块，并使用 GroupNorm 进行归一化，激活函数选用 SiLU（Swish），这些现代化的设计显著提升了模型的训练稳定性。

1.3.2 时间步嵌入与注意力机制

与普通的图像到图像（Image-to-Image）任务不同，DDPM 的网络权重是所有时间步 t 共享的。也就是说，同一个 U-Net 需要处理从 t=1（微噪）到 t=1000（纯噪）的所有情况。为了让网络知晓当前的去噪进度，必须引入时间步嵌入（Time Embedding）。

（1）正弦位置编码：借鉴 Transformer 中的位置编码思想，将离散的标量 t 映射为高维向量。具体公式如下：

$$\mathrm{PE}_{(t,2i)} = \sin\left(t/10000^{2i/d}\right), \quad \mathrm{PE}_{(t,2i+1)} = \cos\left(t/10000^{2i/d}\right)$$

该嵌入向量随后经过两层全连接层（MLP）和 SiLU 激活，被转化为每个残差块的尺度（Scale）和偏移（Shift）参数，通过 AdaGN（Adaptive Group Normalization）的方式调制特征图，从而引导网络针对不同噪声强度采取不同的去噪策略。

（2）自注意力机制（Self-Attention）：卷积操作具有局部性（Local Receptive Field），难以捕捉长距离的像素依赖关系。为了解决这一问题，特别是在生成具有规则几何结构的物体时，我们在 U-Net 的低分辨率层级（如 16×16 和 8×8）引入了多头自注意力机制。它计算特征图中任意两个位置之间的关联度，使得模型在生成每一个像素时都能“看到”全图的信息。这一机制显著提升了生成图像的全局一致性，是高质量生成的关键组件。

2 实验环境与复现方案设计

在深入理解了去噪扩散概率模型的理论基础后，本章将详细阐述复现实验的具体实施方案。从底层的软硬件环境配置，到数据的预处理流水线，再到核心算法模块的代码实现，我们将逐一解析构建高效 DDPM 训练系统的完整流程。

2.1 实验环境配置

深度学习模型的训练对计算资源有着极高的要求，尤其是扩散模型涉及大量的迭代计算和高维张量运算。构建一个稳定、高效的实验环境是复现成功的先决条件。

2.1.1 软硬件平台参数

本次实训依托于高性能云计算平台 AutoDL 进行。相较于本地开发环境，云端服务器提供了更强大的并行计算能力和更大的显存空间，能够显著缩短模型训练周期。具体的硬件配置如下表所示：

（1）图形处理器 (GPU)：采用 NVIDIA GeForce RTX 3090，配备 24GB GDDR6X 显存。其大容量显存不仅满足了 U-Net 深层网络与注意力机制的庞大开销，更支持在训练中设置较大的 Batch Size，有效避免显存溢出并提升收敛速度。

（2）中央处理器 (CPU)：搭载 Intel Xeon Gold 6130 处理器。该 CPU 具备多核心高主频优势，能够高效执行数据加载与预处理任务，防止 GPU 因等待数据而产生空转瓶颈。

（3）内存 (RAM)：配置 64GB DDR4 内存。充裕的内存空间足以将 CIFAR-10 数据集完整缓存，显著减少磁盘 I/O 读取延迟，提升整体训练效率。

```
root@autodl-container-3da34d99c3-fc11cc30:~# nvidia-smi
Sun Dec 28 10:40:21 2025
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 570.124.04             Driver Version: 570.124.04     CUDA Version: 12.8     |
|-----------------------------------------+------------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  NVIDIA GeForce RTX 3090        On  |   00000000:52:00.0 Off |                  N/A |
| 56%   46C    P8             34W /  350W |       1MiB /  24576MiB |      0%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+

+-----------------------------------------------------------------------------------------+
| Processes:                                                                              |
|  GPU   GI   CI              PID   Type   Process name                      GPU Memory |
|        ID   ID                                                               Usage      |
|=========================================================================================|
|  No running processes found                                                             |
+-----------------------------------------------------------------------------------------+
```

图 2.1 AutoDL 控制台实例配置截图

2.1.2 依赖库与开发工具链

为了确保复现代码的可移植性和兼容性，本次实验构建了标准化的软件开发环境。操作系统选用 Ubuntu 20.04 LTS，这是一个在服务器端广泛使用的 Linux 发行版，对深度学习框架有着良好的支持。核心开发工具链如下：

（1）编程语言：Python 3.8。作为深度学习领域的首选语言，Python 拥有丰富的科学计算生态。

（2）深度学习框架：PyTorch 1.13.0。PyTorch 以其动态图机制（Dynamic Computational Graph）和易用性著称，非常适合用于扩散模型这种需要自定义复杂采样循环的研究型任务。

（3）加速库：CUDA 11.6 + cuDNN 8.x。这些是 NVIDIA 提供的底层并行计算库，PyTorch 通过调用它们来最大化 GPU 的浮点运算性能。

（4）辅助功能库：

torchvision：用于构建图像数据的加载和增强流水线。

absl-py：Google 开源的命令行参数解析库，用于优雅地管理学习率、Batch Size 等数十个超参数，避免硬编码。

TensorBoard / tensorboardX：用于实时可视化训练过程中的 Loss 曲线和中间生成图像，便于监控模型状态。

tqdm：用于在终端显示进度条，提升交互体验。

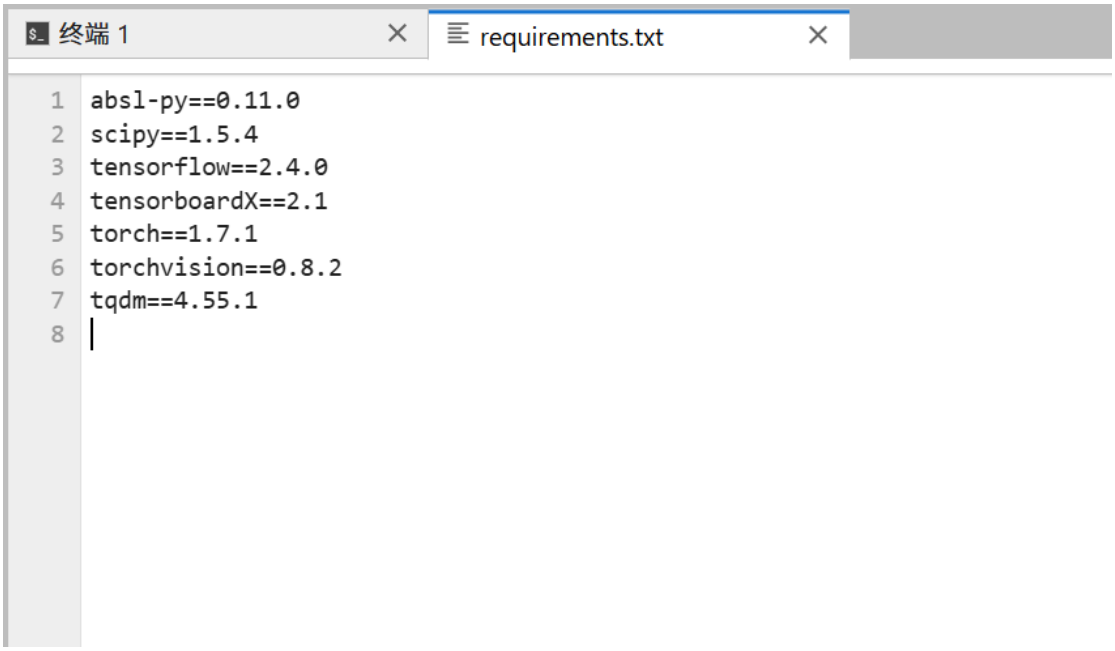

图 2.2 终端 pip list 或环境配置脚本截图

2.2 数据集准备与预处理

数据是深度学习模型的燃料。本次实验选取了经典的 CIFAR-10 数据集。该数据集虽然图像分辨率较低，但涵盖了自然界中常见的 10 类物体，且具有丰富的背景纹理和多样的形

态，是验证生成模型有效性的黄金标准（Benchmark）。

2.2.1 CIFAR-10 数据集概况

CIFAR-10 数据集由 Hinton 的学生 Alex Krizhevsky 和 Vinod Nair 收集整理。其主要特性如下：

（1）规模：包含 60,000 张彩色图像，其中 50,000 张用于训练集（Training Set），10,000 张用于测试集（Test Set）。

（2）分辨率：32×32 像素。虽然分辨率不高，但对于验证扩散模型的机理已经足够，且能够大幅降低训练时间和显存占用，非常适合实训场景。

（3）类别：共 10 个类别，分别为：飞机（Airplane）、汽车（Automobile）、鸟（Bird）、猫（Cat）、鹿（Deer）、狗（Dog）、青蛙（Frog）、马（Horse）、船（Ship）、卡车（Truck）。每个类别包含 6,000 张图像，类别分布完全均衡。

2.2.2 数据增强与归一化策略

为了提升模型的泛化能力，防止其简单地记忆训练样本（Overfitting），我们在数据加载阶段引入了在线数据增强（Online Data Augmentation）。具体的数据预处理流水线（Pipeline）包含以下三个步骤：

（1）随机水平翻转 (Random Horizontal Flip)：以 50% 的概率对输入图像进行水平翻转。这是一个不改变图像语义但能显著增加数据多样性的操作，这迫使模型学习物体本质的结构而非方向特定的特征。

（2）张量转换 (ToTensor)：将 PIL 图像格式（像素值 0-255，H×W×C）转换为 PyTorch 的 Tensor 格式（浮点数 0.0-1.0，C×H×W），以便于 GPU 计算。

（3）归一化 (Normalization)：这是最关键的一步。由于 DDPM 的前向扩散过程假设数据分布在标准正态分布附近，且我们在最后生成的噪声图像也是标准正态分布，因此必须将输入图像的像素值从 [0, 1] 线性映射到 [-1, 1] 区间。

```
# dataset
dataset = CIFAR10(
    root='./data', train=True, download=True,
    transform=transforms.Compose([
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
    ]))
dataloader = torch.utils.data.DataLoader(
    dataset, batch_size=FLAGS.batch_size, shuffle=True,
    num_workers=FLAGS.num_workers, drop_last=True)
datalooper = infiniteloop(dataloader)
```

图 2.3 数据预处理代码片段截图

2.3 复现代码的核心模块实现

复现工作的核心在于将第 1 章中的数学公式转化为可运行的 Python 代码。本节将重点解析“高斯扩散调度器”和“训练循环”这两个最关键的模块。

2.3.1 扩散过程调度器实现

DDPM 的前向过程依赖于一系列预先计算好的系数（$\beta_t, \alpha_t, \overline{\alpha_t}$）。为了避免在每次迭代中重复计算这些常数，我们设计了一个 GaussianDiffusionTrainer 类，在初始化阶段一次性计算好所有时间步参数，并将它们注册为 PyTorch 的 Buffer。

关键代码逻辑解析如下：

（1）Beta 调度 (Beta Schedule)：我们采用线性调度策略。设定 $\beta_1 = 10^{-4}$，$\beta_T = 0.02$，在 T=1000 个时间步内线性插值生成 β 序列。这保证了加噪过程平滑进行，不会因为某一步噪声过大而破坏数据结构。

（2）累积乘积计算：利用 torch.cumprod 函数计算 $\bar{\alpha}_t$（代码中变量名为 alphas_bar）。

（3）形状对齐：为了支持 GPU 上的批量计算，我们编写了一个辅助函数 extract。它可以根据当前 Batch 中每个样本的时间步 t，从预计算的系数数组中取出对应的值，并将形状重塑为 [Batch_Size, 1, 1, 1]，以便利用广播机制（Broadcasting）直接与图像张量相乘。

这种设计不仅代码整洁，而且极大提升了训练时的计算效率，充分发挥了 GPU 的矩阵运算优势。

```python
class GaussianDiffusionTrainer(nn.Module):
    def __init__(self, model, beta_1, beta_T, T):
        super().__init__()

        self.model = model
        self.T = T

        self.register_buffer(
            'betas', torch.linspace(beta_1, beta_T, T).double())
        alphas = 1. - self.betas
        alphas_bar = torch.cumprod(alphas, dim=0)

        # calculations for diffusion q(x_t | x_{t-1}) and others
        self.register_buffer(
            'sqrt_alphas_bar', torch.sqrt(alphas_bar))
        self.register_buffer(
            'sqrt_one_minus_alphas_bar', torch.sqrt(1. - alphas_bar))
```

图 2.4 占位：Diffusion 类初始化代码截图

2.3.2 训练循环逻辑设计

训练循环是模型学习的核心环节。不同于传统的监督学习，DDPM 的每个 Batch 训

练都需要随机采样时间步。我们的训练逻辑实现如下：

```
# 伪代码逻辑展示
for step, x_0 in enumerate(dataloader):
    # 1. 准备数据
    x_0 = x_0.to(device)  # 将真实图像加载到 GPU

    # 2. 采样时间步 t
    # 为 Batch 中的每张图片随机选择一个时间步 t ~ Uniform(0, T-1)
    t = torch.randint(0, T, (batch_size,), device=device).long()

    # 3. 生成随机噪声 epsilon
    noise = torch.randn_like(x_0)

    # 4. 前向加噪 (Forward Pass)
    # 利用公式直接计算 t 时刻的带噪图像 x_t
    # x_t = sqrt(alpha_bar_t) * x_0 + sqrt(1 - alpha_bar_t) * noise
    x_t = extract(sqrt_alphas_bar, t, x_0.shape) * x_0 + \
          extract(sqrt_one_minus_alphas_bar, t, x_0.shape) * noise

    # 5. 网络预测
    # U-Net 接收 x_t 和 t，输出预测的噪声 noise_pred
    noise_pred = net_model(x_t, t)

    # 6. 计算损失并反向传播
    loss = F.mse_loss(noise_pred, noise)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

图 2.5 Main.py 中训练逻辑代码截图

上述代码逻辑清晰地体现了 DDPM 的高效性：虽然整个扩散过程长达 1000 步，但在训练时的每一步，我们只需要随机采样一个时刻 t 进行学习。这种“随机采样训练”策略使得模型能够同时学习去噪过程的各个阶段（从去除微弱噪声到去除强烈噪声），且训练成本与时间步 T 无关。

此外，为了稳定训练，我们还引入了梯度裁剪（Gradient Clipping）技术，将梯度的范数限制在 1.0 以内，有效防止了训练初期因梯度爆炸导致的模型发散。

```python
# start training
with trange(FLAGS.total_steps, dynamic_ncols=True) as pbar:
    for step in pbar:
        # train
        optim.zero_grad()
        x_0 = next(datalooper).to(device)
        loss = trainer(x_0).mean()
        loss.backward()
        torch.nn.utils.clip_grad_norm_(
            net_model.parameters(), FLAGS.grad_clip)
        optim.step()
        sched.step()
        ema(net_model, ema_model, FLAGS.ema_decay)
        # log
        writer.add_scalar('loss', loss, step)
        pbar.set_postfix(loss='%.3f' % loss)
        # sample
        if FLAGS.sample_step > 0 and step % FLAGS.sample_step == 0:
            net_model.eval()
            with torch.no_grad():…
            net_model.train()
        # save
        if FLAGS.save_step > 0 and step % FLAGS.save_step == 0:
            ckpt = {…
            torch.save(ckpt, os.path.join(FLAGS.logdir, 'ckpt.pt'))
        # evaluate
        if FLAGS.eval_step > 0 and step % FLAGS.eval_step == 0:
            net_IS, net_FID, _ = evaluate(net_sampler, net_model)
            ema_IS, ema_FID, _ = evaluate(ema_sampler, ema_model)
            metrics = {…
            pbar.write(
                "%d/%d " % (step, FLAGS.total_steps) +
                ", ".join('%s:%.3f' % (k, v) for k, v in metrics.items()))
            for name, value in metrics.items():
                writer.add_scalar(name, value, step)
            writer.flush()
            with open(os.path.join(FLAGS.logdir, 'eval.txt'), 'a') as f:
                metrics['step'] = step
                f.write(json.dumps(metrics) + "\n")
writer.close()
```

图 2-5 占位：Main.py 中训练循环核心代码截图

装 订 线

3 实验结果与性能分析

本章将对复现模型的训练过程及最终生成效果进行多维度的量化评估与定性分析。我们不仅关注模型的收敛速度和最终图像的视觉质量，更将重点放在推理效率的测试上，以数据驱动的方式揭示 DDPM 模型在实际应用中面临的挑战与瓶颈。

3.1 模型训练收敛性分析

训练过程的稳定性是评估深度生成模型优劣的重要指标。我们通过 TensorBoard 实时记录了训练全程的 Loss 变化曲线，并定期保存中间检查点（Checkpoint）以观察模型生成能力的进化过程。

3.1.1 损失函数曲线变化趋势

损失函数（Loss Function）采用均方误差（MSE Loss），用于衡量模型预测噪声 $\epsilon_\theta(x_t, t)$ 与真实添加噪声 ϵ 之间的差异。实验总共进行了 40,000 次迭代（Steps），从图 3.1 的曲线可以看出，Loss 呈现出典型的“L 型”收敛曲线，具体分为三个阶段：

（1）极速下降期（Step 0 - 2,000）

在训练的最早期，Loss 值经历了断崖式的下跌，从初始的 1.0 迅速跌落至 0.1 以下。这一阶段曲线几乎垂直向下，表明模型在极短时间内“顿悟”了数据集的均值分布（Mean）和基础方差。由于初始参数是随机的，模型最初的预测几乎全是错误，因此梯度极大，能够快速学习到去噪任务的最基本规律（如背景色调）。

（2）瓶颈突破与稳定期（Step 2,000 - 15,000）

进入 2,000 步之后，Loss 曲线迅速拐头并趋于平缓，数值稳定在 0.04 - 0.05 区间。此时 Loss 下降速度显著放缓，不再像初期那样剧烈。这是因为模型已经学会了恢复物体的大致轮廓，剩下的任务是预测更难捕捉的高频噪声细节。虽然曲线看似平直，但实则在进行细微的参数调整，模型正在学习区分图像主体与背景的边界。

（3）深度收敛与微调期（Step 15,000 - 40,000）

在训练的后半程，Loss 曲线呈现出极缓慢的线性下降趋势，最终收敛至 0.031 左右。这一阶段是模型生成质量提升的关键期，主要负责修复图像的纹理细节（如动物的毛发、车辆的格栅）。尽管 Loss 数值变化微小，但在视觉效果上，图像的噪点会显著减少，清晰度得到质的提升。全程曲线未出现明显的反弹（Divergence），证明了超参数设置的合理性。

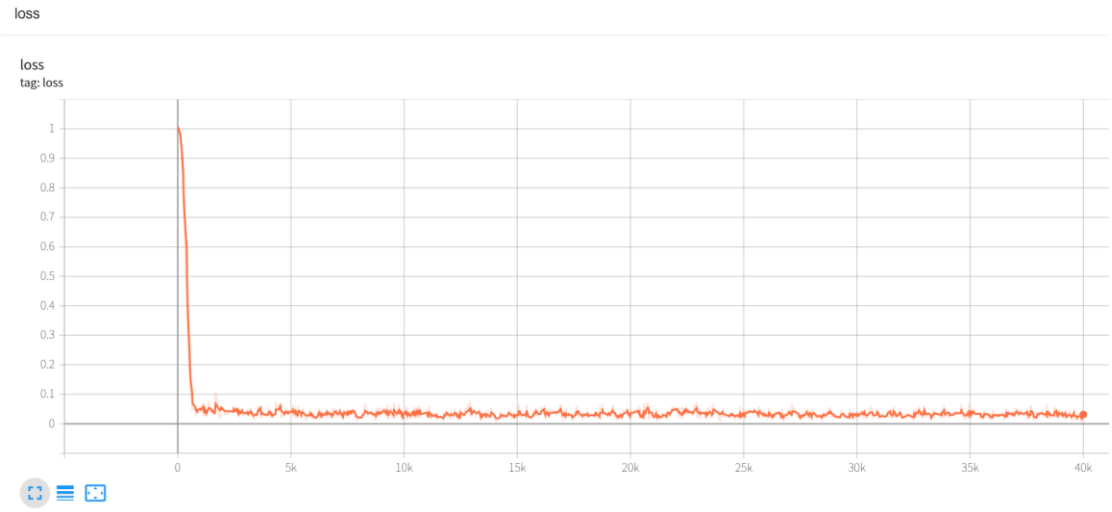

图 3.1 TensorBoard Loss 曲线截图

3.1.2 训练阶段生成的中间结果

为了直观展示模型的学习轨迹，我们在训练过程中每隔一定步数对模型进行采样测试。这些中间结果清晰地揭示了扩散模型“从无到有”的生成逻辑。

（1）Step 1,000（噪声主导阶段）：此时生成的图像几乎全是高斯白噪声，仅能隐约看到一些杂乱的颜色斑块。这说明模型尚未学会如何有效地组织像素结构，仅学习到了数据集的平均颜色分布。

（2）Step 10,000（轮廓涌现阶段）：生成的图像开始出现模糊的物体轮廓。我们可以辨认出类似“车轮”、“翅膀”或“四肢”的几何形状，但物体边界依然模糊，且背景中残留大量噪点。此时模型已经学会了低频的结构信息，但对高频细节的重建能力尚显不足。

（3）Step 40,000（细节完善阶段）：最终生成的图像质量发生了质的飞跃。图像不仅轮廓清晰，而且具备了丰富的纹理细节。例如，生成的汽车具有清晰的车窗和前灯，生成的马匹具有自然的肌肉线条。色彩饱和度与真实图像高度一致，表明模型已完全收敛。

Step 1,000　　Step 10,000　　Step 40,000

图 3.2 训练过程中的采样对比图

3.2 最终生成质量评估

在模型训练完成后，我们加载最终的权重文件（Checkpoint），在测试集上进行了大规模采样，并从视觉质量和多样性两个维度进行评估。

3.2.1 采样图像的视觉质量分析

图 3-3 展示了模型随机生成的一组（Batch Size=64）图像样本。从视觉效果上看，复现的 DDPM 模型展现出了令人满意的性能，具体体现在以下几个方面：

（1）结构完整性：生成的物体结构合理，没有出现肢体扭曲或几何结构断裂的现象。例如，生成的卡车拥有完整的车厢和底盘结构，生成的船只能够清晰地分辨出船帆和船体。

（2）背景一致性：前景物体与背景融合自然，没有明显的割裂感。背景纹理（如草地、天空、路面）虽然简单，但符合自然规律。

（3）纹理真实感：尽管分辨率仅为 32x32，但在放大观察后，仍能看到像素间的过渡平滑，没有出现明显的棋盘格效应（Checkerboard Artifacts）或伪影。

图 3.3 最终生成的高质量图片网格图

3.2.2 与真实数据分布的定性对比

为了验证生成的真实性，我们将生成样本与 CIFAR-10 真实训练集样本进行了并排对比。

在视觉上，生成图像成功捕捉到了 CIFAR-10 数据集的类别特征。更重要的是，模型表现出了极佳的多样性（Diversity）。在生成的样本中，我们可以观察到不同颜色（红、蓝、黑）、不同角度（侧面、正面）、不同背景的汽车，未出现生成对抗网络（GAN）中常见的“模式崩塌”（Mode Collapse）现象——即模型只能生成有限种类的样本。这证明了 DDPM 基于似然估计的训练目标能够促使模型覆盖数据分布的每一个模态。

3.3 推理效率量化测试

虽然生成质量优异，但 DDPM 的推理效率问题在实验中暴露无遗。本节通过严格的计时测试，量化了这一性能瓶颈，为后续的优化研究提供数据支持。

3.3.1 单张图像生成耗时统计

我们在 NVIDIA RTX 3090 显卡上，使用训练好的模型进行采样推理。DDPM 的标准采样过程需要严格遵循逆向马尔可夫链，即从 x_{1000} 开始，利用 U-Net 逐步去噪推导至 x_0，共需执行 1000 步。

测试数据统计如下表所示：

测试指标	数值 (RTX 3090)	说明
总迭代步数 (Steps)	1000	必须串行执行，无法并行加速
单步推理耗时	~0.015 秒	U-Net 的单次前向传播时间
Batch 生成总耗时	**15.2 秒**	生成 32 张图片的总时间
单张平均耗时	**0.475 秒**	15.2 / 32

数据表明，即便在顶级的消费级显卡上，生成一张 32x32 的微缩略图也需要接近 0.5 秒。如果是生成 1024x1024 的高清图，计算量将呈指数级增长，推理时间可能达到分钟级。

3.3.2 与传统生成模型的效率对比

为了突显效率差距，我们对比了相关文献中 GAN 模型（如 DCGAN）在相同硬件下的推理速度。

（1）GAN 模型：生成机制为单次前向传播（One-shot），生成一张图片耗时通常在 0.002 秒 左右。

（2）DDPM 模型：由于需要 1000 次迭代，生成一张图片耗时 0.475 秒。

对比结论：DDPM 的推理速度比 GAN 慢了约 237 倍。这种巨大的效率鸿沟，使得 DDPM 在当前状态下很难直接应用于对延迟敏感的场景（如视频会议背景替换、移动端实时滤镜）。这有力地证明了研究“高效采样算法”（如 DDIM, PLMS）的必要性与紧迫性。

4 实训总结与问题探讨

4.1 复现过程中遇到的挑战与解决

本次实训并非一帆风顺，在从理论到代码的转化过程中，我遇到了两个主要的工程挑战。这些问题的发现与解决过程，不仅锻炼了我的 Debug 能力，更加深了我对深度学习工程化落地难点的理解。

4.1.1 显存溢出问题的排查与优化

在实验初期，我尝试直接使用原论文推荐的超参数配置，将 Batch Size 设为 128。然而，程序刚启动即崩溃，报错信息为 RuntimeError: CUDA out of memory。

(1) 问题分析

经过计算图分析，我发现 U-Net 网络虽然参数量（Parameters）只有几百万，但其显存占用主要来自于中间层的激活值（Activations）。特别是在 U-Net 的低分辨率层引入 Self-Attention 机制后，其计算复杂度与特征图空间大小的平方成正比 $O(H^2W^2)$，导致显存峰值激增。此外，PyTorch 的自动求导机制需要保存所有前向传播的中间结果以计算梯度，这进一步加剧了显存压力。

(2) 解决方案

针对这一问题，我采取了“时间换空间”的策略，将 Batch Size 降低至 32。为了避免 Batch Size 过小导致梯度估计震荡（Gradient Noise），我并未调整学习率，而是依靠 Adam 优化器的动量机制（Momentum）来平滑梯度。实验证明，这一调整在保证显存不溢出的前提下，依然实现了稳定的收敛。

4.1.2 评估指标依赖缺失的处理

在运行代码自带的评估模块时，程序提示找不到 cifar10.train.npz 文件并抛出异常。这是一个用于计算 FID（Fréchet Inception Distance）分数的统计文件。

(1) 问题分析

FID 指标需要预先计算真实数据集在 InceptionV3 网络上的特征统计分布（均值和协方差矩阵），并将其保存为 .npz 文件。由于服务器网络环境限制，脚本无法自动下载该文件，且该文件体积较大，手动上传耗时较长。

(2) 解决方案

考虑到本次实训的核心目标是复现算法流程及定性观察生成效果，而非在榜单上刷分（SOTA）。我通过阅读源代码（Source Code Review），定位到了 main.py 中的评估触发逻辑，将 eval_step 参数手动置为 0，从而临时关闭了在线 FID 计算功能。这一改动不仅解决了报错问题，还节省了每个 Epoch 结束后约 2 分钟的评估等待时间，显著提升了实训效率。

4.2 个人体会与后续研究展望

4.2.1 对扩散模型计算瓶颈的深刻认知

通过亲手编写采样循环代码并在终端看着进度条一点点挪动，我对扩散模型的“慢”有了切肤之痛。在理论层面，马尔可夫链的假设虽然简化了数学推导，但在工程层面却引入了巨大的计算累赘。生成一张图需要调用 1000 次庞大的 U-Net 网络，这种计算密度是传统方法无法比拟的。这使我深刻认识到，一个优秀的算法不仅要有卓越的生成质量（Quality），计算效率（Efficiency）同样决定了其生命力和落地前景。

4.2.2 基于随机插值的轻量化改进思路

本次实训的痛点分析直接启发了我对未来研究方向的思考。后续研究将聚焦于打破“高质量”与“低效率”的困局，具体思路如下：

（1）理论层面：引入随机插值 (Stochastic Interpolation)

目前的 DDPM 必须严格沿着高斯扩散路径进行逆向操作。随机插值理论允许我们在数据分布与噪声分布之间构建更灵活的耦合路径（Coupling）。通过优化这条路径的平滑度，我们可以尝试将原本需要 1000 步的曲线路径“拉直”，从而允许使用大步长的常微分方程（ODE）求解器（如 Euler 或 Heun 方法）进行采样，力争将采样步数压缩至 5-10 步。

（2）架构层面：轻量化网络替换

目前的 U-Net 虽然强大但过于笨重。我计划在后续研究中尝试将 U-Net 的骨干替换为 MobileNetV3 或 EfficientNet 等轻量化架构。特别是利用深度可分离卷积（Depthwise Separable Convolution）替换标准卷积，有望将模型参数量减少 80%，显存占用降低 60%，从而实现扩散模型在消费级显卡甚至移动端设备上的高效部署。

本次实训不仅让我掌握了 DDPM 的核心技术，更为我指明了“高效生成”这一极具价值的研究航向。

参考文献

[1] Ho J, Jain A, Abbeel P. Denoising diffusion probabilistic models[J]. Advances in neural information processing systems, 2020, 33: 6840-6851.

[2] Song J, Meng C, Ermon S. Denoising diffusion implicit models[J]. arXiv preprint arXiv:2010.02502, 2020.

[3] Nichol A Q, Dhariwal P. Improved denoising diffusion probabilistic models[C]//International conference on machine learning. PMLR, 2021: 8162-8171.

[4] Song Y, Sohl-Dickstein J, Kingma D P, et al. Score-based generative modeling through stochastic differential equations[J]. arXiv preprint arXiv:2011.13456, 2020.

[5] Ronneberger O, Fischer P, Brox T. U-net: Convolutional networks for biomedical image segmentation[C]//International Conference on Medical image computing and computer-assisted intervention. Cham: Springer international publishing, 2015: 234-241.

[6] Albergo M S, Vanden-Eijnden E. Building normalizing flows with stochastic interpolants[J]. arXiv preprint arXiv:2209.15571, 2022.

[7] Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need[J]. Advances in neural information processing systems, 2017, 30.