# GAN

## 基本原理

GAN由两个神经网络组成——生成器（Generator）和判别器（Discriminator），通过博弈的方式进行训练。生成器试图“欺骗”判别器，生成尽可能逼真的数据；而判别器则试图区分生成的数据与真实数据。这种对抗训练促进生成器不断提高生成质量，现在常见为某模型组件，提高生成质量

## 优点

生成保真性强，对细节勾画到位，灵活，可无监督

## 缺点

容易崩塌，且难以平衡G、D之间训练关系，且难以具体评估

## 伪代码

```
for epoch in range(num_epochs):
    for real_data in dataloader:
        # === 训练判别器 D ===
        z = sample_noise(batch_size)
        fake_data = G(z).detach()  # 生成假样本（不反传到G）
        D_loss = -log(D(real_data)) - log(1 - D(fake_data))
        D_optimizer.zero_grad()
        D_loss.backward()
        D_optimizer.step()

        # === 训练生成器 G ===
        z = sample_noise(batch_size)
        fake_data = G(z)
        G_loss = -log(D(fake_data))  # 生成器希望 D(fake) 越接近1越好
        G_optimizer.zero_grad()
        G_loss.backward()
        G_optimizer.step()
```

# PixelCNN

## 基本原理

是一种基于自回归的图像生成模型。它通过建模图像中每个像素的条件概率来逐像素地生成图像：

$$P(\mathbf{x}) = \prod_{i=1}^{n} P(x_i \mid x_1, x_2, \ldots, x_{i-1})$$

为了保证每个像素只依赖于“上方和左侧”的像素，PixelCNN 采用了 **掩码卷积（Masked Convolution）**。具体做法是使用特制的卷积核遮蔽掉当前像素之后的信息。

## 优点

**生成质量高**
对像素间的相关性建模精细，图像局部细节表现优秀。

**训练过程稳定**
不依赖对抗训练机制，相较于 GAN 更容易训练和调参。

## 缺点

**生成效率低**
由于自回归机制，必须按顺序逐像素生成，导致生成速度慢。

## 伪代码

```
# 逐像素生成图像（掩码卷积保证依赖左上像素）
for i in range(height):
    for j in range(width):
        # 生成第(i,j)个像素，只依赖之前生成的像素（上方和左侧）
        x[i][j] = model.predict(x[:i, :], x[i, :j])
```

# VAE（变分自编码器）

## 基本原理

变分自编码器（Variational Autoencoder, VAE）是一种生成模型，通过**编码器-解码器结构**实现数据的生成与重构。
它引入了潜在变量 $\mathbf{z}$，通过学习潜在空间的分布来生成数据。模型训练时，采用 **变分推断** 优化证据下界（ELBO），结合重构误差和潜在空间的分布正则化

其中，KL散度（Kullback-Leibler Divergence）用于衡量编码器学习的潜在分布 $q_\phi(\mathbf{z}|\mathbf{x})$ 与先验分布 $p(\mathbf{z})$ 之间的差异。

数学表达为：

$$\log p(\mathbf{x}) \geq \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x}|\mathbf{z})\right] - \mathrm{KL}\left(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z})\right)$$

其中：
第一项是**重构损失**，衡量生成数据与原数据相似度；
第二项是**KL散度正则化**，促进潜在变量服从先验分布（通常是标准正态分布）。

## 优点

**潜在空间结构清晰**，方便插值和数据生成；
**训练稳定**，无对抗机制；
**理论基础扎实**，结合概率图模型和深度学习。

## 缺点

生成图像质量相较GAN通常较低，图像细节不够锐利；
KL散度调节不当可能导致潜在空间“塌陷”（posterior collapse）。

## 伪代码示意

```
for batch in dataloader:
    x = batch

    # 编码器：输入x，输出潜在分布参数mu, logvar
```

```
    mu, logvar = encoder(x)

    # 采样潜变量z（重参数技巧）
    std = torch.exp(0.5 * logvar)
    eps = torch.randn_like(std)
    z = mu + eps * std

    # 解码器：重构x_hat
    x_hat = decoder(z)

    # 计算重构损失（如均方误差或交叉熵）
    recon_loss = reconstruction_loss(x_hat, x)

    # 计算KL散度
    kl_loss = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())

    # 总损失
    loss = recon_loss + kl_loss

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

# VQVAE（Vector Quantized Variational Autoencoder）

## 基本原理

VQVAE 是变分自编码器（VAE）的一个变体，主要特点是引入了**向量量化（Vector Quantization）**来处理潜在空间，从而实现离散的潜在表示。
它通过编码器将输入映射到连续潜在空间中的一个向量，然后利用**最近邻查找**将该向量映射到一个离散的代码字（codebook）中的某个条目，最后解码器根据这个离散代码字重构输入。

这样，VQVAE 将连续潜在空间转化为离散潜在空间，有利于生成高质量的离散数据表示，如图像、语音等。

## 结构特点

**编码器**：将输入映射为连续潜变量向量。
**量化层**：将连续潜变量向量映射到最近的离散代码字向量。
**代码本（Codebook）**：一个包含多个向量的字典，用于表示离散潜变量空间。
**解码器**：从离散代码字向量重构输入。

## 优点

离散潜在表示，有助于模型捕获更丰富的结构信息。
解决了传统VAE中潜在空间连续但样本离散的问题。
在图像生成和表示学习上表现优异。

## 缺点

代码本的训练需要特定的优化技巧，如 **Straight-Through Estimator**。
需要设计合理的代码本大小和平衡训练稳定性。

# 典型损失函数

VQVAE 的训练目标通常包括三部分：

- 重构损失（Reconstruction Loss）
- 代码本向量更新损失（Codebook Loss）
- 编码器承诺损失（Commitment Loss）

数学形式：

$$\mathcal{L} = \underbrace{\|x - \hat{x}\|^2}_{\text{重构损失}} + \underbrace{\|\text{sg}[z_e(x)] - e\|^2}_{\text{代码本损失}} + \underbrace{\beta\|z_e(x) - \text{sg}[e]\|^2}_{\text{承诺损失}}$$

其中：
$z_e(x)$ 是编码器输出的连续向量，
$e$ 是被量化的代码字向量，
$\text{sg}[\cdot]$ 表示**停止梯度**操作，
$\beta$ 是权重超参数。

# 伪代码

```
for batch in dataloader:
    x = batch  # 输入数据

    # 编码器：输入x，输出连续潜变量z_e
    z_e = encoder(x)  # shape: [batch, latent_dim]

    # 向量量化：找到代码本中最近的向量e
    e = quantize(z_e, codebook)  # 量化操作，离散化潜变量

    # 解码器：根据离散代码字重构输入
    x_hat = decoder(e)

    # 计算重构损失（如均方误差）
    recon_loss = reconstruction_loss(x_hat, x)

    # 代码本损失，更新代码字向量
    codebook_loss = torch.mean((sg(z_e) - e).pow(2))

    # 承诺损失，鼓励编码器输出靠近代码字
    commitment_loss = torch.mean((z_e - sg(e)).pow(2))

    # 总损失
    loss = recon_loss + codebook_loss + beta * commitment_loss

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

# 从代码本中随机采样一个离散代码向量
sampled_codes = sample_from_codebook(codebook)

# 将采样的离散代码输入解码器，生成新样本
generated_samples = decoder(sampled_codes)
```

# VQGAN

VQVAE基础上加了GAN，采样换成Transformer

# MAE（Masked Autoencoder）

## 基本原理

MAE 是一种基于自编码器的自监督学习方法，主要用于视觉领域。其核心思想是对输入图像进行随机遮挡（masking），然后训练模型只根据未遮挡的部分重建被遮挡的部分，从而学习图像的有效表示。

具体流程：

- 将输入图像划分为多个小块（patches）；
- 随机遮挡其中大部分patch，只留下少部分作为编码器输入；
- 编码器对未遮挡的patch进行编码，得到潜在表示；
- 解码器根据编码器输出和遮挡位置信息，重建完整图像或被遮挡patch；
- 通过重构损失优化模型。

## 重要信息

图片中冗余信息特别多

## 优点

有效利用大量未标注数据进行预训练；
通过遮挡学习图像局部和全局的上下文信息；
在视觉任务（如图像分类、目标检测）中表现优异。

## 缺点

训练时需要设计合理的遮挡比例和平衡编码器、解码器能力；
重构任务设计和损失函数选择影响性能。

## 数学表达

给输入图像 $x$，划分为patches后，令 $M$ 表示遮挡掩码，编码器仅输入未遮挡部分 $x_{unmasked}$。模型目标是最小化：

$$\mathcal{L} = \|x_{masked} - \hat{x}_{masked}\|^2$$

其中，$x_{masked}$ 是被遮挡的patch，$\hat{x}_{masked}$ 是解码器重构的对应部分。

## 伪代码示意

```
for batch in dataloader:
    x = batch  # 输入图像

    # 划分patch，随机生成掩码
    patches = split_into_patches(x)
    mask = random_mask(patches, mask_ratio=0.75)
```

```
    # 只保留未遮挡的patch输入编码器
    x_unmasked = patches[~mask]

    # 编码器得到潜在表示
    latent = encoder(x_unmasked)

    # 解码器重建完整patches（包含被遮挡部分）
    x_recon = decoder(latent, mask)

    # 计算被遮挡patch的重构损失
    loss = reconstruction_loss(x_recon[mask], patches[mask])

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

# MAR（Masked Autoencoder without VQ）

## 基本概念

MAR 是一种结合 **Masked Autoencoder (MAE)** 和 **扩散模型（Diffusion Model）** 的视觉生成与重建方法。它利用 MAE 快速训练编码器-解码器进行图像潜变量的重建，同时借助扩散模型逐步采样，实现高质量图像潜变量的生成。

## 核心思想

**训练阶段**：类似 MAE，模型在编码器中对未遮挡的潜变量 token 进行编码，解码器则利用这些编码信息对遮挡部分进行重建，同时用扩散损失（Diffusion loss）对潜变量的分布进行建模和正则化。
**生成阶段**：利用扩散模型的逐步采样机制，通过多轮迭代从噪声中生成潜变量 token，再通过解码器映射成完整潜变量，最后解码为图像。

## 主要模块

**编码器**：基于 Vision Transformer，输入未遮挡的潜变量 token，输出潜在表示。
**解码器**：接受编码器输出和 mask 信息，恢复完整潜变量 token。
**扩散模型（Diffusion Loss）**：作为正则项，约束潜变量的生成过程，确保潜变量分布与训练分布一致。

## 数学形式

令输入图像的潜变量序列为 $x$，对应遮挡掩码为 $m$，编码器输出为 $E(x, m)$，解码器输出为 $D(E(x, m), m)$，则训练目标为最小化重构损失与扩散损失的组合：

$$\mathcal{L} = \underbrace{\|x_{masked} - D(E(x_{unmasked}, m), m)\|^2}_{\text{重构损失}} + \lambda \cdot \underbrace{\text{DiffusionLoss}(z, x)}_{\text{扩散正则}}$$

其中，$z$ 为解码器输出的潜变量表示。

## 训练流程

1. 输入图像经过 VAE 得到潜变量序列 $x$。

2. 随机采样遮挡掩码 $m$，将 $x$ 中部分 token 遮挡。
3. 编码器输入未遮挡的 token，输出编码表示。
4. 解码器输入编码表示和掩码，重建完整潜变量 $z$。
5. 计算重构损失和扩散损失，反向传播优化模型参数。

## 生成采样流程

1. 初始化潜变量 token 序列为噪声。
2. 通过多轮迭代，逐步更新潜变量 token：
    - 计算当前 token 的编码表示。
    - 利用扩散模型预测采样潜变量。
    - 根据采样顺序逐步更新遮挡掩码和待生成 token。
3. 最终解码器输出完整潜变量，映射为图像。

## 伪代码示意

```
def forward(imgs):
    x = patchify(imgs)                          # VAE潜变量token序列
    orders = sample_random_orders(batch_size=x.size(0))
    mask = generate_mask(x, orders)             # 遮挡掩码

    encoded = encoder(x, mask, class_emb)    # 编码器
    decoded = decoder(encoded, mask)         # 解码器

    loss = diffusion_loss(decoded, x, mask) # 结合扩散模型的损失
    return loss

def sample_tokens(batch_size, num_steps, cfg):
    mask = all_masked(batch_size)
    tokens = init_noise(batch_size)
    orders = sample_random_orders(batch_size)

    for step in range(num_steps):
        encoded = encoder(tokens, mask)
        decoded = decoder(encoded, mask)

        # 计算下一步掩码，采样新token
        mask_next = update_mask(mask, orders, step)
        tokens[mask_to_predict] = diffusion_sample(decoded, temperature, cfg)
        mask = mask_next

    imgs = unpatchify(tokens)
    return imgs
```

# Diffusion

扩散模型是一类强大的生成模型，主要通过逐步去噪过程从随机噪声中生成图像。其基本思想是通过训练模型来学习数据分布的反向扩散过程，从而能够从噪声中恢复出高质量图像。

# 1. 扩散模型的基本流程

扩散模型包括两个阶段：

- **正向扩散过程（Forward Process）**：逐步向图像添加噪声，使其最终接近各向同性高斯分布。
- **反向扩散过程（Reverse Process）**：训练一个模型以逐步去噪，重建原始图像。

## 正向过程公式

给定原始图像 $\mathbf{x}_0$，正向过程构造一系列加噪图像 $\mathbf{x}_1, \ldots, \mathbf{x}_T$：

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I})$$

其中 $\beta_t$ 是时间步 $t$ 的噪声强度，通常是线性或余弦调度的。

## 直接从 $\mathbf{x}_0$ 生成 $\mathbf{x}_t$ 的简化公式：

$$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I})$$

其中 $\bar{\alpha}_t = \prod_{s=1}^{t}(1-\beta_s)$。

# 2. 学习反向过程

目标是学习从噪声数据中一步步恢复原始图像的过程：

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \Sigma_\theta(\mathbf{x}_t, t))$$

多数方法仅预测均值 $\mu_\theta$ 或是直接预测噪声 $\epsilon_\theta$。

# 3. 损失函数

通过重参数化技巧，常用损失函数如下：

$$\mathcal{L}_{\text{simple}} = \mathbb{E}_{t,\mathbf{x}_0,\epsilon}\left[\|\epsilon - \epsilon_\theta(\mathbf{x}_t, t)\|^2\right]$$

其中 $\epsilon$ 是标准高斯噪声。

# 4. 条件扩散模型（Conditional Diffusion）

在标准扩散模型中加入条件信息 $y$（例如文本、标签）可构建条件生成模型：

$$\epsilon_\theta(\mathbf{x}_t, t, y)$$

如：

- **Guided Diffusion**：在网络中注入文本信息进行引导。
- **Classifier Guidance**：训练一个分类器引导生成。

# 5. 推理过程（采样）

使用反向扩散过程迭代生成图像：

```
# 给定：一个真实图像 x_0 ~ data
x_0 ← sample_from_dataset()

# 随机选择一个时间步 t ∈ {1, ..., T}
```

```
t ← random_integer(1, T)

# 从标准高斯分布采样噪声
ϵ ← sample from N(0, I)

# 正向扩散：将 x_0 加噪生成 x_t
x_t ← sqrt(ᾱ_t) * x_0 + sqrt(1 - ᾱ_t) * ϵ
# ᾱ_t 是累计噪声系数，预先根据 β_t 计算得到

# 预测噪声（或 x_0）：
ϵ_θ ← neural_net(x_t, t)

# 损失函数（常用 MSE 预测噪声）：
L ← MSE(ϵ, ϵ_θ)

# 反向传播并优化模型参数 θ
L.backward()
optimizer.step()
```

# Flow Matching 与 Mean Matching

## 背景

在生成建模中，我们希望学习从输入分布 $x \sim p_0$ 到高斯噪声 $\varepsilon \sim p_1$ 的变换路径。理想情况下，我们希望显式地建模这个轨迹或路径（trajectory）：

$$\phi(t, x) : [0, 1] \times \mathbb{R}^d \to \mathbb{R}^d$$

但是从起点到终点的路径是非唯一的，存在无数种可能。于是我们希望找到一条最优路径 —— 最小代价路径。

## Flow Matching（条件速度匹配）

### 核心思想

- 从 $x_0 \sim p_0$ 到 $x_1 \sim p_1$ 存在无数条路径，但我们只想要最优路径。
- 直接建模边际速度场（marginal velocity）困难，因为路径未知。
- 条件速度场是可计算的，且其期望就是最优的边际速度。
- 所以目标转为拟合条件速度场，使其期望等于最优路径的边际速度。
- 换句话说，路径的期望等价于条件速度场的期望。

### ODE 形式表示

生成过程可理解为沿时间维度从 $x_0$ 向 $x_1$ 运动。这个运动由速度场 $v_\theta(t, x)$ 决定，对应一阶常微分方程：

$$\frac{dx}{dt} = v_\theta(t, x)$$

速度未知，用神经网络拟合，通过 Euler 积分或其他数值方法求解路径。

### 可学习目标（Flow Matching Loss）

- 利用线性插值构造路径：

$$x_t = (1-t)\cdot x_0 + t\cdot x_1$$

- 目标速度：

$$v^*(t, x_t) = x_1 - x_0$$

- 训练目标：

$$L_{flow} = \mathbb{E}_{x_0,x_1,t}\left[\|v_\theta(t,x_t) - v^*(t,x_t)\|^2\right]$$

## Mean Matching（均值轨迹匹配）

Kaiming He 提出端到端生成模型 Mean Flow Matching，核心思想是：

- 不再拟合瞬时速度，而是一步到位地学习从 $x_0 \to x_1$ 的整体路径。
- 将 ODE 中的速度积分表示直接用均值差表示：

$$x_1 = x_0 + \int_0^1 v_\theta(t, x_t)dt$$

- 设计损失函数让预测的轨迹积分结果等于目标：

$$L_{mean} = \mathbb{E}_{x_0,x_1}\left[\|x_1 - (x_0 + \int_0^1 v_\theta(t,x_t)dt)\|^2\right]$$

这种方式绕过了逐步 Euler 积分过程，实现了一步式生成，类似于 MAE 对比 Diffusion 的结构性优势。

## 总结对比

| 方法 | 核心 | 特点 |
| --- | --- | --- |
| Flow Matching | 拟合速度场 | 类似 score matching，逐步生成 |
| Mean Matching | 拟合整体轨迹 | 端到端训练，一步生成 |

# TiTok(Transformer-based 1-Dimensional Tokenizer)

TiTok 是一种基于 Transformer 的创新 1D 图像 Tokenizer 框架，旨在用极少量的离散 token（如 32 个）实现高质量的图像重建和生成。它区别于传统基于 2D 网格的 VQ-VAE 类型方法，利用一维序列表示进行图像的编码和解码，赋予了更高的灵活性和压缩效率。

## 相关工作简述

- **图像编码 (Image Tokenization)**传统的图像压缩方法多采用自动编码器（Autoencoder），典型如 VAE、VQ-VAE 和 VQGAN。它们通过将图像编码成二维网格的离散 latent 表示，再通过解码器还原图像。尽管诸如 RQ-VAE、MoVQ、MAGVIT-v2 等方法在量化策略或架构上有所创新，但绝大多数仍沿用 2D 网格 latent 表示的设计。TiTok 创新点在于提出了 1D 序列作为 latent 表示，打破了传统网格的限制。

- **图像理解任务中的 Tokenization**多模态大模型（MLLMs）通常使用 CLIP 等编码器生成高语义层次的 token，适用于图像分类、问答等任务。但这类 token 无法很好还原图像的细节和布局。
  TiTok 目标不同，既要还原图像的高层语义，也要兼顾低层次细节，类似 VQ-VAE 的设计思路。
- **图像生成方法**
  主流生成模型包括基于 VAE、GAN、Diffusion 及自回归 Transformer 等。自回归模型如 VQGAN Transformer 需要逐 token 生成，计算开销较大。
  TiTok 采用 MaskGIT 的非自回归生成策略，显著提升生成效率。

# TiTok

## 1D Tokenization 框架

- **架构组成**TiTok 由一个 Vision Transformer (ViT) 编码器（Enc）、向量量化器（Quant）和一个 ViT 解码器（Dec）组成。输入图像被切分为 patches，并映射为 patch tokens，记作 $P \in \mathbb{R}^{H_f \times W_f \times D}$。同时引入固定数量 $K$ 的 learnable latent tokens，记作 $L \in \mathbb{R}^{K \times D}$。将这两部分 token 拼接后输入 ViT 编码器，编码器输出中仅保留 latent tokens，形成灵活且紧凑的 1D latent 表示 $Z_{1D} \in \mathbb{R}^{K \times D}$，该表示与图像分辨率无关，具有更高的适应性。
- **解码阶段**
  TiTok 解码器采用基于 Transformer 的架构，通过学习利用编码器中的图像特征来引导生成过程。具体做法是将量化后的 latent tokens 与 mask tokens 拼接后输入解码器，解码器结合编码器输出的上下文信息，有效提升图像的重建细节和生成质量。

## 图像生成

- 在训练阶段，随机掩码一部分 latent tokens，解码器学习预测被掩盖 token 的真实离散编码。
- 推理时，采用 MaskGIT 的多轮迭代采样策略，从全 mask tokens 序列逐步生成完整的 token 序列。
- 该方法相比传统自回归模型大幅提高生成速度。
- 解码器对编码器特征的利用有效保证了生成过程中的语义一致性和细节还原，即使在压缩 latent 维度的情况下，仍能保持较高的生成质量。

## 两阶段训练策略

- **第一阶段（Warm-up）**直接训练 TiTok 使用已有 MaskGIT-VQGAN 模型生成的“代理码本（proxy codes）”作为训练目标，避免复杂的对抗损失和多重辅助损失，专注于优化 1D token 表示。TiTok 解码器输出代理码本 token，后续通过预训练的 VQGAN 解码器转换为最终 RGB 图像。
- **第二阶段（Decoder Fine-tuning）**
  保持编码器和量化器参数冻结，仅微调解码器以提升重建质量，使用典型的 VQGAN 损失函数。
  该策略显著提升训练稳定性和图像生成质量。

# 总结

TiTok 打破传统二维网格 token 表示，采用紧凑灵活的一维 token 序列表示，结合 MaskGIT 高效非自回归生成策略和解码器对编码器特征的有效利用，实现了在保持高质量重建与生成的同时，大幅减少 token 数量和计算开销。
两阶段训练策略利用代理码本简化训练流程，为实现稳定且高效的图像 tokenization 与生成提供了新思路。

# SoftVQ-VAE 核心点解析

SoftVQ-VAE 的核心创新在于使用**对多个码本向量（codewords）进行加权线性组合**来表示潜在向量，而非传统 VQ-VAE 中的硬量化。

## 传统硬量化表示

传统 VQ-VAE 通过最近邻查找，将潜在向量 $\hat{z}$ 硬性映射到码本中的一个向量：

$$z = c_i, \quad i = \arg\min_j \|\hat{z} - c_j\|^2$$

其中，$c_j$ 是码本中的第 $j$ 个码本向量。

## SoftVQ-VAE 软量化表示

SoftVQ-VAE 引入了一个基于距离的 softmax 权重，计算潜在向量在所有码本向量上的概率分布：

$$q_\phi(z|x) = \text{Softmax}\left(-\frac{\|\hat{z} - C\|^2}{\tau}\right)$$

然后将潜在向量表示为码本向量的加权线性组合：

$$z = \sum_{j=1}^{K} q_\phi(z = j|x) \cdot c_j$$

- $C = \{c_j\}_{j=1}^K$ 是码本集合
- $\tau$ 是温度参数，控制 softmax 分布的平滑度

## 空间对齐（Representation Alignment）

为了让潜在表示与图像的空间结构对应，SoftVQ-VAE 通过**重复已经组合过的 latent tokens** 来对齐到输入图像的空间尺度。具体做法是：

$$z_r = [\underbrace{z_0, \ldots, z_0}_{N/L \text{ 次}}, \underbrace{z_1, \ldots, z_1}_{N/L \text{ 次}}, \ldots, \underbrace{z_{L-1}, \ldots, z_{L-1}}_{N/L \text{ 次}}]$$

其中：

- $L$ 是 latent tokens 数量
- $N$ 是图像 patch tokens 数量
- $z_r$ 是重复扩展后的 latent tokens 序列

随后，通过一个多层感知机（MLP）对扩展后的 $z_r$ 进行投影，使其与预训练视觉编码器的图像特征 $y^*$ 对齐：

$$\mathcal{L}_{align} = \frac{1}{N} \sum_{n=1}^{N} \text{sim}\left(y_n^*, \text{MLP}(z_r[n])\right)$$

此空间对齐有助于提升潜在空间的语义表达能力和下游生成模型的效果。

## 优点总结

- **可微分**：整个过程端到端可训练，无需复杂的梯度估计技巧
- **丰富表达能力**：潜在表示是多个码本向量的组合，更灵活

- **减少量化误差**：相比硬量化更平滑，提升重建质量
- **空间对齐**：通过重复 latent tokens 实现与输入图像空间结构的对应
- **降低潜在长度需求**：用更少的 latent tokens 达到更好效果

**总结：**
SoftVQ-VAE 通过软量化的线性组合策略和空间对齐机制，有效增强了潜在空间的表达能力和训练的稳定性，是其区别于传统 VQ-VAE 的关键技术点。

# LFQ (Lookup-Free Quantization)

传统的VQ-VAE通过高维码本嵌入向量进行量化，码本维度$d$通常较大，且每个码本向量是连续值，这样在词汇量较大时会导致查表开销大，且训练难度高。

而LFQ提出了一种极简的设计思路：

- **量化向量的每个维度只有两个可能取值，记作二值码本：**

$$C_i = \{-1, 1\}$$

- **潜变量向量$z \in \mathbb{R}^{\log_2 K}$中的每个分量独立量化为$-1$或$1$：**

$$q_p(z_i) = \text{sign}(z_i) = \begin{cases} -1 & z_i \leq 0 \\ 1 & z_i > 0 \end{cases}$$

- **多个这样的二值维度组合，形成码本容量为$K = 2^{\log_2 K}$的码本空间，也就是$K$个token：**

$$C = \prod_{i=1}^{\log_2 K} C_i$$

- **这样，虽然每个维度信息极少，但通过多维组合可以表示丰富多样的token，避免了高维连续码本的查表开销，提高码本利用率和表达能力。**
- **token索引计算简单为：**

$$\text{Index}(q_p(z)) = \sum_{i=1}^{\log_2 K} \mathbf{1}_{z_i > 0} \cdot 2^{i-1}$$

这种设计使得：

- 码本维度大幅度降低，嵌入维度为1（即二值化），
- 码本容量通过维度的笛卡尔积指数增长，
- 计算效率和训练稳定性均得到提升，
- 码本利用率更高，丰富性也更强。

因此，LFQ创新地用**多码本低维度二值组合**的方式，实现了高效且表达力强的视觉分词量化。

# RQ（Residual Quantization）

Residual Quantization (RQ) 的核心创新点在于：

- **递归残差量化**RQ不是对特征向量进行一次性量化，而是先对原始向量进行第一次量化，然后计算量化残差（原始向量与量化向量的差），接着对残差继续进行量化，如此递归多次。通过这种方式，RQ能够逐步减少量化误差，实现更精细的向量近似。

- **单一共享码本**RQ使用同一个共享码本进行多层残差的量化，而不是为每层残差设计不同的码本。这种设计降低了超参数复杂度，同时使码本的每个码字可以在不同层次复用，提高码本利用率。
- **指数级的表示能力提升**由于量化深度为$D$时，RQ可以表示的码字组合数量为$K^D$（$K$为码本大小），大大提升了码本的表达能力，等价于VQ中使用$K^D$大小的超大码本，但保持了较小的码本规模。
- **提升图像重构与生成质量**通过更精确的特征量化，RQ-VAE重构的图像更真实，且允许在保持图像质量的前提下降低特征图的空间分辨率，减轻后续自回归模型的计算负担。
- **结合深度分层Transformer进行高效生成**RQ-Transformer设计了空间Transformer和深度Transformer两阶段结构，有效捕捉空间和深度维度的依赖关系，降低计算复杂度，提升生成速度和质量。
- **软标签与随机采样训练策略**
  通过利用码本间的相似度，采用软标签训练，以及训练阶段随机采样代码，缓解训练与推理阶段的分布差异（曝光偏差），提高模型鲁棒性。

RQ的这一系列创新有效解决了传统VQ-VAE码本膨胀带来的效率和稳定性问题，实现了高效、精准的图像量化及高质量的自回归生成。

# MaskGIT

MaskGIT 的主要创新点包括：

- **非自回归的并行生成**与传统自回归模型（如GPT系列）逐步生成不同，MaskGIT采用非自回归生成方式，可以同时预测多个token，大幅提升生成速度。
- **基于掩码语言模型的生成机制**类似BERT的双向Transformer结构，MaskGIT在训练时对输入序列随机掩码，模型学习预测被掩盖的token，从而获得上下文的双向信息。
- **迭代更新掩码策略**生成阶段从全掩码序列开始，模型预测当前未掩码位置的token概率分布，再根据概率最低的token位置重新掩码，逐步减少掩码比例，迭代优化生成结果。
- **利用已生成token继续生成**通过保留模型当前确定的token，将其作为上下文继续进行后续预测，使生成过程能充分利用已生成信息，提高生成质量。
- **融合非自回归效率与自回归质量**
  通过迭代掩码更新策略，MaskGIT兼顾了生成速度和生成质量，在很多视觉生成任务中表现出色。

总体来说，MaskGIT的创新核心在于将掩码语言模型应用于生成任务，实现了高效的并行生成，同时利用迭代掩码机制逐步优化生成结果，弥补了传统非自回归方法质量不足的缺点。

# VAR（Visual Autoregressive Model）创新点总结

- **多尺度图像表征**VAR利用多尺度的图像表示，将图像在不同分辨率下进行建模，有助于捕捉图像的全局结构和局部细节。
- **插值机制**在较低分辨率的特征图上进行预测后，通过插值将信息传递到更高分辨率，实现逐步细化图像内容。
- **残差细节刻画**利用残差连接或残差学习机制，模型在每个尺度上专注于补充和勾勒细节信息，提高图像的精细度和质量。
- **高效的自回归生成**
  通过多尺度和残差的设计，VAR在保证生成质量的同时，提高了生成效率，缓解了单尺度自回归模型的计算负担。

综上，VAR的创新点在于结合多尺度特征与残差细节刻画，通过插值实现逐层细化，有效提升图像生成的质量与速度。

# FSQ（Finite Scalar Quantization）——VQ-VAE 简化版创新点总结

- **简化的标量量化方法**FSQ采用基于四舍五入的标量量化策略，替代了传统VQ-VAE中复杂的向量量化（Vector Quantization），极大简化了量化过程。
- **无需训练码本**与传统VQ-VAE依赖学习码本不同，FSQ使用固定的标量区间和四舍五入规则，无需额外学习，降低了训练复杂度。
- **高效且稳定**
  通过简单的四舍五入实现量化，FSQ避免了向量量化中的不连续梯度问题，提高了训练的稳定性和效率。

总结：FSQ通过用简单的四舍五入标量量化替代复杂向量量化，简化了VQ-VAE框架，提升了训练效率和稳定性，同时保持了良好的生成性能。

# VQGAN-LC 核心创新点总结

- **码本初始化**利用预训练视觉编码器（如CLIP ViT）在目标数据集上提取Patch级特征，进行K-means聚类得到码本中心，初始化静态码本，解决随机初始化导致的低利用率问题。
- **码本冻结 + 投影器训练**
  码本本身保持冻结，不直接优化，训练一个线性投影器（projector）将码本映射到编码器特征空间，有效对齐分布，提升量化效果。

综上，VQGAN-LC的创新核心在于结合预训练特征的静态大码本初始化与投影器训练机制，实现超大码本高效利用，提升图像量化质量和生成性能。