Diffusion Model 简单介绍

目录

常见Diffusion Model建模

> 这一节主要讲的是，从不同角度来理解生成模型的建模
>
> 此处参考了苏神的科学空间的内容，并在一些地方附加了其他的例子帮助理解

1. DDPM(直觉理解版本)

让我们首先从直觉的角度上来理解下生图的过程，在生活中提到生成模型很容易看到这样的逐步去噪图：

可以得到的信息是：

> **Caution**
>
> 我们需要一个可以***逐步预测***从“脏图像”到“干净一点的脏图像”的模型

完成这个工作的模型不是白来的，我们需要对应的数据来让模型学会这个“去噪的过程”

一个天然的想法就是：**如果我有一张干净的图片，我给他逆过来不断加噪，不就得到了足够训练这个模型的数据了吗。**

现实里也是这样做的：

$$x_t = \alpha_t x_{t-1} + \beta_t \epsilon_t, \quad \epsilon_t \sim \mathcal{N}(0, I) \tag{1}$$

这里的x_{t-1}代表着比较干净的图，x_t代表着x_{t-1}时刻加噪一次后的图像。ϵ_t是独立的随机噪声。

之后，我们递归的把等式右边的x_{t-1}都换掉，就可以得到**纯噪声图片**和**干净图片**的映射关系：

$$\begin{aligned} x_t &= \alpha_t x_{t-1} + \beta_t \epsilon_t \\ &= \alpha_t(\alpha_{t-1} x_{t-2} + \beta_{t-1}\epsilon_{t-1}) + \beta_t \epsilon_t \\ &= \dots \\ &= (\alpha_t \dots \alpha_1) x_0 + \underbrace{(\alpha_t \dots \alpha_2)\beta_1\epsilon_1 + (\alpha_t \dots \alpha_3)\beta_2\epsilon_2 + \dots + \alpha_t\beta_{t-1}\epsilon_{t-1} + \beta_t\epsilon_t}_{\text{多个相互独立的正态噪声之和}} \end{aligned} \tag{2}$$

式子的右边不太优雅，不如给他加个约束：

$$\alpha_t^2 + \beta_t^2 = 1 \tag{3}$$

这个的好处是，因为上面的ϵ_t都是独立的均值为0、方差为1的Gaussian。而独立的Gaussian可加，后面一堆东西就变成了均值为0，方差为$1-(\alpha_t \dots \alpha_1)^2$的标准高斯了

$$(\alpha_t \dots \alpha_2)^2\beta_1^2 + (\alpha_t \dots \alpha_3)^2\beta_2^2 + \dots + \alpha_t^2\beta_{t-1}^2 + \beta_t^2 = 1 - (\alpha_t \dots \alpha_1)^2 \tag{4}$$

我们再把**纯噪声图片**和**干净图片**的映射关系重写一下：

$$x_t = \underbrace{(\alpha_t \cdots \alpha_1)}_{\text{记为 } \bar{\alpha}_t} x_0 + \underbrace{\sqrt{1-(\alpha_t \cdots \alpha_1)^2}\,\bar{\epsilon}_t}_{\text{记为 } \bar{\beta}_t}, \quad \bar{\epsilon}_t \sim \mathcal{N}(0, I) \tag{5}$$

> 其实这里就只有一个变量了

好了，现在我们得到了一个很简洁的建模方式来关联任意两张噪声图了。是时候让模型去学去噪过程了。

再梳理下我们模型的目标：

- *输入: 噪声图*x_t
- *输出: 比较干净的图*x_{t-1}

一个很直觉的想法是，最小化**预测图片**$\mu(x_t)$和**GT(Ground True)图片**x_{t-1}的欧氏距离

$$Loss = \|\mathbf{x}_{t-1} - \boldsymbol{\mu}(\mathbf{x}_t)\|^2 \tag{6}$$

我们不妨让：

$$\boldsymbol{\mu}(\mathbf{x}_t) = \frac{1}{\alpha_t}(\mathbf{x}_t - \beta_t \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)) \tag{7}$$

带入Loss函数可以得到：

$$Loss = \|\mathbf{x}_{t-1} - \boldsymbol{\mu}(\mathbf{x}_t)\|^2 = \frac{\beta_t^2}{\alpha_t^2}\|\boldsymbol{\epsilon}_t - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\|^2 \tag{8}$$

又因为x_t可以进一步化简：

$$\begin{aligned} \mathbf{x}_t &= \alpha_t \mathbf{x}_{t-1} + \beta_t \boldsymbol{\epsilon}_t \\ &= \alpha_t(\bar{\alpha}_{t-1}\mathbf{x}_0 + \bar{\beta}_{t-1}\bar{\boldsymbol{\epsilon}}_{t-1}) + \beta_t \boldsymbol{\epsilon}_t \\ &= \bar{\alpha}_t \mathbf{x}_0 + \alpha_t \bar{\beta}_{t-1}\bar{\boldsymbol{\epsilon}}_{t-1} + \beta_t \boldsymbol{\epsilon}_t \end{aligned} \tag{9}$$

带入(8)可得：

$$Loss = \|\boldsymbol{\epsilon}_t - \boldsymbol{\epsilon}_\theta(\bar{\alpha}_t \mathbf{x}_0 + \alpha_t \bar{\beta}_{t-1}\bar{\boldsymbol{\epsilon}}_{t-1} + \beta_t \boldsymbol{\epsilon}_t, t)\|^2 \tag{10}$$

> **Tip**
>
> 这里的带\bar的参数代表着从干净图到目前状态，而不带\bar的则代表是单步的

好了，理论上来说我们已经建模好这个问题，可以开始训模型了。但是实际上我们会发现，这个Loss里含有

1. x_0
2. $\bar{\boldsymbol{\epsilon}}_{t-1}$
3. t
4. $\boldsymbol{\epsilon}_t$

四个随机数，训练起来会非常不稳定，所以还需要进一步的简化，看看能不能把随机变量进一步减少。

我们不妨换个视角，把($\bar{\boldsymbol{\epsilon}}_{t-1}$, $\boldsymbol{\epsilon}_t$)看作一组正交积。

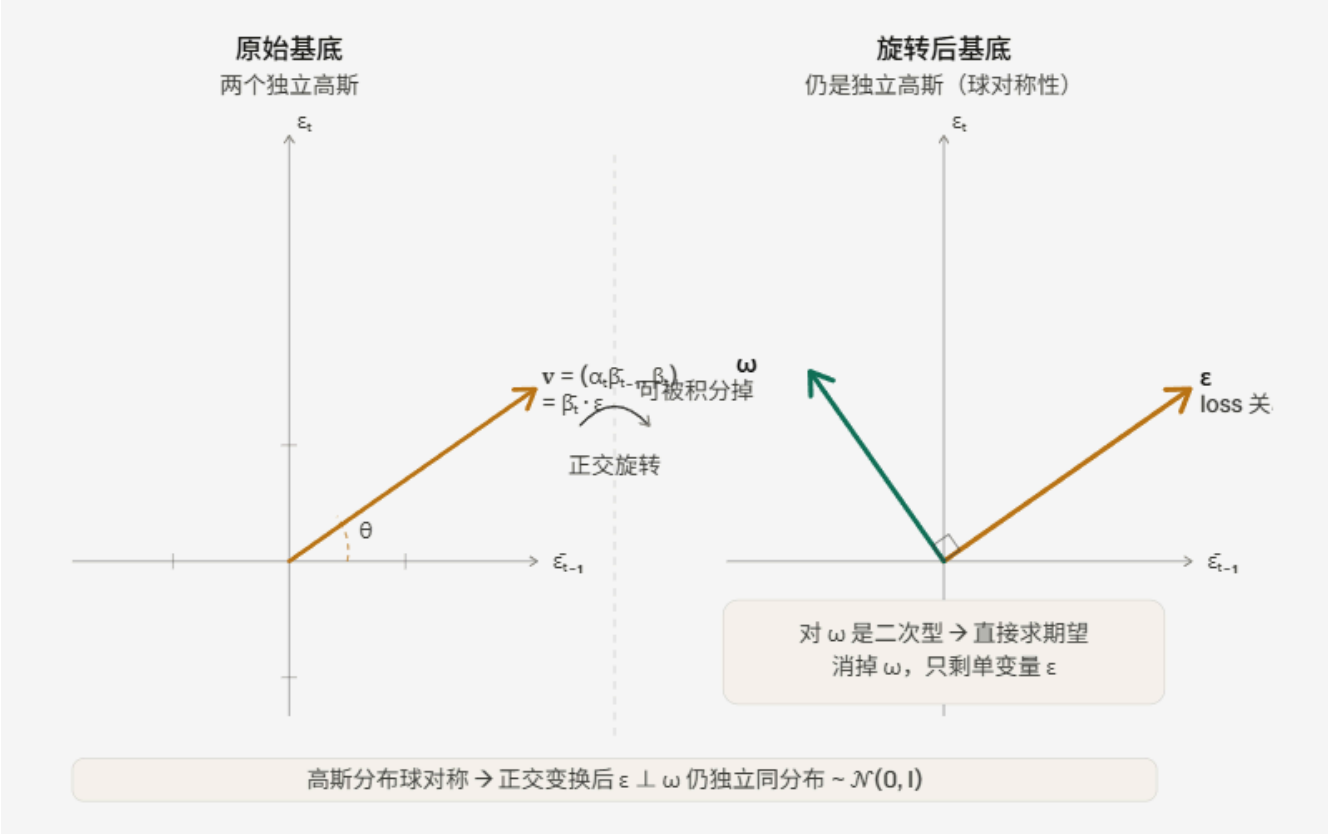
这样，Eq. (10)里的 $\alpha_t \bar{\beta}_{t-1}\bar{\boldsymbol{\epsilon}}_{t-1} + \beta_t \boldsymbol{\epsilon}_t$ 可以看成在这个二维空间里沿某个**方向**做投影，这个方向的系数向量是：

$$\mathbf{v} = (\alpha_t \bar{\beta}_{t-1}, \beta_t) \tag{11}$$

Gaussian的线性组合又一次帮了我们，还记得我们一开始为了让公式更简单设置的约束(3)吗， 它让v的模恰好是：

$$\|\mathbf{v}\| = \sqrt{\alpha_t^2 \bar{\beta}_{t-1}^2 + \beta_t^2} = \bar{\beta}_t \tag{12}$$

此时自然想到，我只要把这个二维空间的坐标系旋转到v的方向上，那就可以把$\bar{\boldsymbol{\epsilon}}_{t-1}$和$\boldsymbol{\epsilon}_t$俩随机变量变成一个了！



> 让claude画的示意图，大致准确

既然已经理解了原理，具体计算就下略了，最后我们可以得到这样的化简结果：

$$Loss = \left\| \boldsymbol{\epsilon} - \frac{\beta_t}{\bar{\beta}_t} \boldsymbol{\epsilon}_\theta(\bar{\alpha}_t \mathbf{x}_0 + \bar{\beta}_t \boldsymbol{\epsilon}, t) \right\|^2 \tag{13}$$

ok，到这里为止，我们就从直觉观察到的现象出发，把DDPM（Denoising Diffusion Probabilistic Models）的训练目标推导出来了，更多的细节可以去最后的Reference部分参考DDPM原文 :)

2. DDPM(概率推导版本)

在直觉理解版本中，我们得到了 DDPM 的训练目标。这一节，我们将换用概率论的语言重新推导同一套框架——不是为了重复，而是因为这套语言能让我们更清晰地看到 DDPM 背后的假设边界在哪里。正是这些假设，构成了后续 DDIM 改进的出发点。

让我们回溯到最开始的Eq.(1)

$$x_t = \alpha_t x_{t-1} + \beta_t \epsilon_t, \quad \epsilon_t \sim \mathcal{N}(0, I) \tag{1}$$

其实换一种角度看，x_t就是均值是$\alpha_t x_{t-1}$为均值，${\beta_t}^2$为方差的Guassian

$$p(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \alpha_t \mathbf{x}_{t-1}, \beta_t^2 \mathbf{I}) \tag{14}$$

同理，Eq. (2)也可以递归的得到：

$$p(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \bar{\alpha}_t \mathbf{x}_0, \bar{\beta}_t^2 \mathbf{I}) \tag{15}$$

因为此时的求解目标是：$p(\mathbf{x}_{t-1}|\mathbf{x}_t)$。我们自然可以想到用**贝叶斯**求解：

$$p(\mathbf{x}_{t-1}|\mathbf{x}_t) = \frac{p(\mathbf{x}_t|\mathbf{x}_{t-1}) p(\mathbf{x}_{t-1})}{p(\mathbf{x}_t)} \tag{16}$$

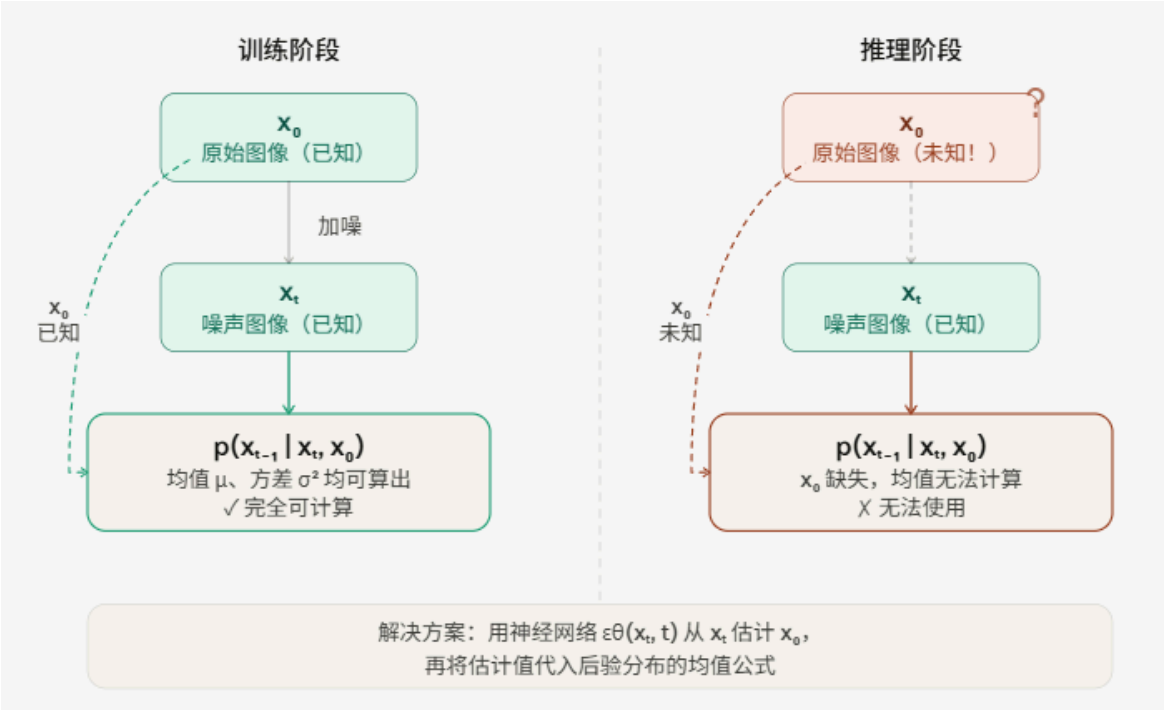
> 这里绿色是已知的，红色是未知的

离能求解还差一小步，可以想到虽然我们无法求解x_t的**独立概率**$p(\mathbf{x}_{t-1})$，但是我们可以求x_t的**条件概率**$p(\mathbf{x}_{t-1}|\mathbf{x}_0)$：

$$p(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0) = \frac{p(\mathbf{x}_t|\mathbf{x}_{t-1},\mathbf{x}_0)p(\mathbf{x}_{t-1}|\mathbf{x}_0)}{p(\mathbf{x}_t|\mathbf{x}_0)} = \mathcal{N}\left(\mathbf{x}_{t-1}; \underbrace{\frac{\alpha_t\bar{\beta}_{t-1}^2}{\bar{\beta}_t^2}\mathbf{x}_t + \frac{\bar{\alpha}_{t-1}\beta_t^2}{\bar{\beta}_t^2}\mathbf{x}_0}_{\boldsymbol{\mu}_q(\mathbf{x}_t,\mathbf{x}_0)}, \underbrace{\frac{\bar{\beta}_{t-1}^2\beta_t^2}{\bar{\beta}_t^2}}_{\sigma_q^2(t)}\mathbf{I}\right) \tag{17}$$

这下等式的右侧都是已知的变量了。不过还是一样的问题，理论可解并不代表在实际推理上可行

虽然公式 (17) 给出了完美的后验分布，但它依赖于推理阶段不可知的 $\mathbf{x}_0$。我们的最终目标是仅凭当前的噪声观测值 $\mathbf{x}_t$ 来预测上一时刻的状态 $\mathbf{x}_{t-1}$。



到现在为止，我们都尝试在用数学推导的方式得到优化目标，Diffusion Model中的"Model"还没发挥过作用，在这里，我们就用这个预测网络来打上最后的补丁：

用$\bar{\mu}(x_t)$来估算x_0，此时我们还是选择两者差的二范数作为损失函数

此时可得：

$$p(\mathbf{x}_{t-1}|\mathbf{x}_t) \approx p(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0=\bar{\boldsymbol{\mu}}(\mathbf{x}_t)) = \mathcal{N}\left(\mathbf{x}_{t-1}; \frac{\alpha_t\bar{\beta}_{t-1}^2}{\bar{\beta}_t^2}\mathbf{x}_t + \frac{\bar{\alpha}_{t-1}\beta_t^2}{\bar{\beta}_t^2}\bar{\boldsymbol{\mu}}(\mathbf{x}_t), \frac{\bar{\beta}_{t-1}^2\beta_t^2}{\bar{\beta}_t^2}\mathbf{I}\right) \tag{18}$$

我们已经可以拿Eq.(18)来进行训练了，不过更多的做法是预测噪声而不是预测图片，所以一般会再往下走几步：

利用正向过程的重参数化公式 Eq. (1)，我们可以反解出原始图像的理论表达式：

$$\mathbf{x}_0 = \frac{1}{\bar{\alpha}_t}(\mathbf{x}_t - \bar{\beta}_t\boldsymbol{\epsilon}) \tag{19}$$

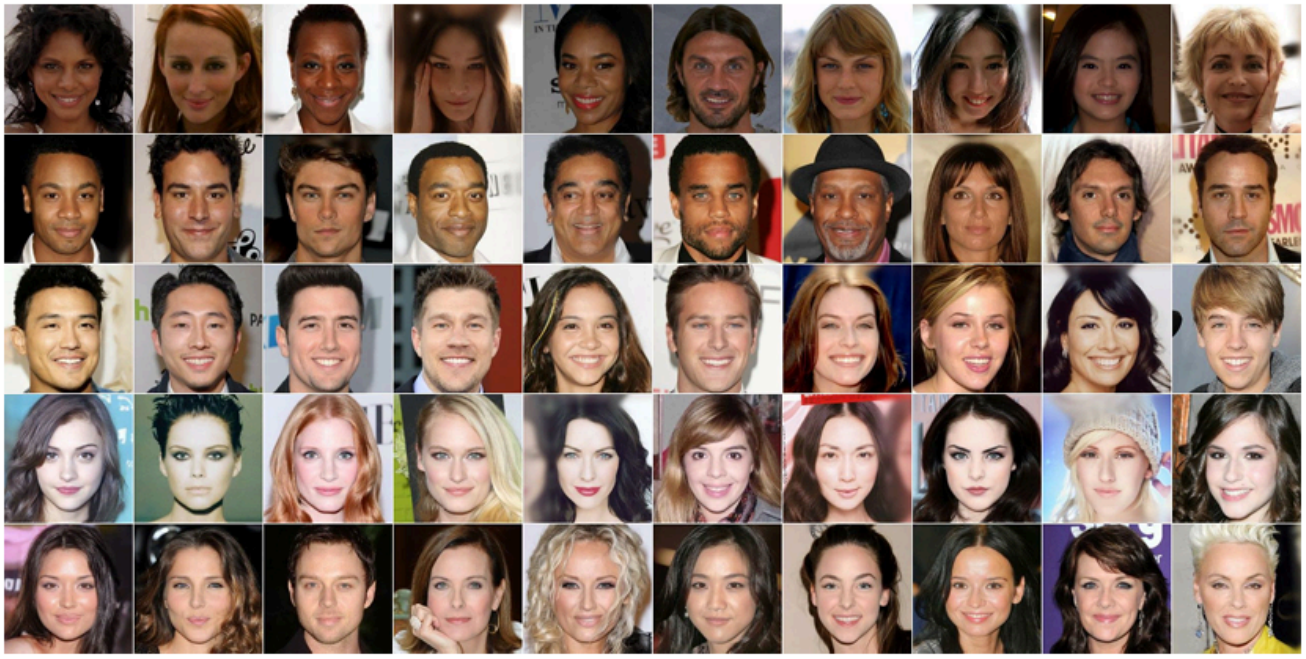
这一线性关系启发我们将对 $\mathbf{x}_0$ 的预测转化为对噪声 $\boldsymbol{\epsilon}$ 的预测。因此，我们引入神经网络 $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t,t)$ 来替代不可知的真实噪声 $\boldsymbol{\epsilon}$，从而将 $\mathbf{x}_0$ 的估计值参数化为：

$$\bar{\boldsymbol{\mu}}(\mathbf{x}_t) = \frac{1}{\bar{\alpha}_t}(\mathbf{x}_t - \bar{\beta}_t\boldsymbol{\epsilon}_\theta(\mathbf{x}_t,t)) \tag{20}$$

将Eq. (20)带入Eq. (18)，化简得到：

$$p(\mathbf{x}_{t-1}|\mathbf{x}_t) \approx p(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0=\bar{\boldsymbol{\mu}}(\mathbf{x}_t)) = \mathcal{N}\left(\mathbf{x}_{t-1}; \frac{1}{\alpha_t}\left(\mathbf{x}_t - \frac{\beta_t^2}{\bar{\beta}_t}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t,t)\right), \frac{\bar{\beta}_{t-1}^2\beta_t^2}{\bar{\beta}_t^2}\mathbf{I}\right) \tag{21}$$

这样，我们就从另一条路推导出了DDPM，也给我们下一个小章节介绍DDIM铺了点路。



> DDPM生图的Case

3. DDIM：基于概率视角的 DDPM 延伸

DDPM已经有了良好的生图质量，但实际用起来是有些小问题，比如说：

> **Important**
>
> **推理的时候，去噪的步数必须和训练时候的去噪步数相同，不能做跳步**

出现这个问题的原因我们可以从概率推导版本中窥见一二，但是 DDPM 的后验 $p(x_{t-1}|x_t)$ 是通过贝叶斯公式从 $p(x_t|x_{t-1})$ 推出来的，这个公式**只能走相邻的一步**，没办法直接跳。

回头看 DDPM 的推导，我们真的需要这个马尔可夫假设吗？

仔细看推导里实际用了什么：

- $p(x_t|x_0) = \mathcal{N}(x_t;\ \bar{\alpha}_t x_0,\ \bar{\beta}_t^2 I)$ ← 用了
- $p(x_{t-1}|x_0) = \mathcal{N}(x_{t-1};\ \bar{\alpha}_{t-1} x_0,\ \bar{\beta}_{t-1}^2 I)$ ← 用了
- $p(x_t|x_{t-1})$ ← 只是为了通过贝叶斯公式推出后验，**本身不出现在最终的采样公式里**

换句话说，$p(x_t|x_{t-1})$ 是一个**中间工具**，不是目的。最终采样只需要知道后验 $p(x_{t-1}|x_t, x_0)$ 长什么样，而决定后验的其实只是两个边际分布。

那让我们回到Eq. (15)，思考：

- *给定 $p(x_t|x_0)$ 和 $p(x_{t-1}|x_0)$，哪些后验 $p(x_{t-1}|x_t, x_0)$ 是合法的?*

边缘分布积分天然符合：

$$\int p(x_{t-1}|x_t, x_0)p(x_t|x_0)dx_t = p(x_{t-1}|x_0) \tag{22}$$

Eq. (18)中，我们推导出$p(x_{t-1}|x_t, x_0)$ 其实是个Gaussian分布，我们不妨通过待定系数的方式来解Eq. (22)

$$\begin{cases} p(x_{t-1}|x_0) = \mathcal{N}(x_{t-1}; \bar{\alpha}_{t-1}x_0, \bar{\beta}_{t-1}^2\mathbf{I}) \\ p(x_t|x_0) = \mathcal{N}(x_t; \bar{\alpha}_t x_0, \bar{\beta}_t^2\mathbf{I}) \\ p(x_{t-1}|x_t, x_0) = \mathcal{N}(x_{t-1}; \kappa_t x_t + \lambda_t x_0, \sigma_t^2\mathbf{I}) \end{cases} \tag{23}$$

第一项和第二项在之前就已经假设过了，Eq. (23)只是引入了 κ_t，λ_t，σ_t 这三个待定系数

让我们把Eq. (22)和Eq. (23)联合求解，可以得到：

$$\begin{cases}\bar{\alpha}_{t-1}=\kappa_t\bar{\alpha}_t+\lambda_t\\ \bar{\beta}_{t-1}=\sqrt{\kappa_t^2\bar{\beta}_t^2+\sigma_t^2}\end{cases}\tag{24}$$

由于我们少了$p(x_t|x_{t-1})$的约束，所以我们不能像DDPM一样完整的解出这里的待定系数，我们不妨用σ_t来表示κ_t和λ_t：

$$\begin{aligned}\kappa_t&=\frac{\sqrt{\bar{\beta}_{t-1}^2-\sigma_t^2}}{\bar{\beta}_t}\\ \lambda_t&=\bar{\alpha}_{t-1}-\bar{\alpha}_t\frac{\sqrt{\bar{\beta}_{t-1}^2-\sigma_t^2}}{\bar{\beta}_t}\end{aligned}\tag{25}$$

或者写成

$$p(x_{t-1}|x_t,x_0)=\mathcal{N}\left(x_{t-1};\frac{\sqrt{\bar{\beta}_{t-1}^2-\sigma_t^2}}{\bar{\beta}_t}x_t+\left(\bar{\alpha}_{t-1}-\bar{\alpha}_t\frac{\sqrt{\bar{\beta}_{t-1}^2-\sigma_t^2}}{\bar{\beta}_t}\right)x_0,\sigma_t^2\mathbf{I}\right)\tag{26}$$

ok，观察这个式子，现在又遇到了和Eq. (17)一样的问题：**它依赖于推理阶段不可知的** x_0。那我们的做法也很简单：**用**$\bar{\mu}(x_t)$**来估算**x_0。把之前那套重新搬过来就行了。

最后我们整理可以得到：

$$\begin{aligned}p(x_{t-1}|x_t)&\approx p(x_{t-1}|x_t,x_0=\bar{\mu}(x_t))\\&=\mathcal{N}\left(x_{t-1};\frac{1}{\alpha_t}\left(x_t-\left(\bar{\beta}_t-\alpha_t\sqrt{\bar{\beta}_{t-1}^2-\sigma_t^2}\right)\epsilon_\theta(x_t,t)\right),\sigma_t^2\mathbf{I}\right)\end{aligned}\tag{27}$$

它和Eq. (21)的差别在于，多了一个可以调节的超参数σ_t。也正是这个参数，解决了我们在这个小节最开始提到的跳步生成问题：

我们不妨来观察几个特殊的取值：

1. $\sigma_t=\frac{\bar{\beta}_{t-1}\beta_t}{\bar{\beta}_t}$

   这个取值带入Eq. (27)计算一下，可以得到和Eq. (21)**完全一致**的结果，所以把DDPM当成DDIM的一种特例也不是不行

2. $\sigma_t=0$

   当 $\sigma_t=0$ 的时候，Eq. (27)的方差就变成0了，也就是说它变成了一个**确定性采样**。那就可以做到用较多的步数训练，较少的步数推理了。

4. Score-Based Model：从离散到连续

前两节我们一直在离散时间里做推导——T 步加噪，t 步去噪，步数是人为定好的。但真实的加噪过程并没有"第1步、第2步"这样的刻意划分，它本质上是一个连续变化的过程。

如果我们把步数 T 推向无穷，让每一步变得无限小，离散的马尔可夫链就变成了一条连续的轨迹，可以用SDE来描述。这个视角的好处不只是"更优雅"——它把 DDPM、DDIM 以及更多变体统一成了同一个方程的不同参数选择，并且严格推导出了逆向生成过程的形式。

我们不妨把前向过程（把干净图进行加噪）进行如下建模：

$$dx=f_t(x)dt+g_tdw\tag{28}$$

或者：

$$x_{t+\Delta t}-x_t=f_t(x_t)\Delta t+g_t\sqrt{\Delta t}\epsilon,\quad \epsilon\sim\mathcal{N}(0,I)\tag{29}$$

> **Note**
>
> 这里离散化之后为啥变成Δt，可以参考Wiener Processes — Econ 236 2016.03.28 documentation

将Eq. (29)写成概率形式，就可以得到：

$$\begin{aligned}p(\mathbf{x}_{t+\Delta t}|\mathbf{x}_t)&=\mathcal{N}\left(\mathbf{x}_{t+\Delta t};\mathbf{x}_t+\mathbf{f}_t(\mathbf{x}_t)\Delta t,g_t^2\Delta t\mathbf{I}\right)\\&\propto\exp\left(-\frac{\|\mathbf{x}_{t+\Delta t}-\mathbf{x}_t-\mathbf{f}_t(\mathbf{x}_t)\Delta t\|^2}{2g_t^2\Delta t}\right)\end{aligned}\tag{30}$$

可能已经猜到为什么要写成概率形式了，就是和**DDPM概率推导版本**做的事情一样，使用**贝叶斯**：

$$\begin{aligned} p(\mathbf{x}_t|\mathbf{x}_{t+\Delta t}) &= \frac{p(\mathbf{x}_{t+\Delta t}|\mathbf{x}_t)p(\mathbf{x}_t)}{p(\mathbf{x}_{t+\Delta t})} = p(\mathbf{x}_{t+\Delta t}|\mathbf{x}_t)\exp(\log p(\mathbf{x}_t) - \log p(\mathbf{x}_{t+\Delta t})) \\ &\propto \exp\left(-\frac{\|\mathbf{x}_{t+\Delta t} - \mathbf{x}_t - \mathbf{f}_t(\mathbf{x}_t)\Delta t\|^2}{2g_t^2\Delta t} + \log p(\mathbf{x}_t) - \log p(\mathbf{x}_{t+\Delta t})\right) \end{aligned} \tag{31}$$

泰勒展开并代入配方后：

$$p(\mathbf{x}_t|\mathbf{x}_{t+\Delta t}) \propto \exp\left(-\frac{\|\mathbf{x}_{t+\Delta t} - \mathbf{x}_t - [\mathbf{f}_t(\mathbf{x}_t) - g_t^2\nabla_{\mathbf{x}_t}\log p(\mathbf{x}_t)]\Delta t\|^2}{2g_t^2\Delta t} + \mathcal{O}(\Delta t)\right) \tag{32}$$

当 $\Delta t \to 0$ 时，$\mathcal{O}(\Delta t) \to 0$ 不起作用，因此：

$$\begin{aligned} p(\mathbf{x}_t|\mathbf{x}_{t+\Delta t}) &\propto \exp\left(-\frac{\|\mathbf{x}_{t+\Delta t} - \mathbf{x}_t - [\mathbf{f}_t(\mathbf{x}_t) - g_t^2\nabla_{\mathbf{x}_t}\log p(\mathbf{x}_t)]\Delta t\|^2}{2g_t^2\Delta t}\right) \\ &\approx \exp\left(-\frac{\|\mathbf{x}_t - \mathbf{x}_{t+\Delta t} + [\mathbf{f}_{t+\Delta t}(\mathbf{x}_{t+\Delta t}) - g_{t+\Delta t}^2\nabla_{\mathbf{x}_{t+\Delta t}}\log p(\mathbf{x}_{t+\Delta t})]\Delta t\|^2}{2g_{t+\Delta t}^2\Delta t}\right) \end{aligned} \tag{33}$$

即 $p(\mathbf{x}_t|\mathbf{x}_{t+\Delta t})$ 近似一个均值为 $\mathbf{x}_{t+\Delta t} - [\mathbf{f}_{t+\Delta t}(\mathbf{x}_{t+\Delta t}) - g_{t+\Delta t}^2\nabla_{\mathbf{x}_{t+\Delta t}}\log p(\mathbf{x}_{t+\Delta t})]\Delta t$、协方差为 $g_{t+\Delta t}^2\Delta t\mathbf{I}$ 的正态分布。取 $\Delta t \to 0$ 的极限，那么对应于 SDE：

$$d\mathbf{x} = \left[\mathbf{f}_t(\mathbf{x}) - g_t^2\nabla_{\mathbf{x}}\log p_t(\mathbf{x})\right]dt + g_t d\bar{\mathbf{w}} \tag{34}$$

这个SDE就是从噪声图片慢慢变成干净图片（逆向过程）的描述。

不过，还是那个问题，理论成立不一定实际能训练，如果我们想得到这里$p_t(\mathbf{x})$的score function，我们可能需要计算所有样本取平均，这是不可接受的。所以，这里的方法是：

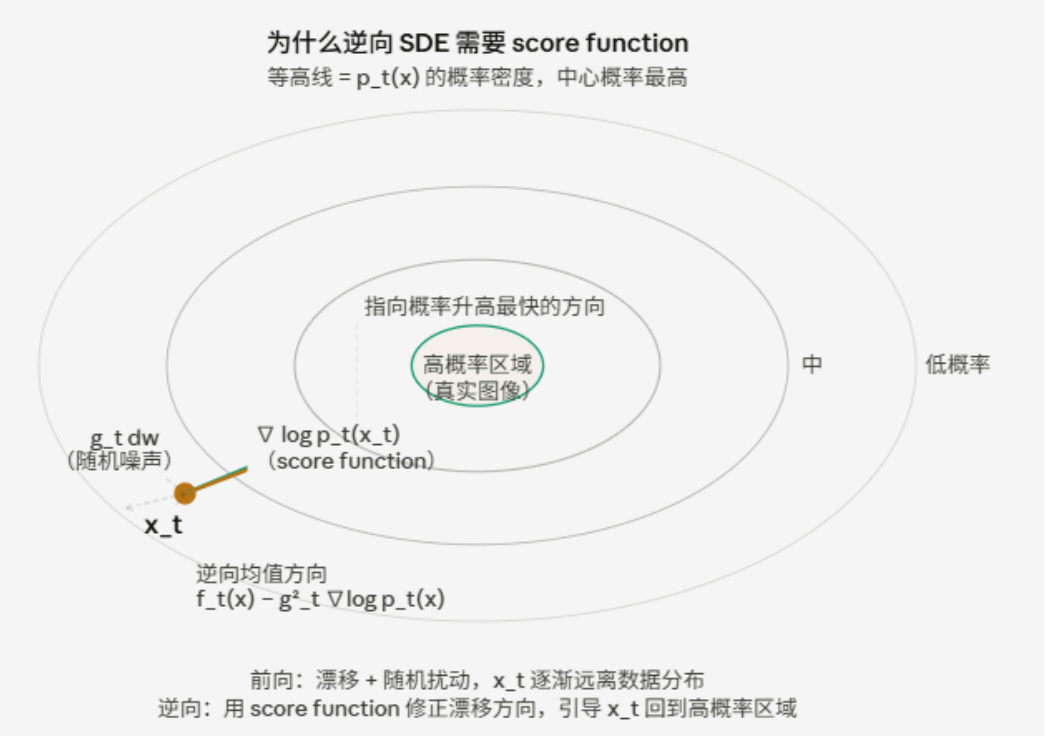
让模型去学一个$s_\theta(x_t, t) \approx \nabla_{x_t}\log p_t(x_t)$

具体的损失函数很简单：

$$Loss = \mathbb{E}_{\mathbf{x}_0,\mathbf{x}_t\sim p(\mathbf{x}_t|\mathbf{x}_0)\tilde{p}(\mathbf{x}_0)}\left[\|\mathbf{s}_\theta(\mathbf{x}_t, t) - \nabla_{\mathbf{x}_t}\log p(\mathbf{x}_t|\mathbf{x}_0)\|^2\right] \tag{35}$$

至此，我们又得到了一个很fancy的生成模型推导过程，但他对于前两者有啥优势呢：

1. DDPM 只是令 $\bar{\alpha}_t^2 + \bar{\beta}_t^2 = 1$ 时的一个特例，这里的f和g的调整会更灵活。更重要的是，连续化之后步数不再是超参数。
2. DDPM和DDIM的网络学习目标都是真实图片x_0（或者噪声），而Score-Based Model则是score function，即当前噪声级别下概率密度的梯度，这是一个有明确几何含义的量。这给后面的加速、rl等方向提供理论依据。



> **Tip**
>
> 可能细心的可以发现为啥这个sde建模里没有像DDIM一样有一个可以调节的σ_t，生成扩散模型漫谈（六）：一般框架之ODE篇 - 科学空间|Scientific Spaces这个博客里给出了比较详尽的讨论，篇幅限制就先不转述了。

5. Flow Matching：从离散到连续

前几节我们从 DDPM 出发，经历了**"离散 → 概率 → SDE"**这条路。每一步都在试图更优雅地描述同一件事：

- 如何从噪声走到数据。

但如果跳出来看，扩散模型绕了一个弯——它先定义了一个随机过程，再从这个过程里反推出生成轨迹。Score-Based 的路径不是我们选的，是 SDE 解出来的副产品。

这个"绕路"在小模型上看不出代价，但放到大规模训练里就有弊端：

1. 在 t 接近 1 时（几乎纯噪声），网络其实在学一个近乎不可能的去噪任务（从一张没有意义的噪声图预测另一张没有意义的噪声图），但这部分对梯度的贡献偏偏很大。
2. 在高分辨率 latent 上，前向加噪的"有效 SNR"在不同 t 上偏移得很厉害。对一张 1024×1024 的图加同样比例的噪声，比对 256×256 要"破坏得更彻底"，因为高频信息多得多。

那能不能干脆不绕这个弯？既然我们想要的就是"从噪声到数据"的一条路径，能不能跳过 SDE，**直接预测这条路径**？

这就是 Flow Matching 的出发点，也是目前 SD3、Flux、Qwen Image 等**主流模型采用的训练框架。**

回到 Score-Based 给出的概率流 ODE：

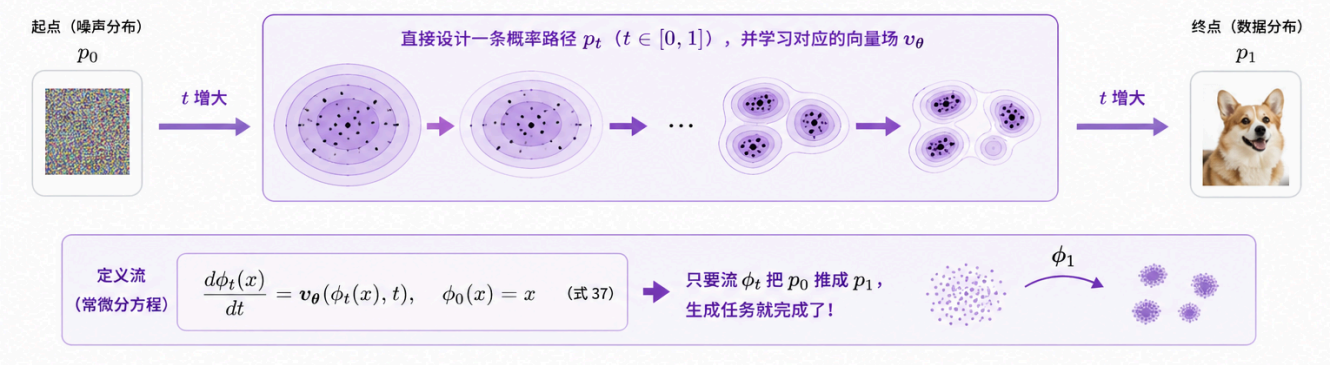
$$dx = \left[f_t(x) - \frac{g_t^2}{2} \nabla \log p_t(x) \right] dt \tag{36}$$

它告诉我们：只要有了 score function，就可以通过这个 ODE 把噪声变成数据。Score-Based 的做法是先学 score、再代回这个 ODE。但 ODE 本身的形式我们其实并不在乎——我们关心的只是它**能做的事**：把 p_0（噪声）传输成 p_1（数据）。

所以一个更直接的想法是：**跳过了 SDE 这一层中间结构，直接设计概率路径 p_t 和对应的向量场。**形式上，定义流 ϕ_t 满足：

$$\frac{d\phi_t(x)}{dt} = v_\theta(\phi_t(x), t), \quad \phi_0(x) = x \tag{37}$$

只要 ϕ 能把 p_0 推成 p_1，生成任务就完成了。



> 可以参考这个图简要理解这里的向量场是啥

那么如何训练模型拟合"真实向量场" u_t呢？比较容易想到的还是范数：

$$\mathcal{L} = \mathbb{E} * t, p_t(x)|v_\theta(x,t) - u_t(x)|^2 \tag{38}$$

不过我们还缺少每个时刻的$u_t(x)$，要精确计算的话依赖整个数据分布 q，这个cost是无法接受的。

我们还是借用DDPM 概率推导版本的技巧：**条件化。**

这其实可以算一个数值上的小技巧：

- **我向一个正确的方向优化一步，等效于我向正确方向的所有分量优化一步**

具体来说，不去想整体的概率路径，而是只考虑"从噪声走向某个特定数据点 x_1"的条件路径：

$$p_t(x|x_1) = \mathcal{N}(x; \mu_t(x_1), \sigma_t(x_1)^2 I) \tag{39}$$

数据集有多少张图，就有多少条条件路径，叠加起来就是等效的正确优化方向。

现在让我们形式化的推导Loss，因为已经规定了条件路径Eq. (39)，所有可以得到粒子的位置随时间的演化：

$$\psi_t(x_0|x_1) = \sigma_t(x_1)x_0 + \mu_t(x_1) \tag{40}$$

> 其中$x_0 \sim \mathcal{N}(0, I)$是标准噪声。

那么速度就是对Eq. (40)求个导：

$$u_t(x|x_1) = \frac{\sigma_t'(x_1)}{\sigma_t(x_1)}(x - \mu_t(x_1)) + \mu_t'(x_1) \tag{41}$$

现在每一项都能算了。定义 **Conditional Flow Matching (CFM) Loss**：

$$\mathcal{L}_{\text{CFM}}(\theta) = \mathbb{E}_{t\sim\mathcal{U}[0,1], x_1\sim q(x_1), x_0\sim p(x_0)} \|v_\theta(x_t,t) - u_t(x_t|x_1)\|^2 \tag{42}$$

到目前为止 μ_t, σ_t 都是设计自由度。

如果令 $\mu_t(x_1) = \bar{\alpha}_{1-t}x_1$、$\sigma_t = \bar{\beta}_{1-t}$，代入 Eq. (41) 推导出来的结果和 Score-Based 概率流 ODE 完全一致——所以**Score-Based 是 FM 的一个特例。**这种选择在 FM 框架下训练比 Score Matching 更稳定，因为绕开了 score 在 $t \to 0$ 时方差爆炸的问题。

但目前主流是更激进的**OT 路径**——均值方差都线性变化：

$$\mu_t(x_1) = t \cdot x_1, \quad \sigma_t = 1 - t \tag{43}$$

此时流映射就是仿射插值：

$$\psi_t(x_0|x_1) = (1-t)x_0 + tx_1 \tag{44}$$

代入 Eq. (41) 化简，条件向量场变成：

$$u_t(x|x_1) = \frac{x_1 - x}{1 - t} \tag{45}$$

进一步把这个值在采样点 $x_t = (1-t)x_0 + tx_1$ 上求值：

$$u_t(x_t|x_1) = x_1 - x_0 \tag{46}$$

所以最终 CFM 训练目标变得极其简洁：

$$\mathcal{L}_{\text{CFM}}(\theta) = \mathbb{E}_{t,x_0,x_1}\|v_\theta(x_t,t) - (x_1 - x_0)\|^2 \tag{47}$$

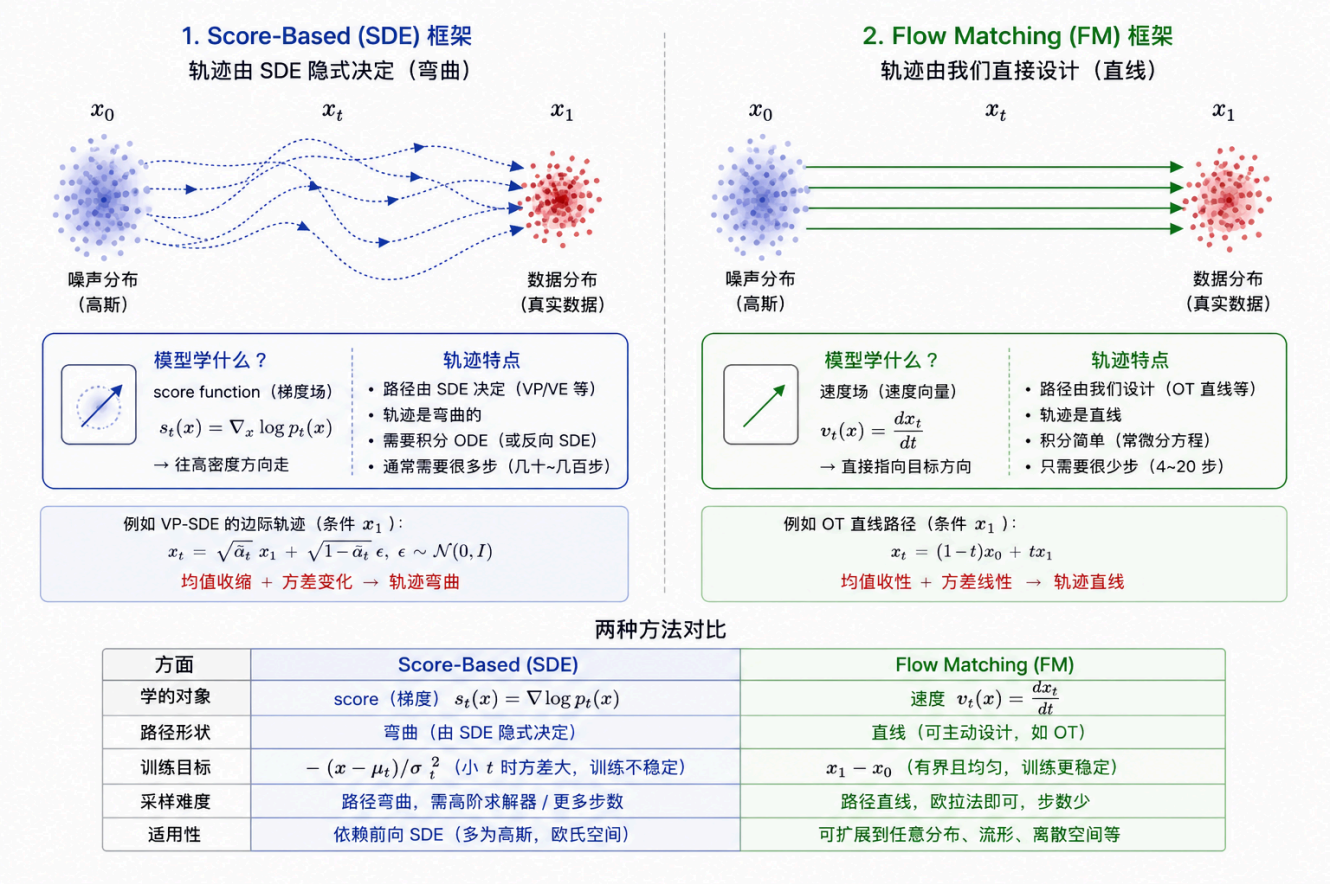
模型只需要预测一个常数方向，从 x_0 指向 x_1 的向量，这也是直觉上最快的方向

需要注意的一点是：FM 默认的求解过程是 ODE，是确定性的——同一个起点 x_0 一定生成同一个 x_1。

这看起来是个限制，但实际不构成问题，可以把 ODE 升级回 SDE：

$$dx = v_\theta(x,t)dt + \sigma_t dw \tag{48}$$

零噪声那条是概率流 ODE，加噪声重新得到 SDE 框架。



两种方法对比

方面	Score-Based (SDE)	Flow Matching (FM)
学的对象	score（梯度） $s_t(x) = \nabla \log p_t(x)$	速度 $v_t(x) = \frac{dx_t}{dt}$
路径形状	弯曲（由 SDE 隐式决定）	直线（可主动设计，如 OT）
训练目标	$-(x-\mu_t)/\sigma_t^2$（小 t 时方差大，训练不稳定）	$x_1 - x_0$（有界且均匀，训练更稳定）
采样难度	路径弯曲，需高阶求解器 / 更多步数	路径直线，欧拉法即可，步数少
适用性	依赖前向 SDE（多为高斯，欧氏空间）	可扩展到任意分布、流形、离散空间等

> 一个Flow matching的大致总结

至此，整体的建模脉络就从 DDPM 走到了 FM。简单来说，这条路是：**离散马尔可夫链 → 贝叶斯后验近似（DDPM）→ 去掉马尔可夫假设（DDIM）→ 连续 SDE + score 函数（Score-Based）→ 直接设计概率路径 + 条件向量场（Flow Matching）**。每一步都在放松某个限制，而 FM 目前是这条路上限制最少、工程上最简洁的终点。

比较常见的问题：

1. 如何生成一张text控制的图片

到目前为止我们一直在做**无条件生成**"，但实际想要的是"画一只在月球上的猫"这样的精确控制。

原理上的改动只有一处：**把网络从 $v_\theta(x_t,t)$ 改成 $v_\theta(x_t,t,c)$**，其中 c 是文本条件（一组从 CLIP/T5 等预训练 encoder 跑出来的 token embedding）。训练目标和 Eq. (47) 几乎一样：

$$\mathcal{L} = \mathbb{E}_{t,x_0,(x_1,c)\sim q}\|v_\theta(x_t,t,c) - (x_1 - x_0)\|^2 \tag{49}$$

但是在实际模型训练中，这不是简单的多一种输入的事情，需要有特定的设计把文本特征融合进去，这里就简单介绍两种text feature注入的方式：

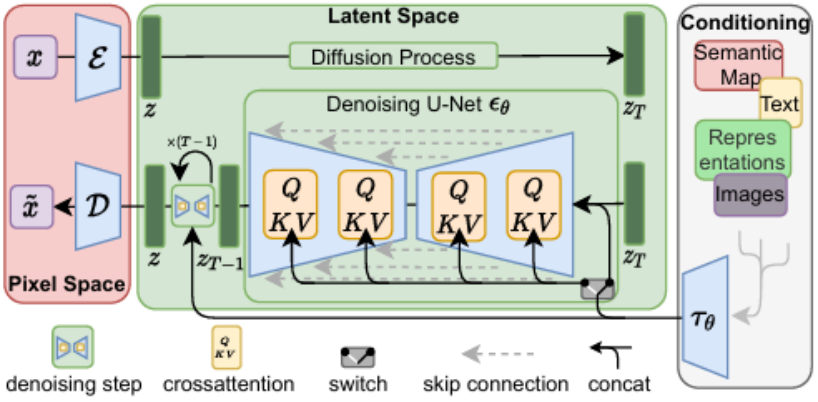
Cross-Attention

这个方法的出发点是，我能不能最小限度修改已有的模型，让它兼容文本的条件控制？

具体的做法就让整个网络的主干仍然处理图像，文本只作为"外部信息源"出现在 cross-attention 里：

$$Attn(Q,K,V)=\text{softmax}\left(\frac{QK^{\top}}{\sqrt{d_k}}\right)V \tag{50}$$

> 这里的Q来自图像，KV来自文本



可以参考这张LDM的图片，最右边就是文本注入的方式。

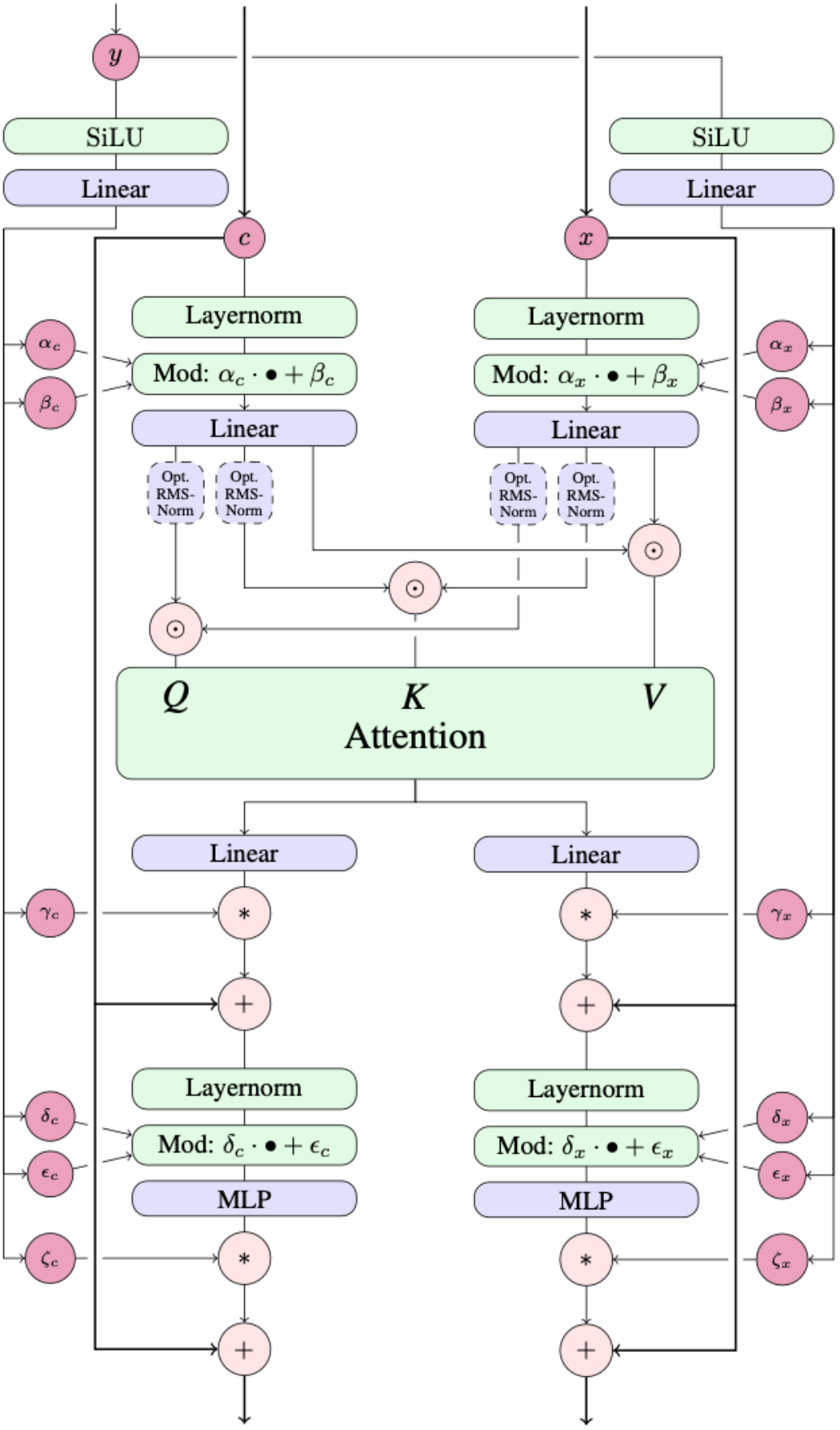
不过这个方法有两个问题。

一是**文本表征在去噪网络里是固定的**：CLIP/T5 编完之后，文本 embedding 在整个 UNet 里就再也不变了，只能被图像反复查询，不会基于当前 x_t 重新组织自己。SD3 论文把这一点总结为信息流动的"单向性"——图像可以查文本，但文本拿不到图像的反馈。

二是**文本和图像的特征空间不对齐**：文本住在 CLIP/T5 的语义空间，图像住在 VAE 的视觉空间，cross-attention 靠 W_K, W_V 把文本投到图像空间凑合用，但这只是个线性投影，对齐很粗糙。

所以需要一个更好的融合文本和图像特征的方法，就不得不从整体的架构入手了。

MMDiT



(b) One *MM-DiT* block

MMDiT的出发点很简单：

- **既然transformer在多模态和纯文本领域那么好用，我能不能把它也放到生成里来**

具体来说，就是把文本 token 和图像 token 拼到同一个 sequence 里，走 self-attention。

```
# 输入：
#   x_img: 图像 token, shape [B, N_img, D]
#   x_txt: 文本 token, shape [B, N_txt, D]
#   y:     条件向量 (timestep + pooled text), shape [B, D]

def MMDiT_Block(x_img, x_txt, y):
    # 每个模态从 y 投出自己的一套 modulation 参数
    α_img, β_img, γ_img, α'_img, β'_img, γ'_img = MLP_img(y)
    α_txt, β_txt, γ_txt, α'_txt, β'_txt, γ'_txt = MLP_txt(y)

    # ---- Joint Attention ----
    # 每个模态先做自己的 LayerNorm + modulation
    h_img = (1 + α_img) * LN(x_img) + β_img
    h_txt = (1 + α_txt) * LN(x_txt) + β_txt

    # 每个模态有自己的 QKV 投影矩阵 (注意权重是分开的)
    q_img, k_img, v_img = QKV_img(h_img)
    q_txt, k_txt, v_txt = QKV_txt(h_txt)

    # 在 sequence 维度上拼起来，做一次联合 self-attention
    Q = concat([q_img, q_txt], dim=seq)
    K = concat([k_img, k_txt], dim=seq)
    V = concat([v_img, v_txt], dim=seq)
    attn = softmax(Q @ K.T / sqrt(d)) @ V       # ← MMDiT 的核心

    # 拆回两路
    attn_img, attn_txt = split(attn, [N_img, N_txt])

    # 残差 + 各自的 output projection (gated)
    x_img = x_img + γ_img * Proj_img(attn_img)
    x_txt = x_txt + γ_txt * Proj_txt(attn_txt)

    # ---- FFN（也是各自一套）----
    x_img = x_img + γ'_img * FFN_img((1 + α'_img) * LN(x_img) + β'_img)
    x_txt = x_txt + γ'_txt * FFN_txt((1 + α'_txt) * LN(x_txt) + β'_txt)

    return x_img, x_txt
```

> 可以看这个伪代码

可以浅显的理解成文本和视觉特征都走各自的transformer block，只有再attention层才聚合下信息，这解决cross attenion的两个问题：

1. **文本表征在去噪网络里不再固定:** 视觉和文本的特征都会随着层数的加深不断改变
2. **文本和图像的特征空间对齐**: 仔细看这个框架其实文本和图像的地位是对等的，这两者能做双向混合，自然特征空间对齐的就更好了

2. cfg是什么

上的section讨论了如何从架构上加入文本指导，但单纯按 Eq. (49) 训出来的模型其实指令follow能力还不够好。让我们重新分析下公式：

Eq. (35) 让模型学的是条件 score $\nabla_x \log p(x|c)$（这里拿sde的那一套举例，换成FM是一样的，改成u就可以），这里的c代表的是所**有训练集的文本条件**，自然是比较嘈杂的，有没有方法特别突出其中一条的文本条件呢?

先把条件概率拆开：

$$\log p(\mathbf{x}|\mathbf{c}) = \log p(\mathbf{x}) + \log p(\mathbf{c}|\mathbf{x}) - \log p(\mathbf{c}) \tag{51}$$

最后一项是常数项，求导之后可得：

$$\underbrace{\nabla_{\mathbf{x}} \log p(\mathbf{x}|\mathbf{c})}_{\text{条件 score}} = \underbrace{\nabla_{\mathbf{x}} \log p(\mathbf{x})}_{\text{无条件 score}} + \underbrace{\nabla_{\mathbf{x}} \log p(\mathbf{c}|\mathbf{x})}_{\text{分类器导向 (Guidance)}} \tag{52}$$

如果想加强分类器的引导，是不是只需要在$\log p(\mathbf{c}|\mathbf{x})$前面加个大于1的系数就好了：

$$\nabla_{\mathbf{x}} \log p_w(\mathbf{x}|\mathbf{c}) = \nabla_{\mathbf{x}} \log p(\mathbf{x}) + (1+w)\nabla_{\mathbf{x}} \log p(\mathbf{c}|\mathbf{x}) \tag{53}$$

等价的目标分布表达

$$p_w(\mathbf{x}|\mathbf{c}) \propto p(\mathbf{x}) \cdot p(\mathbf{c}|\mathbf{x})^{1+w} \tag{54}$$

不过这里的$\log p(\mathbf{c}|\mathbf{x})$没有模型没有学过怎么办，其实Eq. (52)已经给出了等效的替换等式：

$$\underbrace{\nabla_{\mathbf{x}} \log p(\mathbf{c}|\mathbf{x})}_{\text{分类器梯度 (牵引力)}} = \underbrace{\nabla_{\mathbf{x}} \log p(\mathbf{x}|\mathbf{c})}_{\text{有条件 score}} - \underbrace{\nabla_{\mathbf{x}} \log p(\mathbf{x})}_{\text{无条件 score}} \tag{55}$$

右边两项都是扩散模型自己能学的——**前者是带条件的网络，后者是不带条件的网络。**

再带入Eq. (53)

$$\tilde{v}_\theta(\mathbf{x}_t, t, \mathbf{c}) = (1+w) \cdot v_\theta(\mathbf{x}_t, t, \mathbf{c}) - w \cdot v_\theta(\mathbf{x}_t, t, \emptyset) \tag{56}$$

所以我们在推理的时候，只需要跑一遍带条件的网络，再跑一遍不带条件的网络就ok了，完全不用重新训练模型就能得到更好的instruction following能力。

Figure 2: The effect of guidance on a mixture of three Gaussians, each mixture component representing data conditioned on a class. The leftmost plot is the non-guided marginal density. Left to right are densities of mixtures of normalized guided conditionals with increasing guidance strength.

> Guidence不断强化之后的数据分布

其实cfg到这里还没结束：它有个小缺点，就是inference的时候需要跑两遍，这样cost也会翻倍，所以如果对inference cost很敏感的话，我们完全可以训一个单次前向的学生模型 v_ϕ，让它直接输出 CFG 之后的结果：

$$v_\phi(x_t, t, c) \approx \tilde{v}_\theta(x_t, t, c) = (1+w) \cdot v_\theta(x_t, t, c) - w \cdot v_\theta(x_t, t, \emptyset) \tag{57}$$

这样训练出来的模型就自动把w硬编码进了权重，或者：

$$v_\phi(x_t, t, c, w) \approx \tilde{v}_\theta(x_t, t, c) = (1+w) \cdot v_\theta(x_t, t, c) - w \cdot v_\theta(x_t, t, \emptyset) \tag{58}$$

把w当成一个超参，这样模型学到的是整条 w 轴上的映射，推理时 w 重新变回可调超参。

3. 为什么inference的步数和训练不一致

这个问题其实在我们第一个Chapter里已经回答的差不多了，这里来汇总一下：

首先，为什么会有这个问题，因为我们在最开始推导DDPM的时候（概率版本），是通过贝叶斯推出来的。这个时候是不能跳步的。

而在DDIM的时候去掉了马尔可夫假设，整个过程只依赖两个边际分布 $p(x_t|x_0)$ 和 $p(x_{t-1}|x_0)$。Eq. (27) 里的 $\sigma_t = 0$ 那个特例对应的就是确定性少步采样。

在SDE/FM连续化之后，已经没有步数这个概念了，只需要控制采样的精度。推理时步数完全是 ODE/SDE solver 的选择。至于为什么是多步训练少步推理，一般认为有两个原因：

1. 推理效率
2. 模型泛化能力

至于怎么把推理步数从几十步压到几步、甚至 1 步，那是另一个故事——见后面的"少步生成与加速"。

研究方向

> 前面两个chapter梳理生成模型的脉络，下面就对几个相关的比较有意思的方向做些简单介绍
>
> 因为每个方向的方法很多，很难一次说清，这里就挑些有代表性的工作介绍一下

1. 少步生成与加速

Consistency Models

如果我们直接使用flow matching那一套做少步甚至单步生成，会有什么问题呢?

回想一下 FM 学的是**瞬时速度场** $v_\theta(x_t, t)$。生成时要解一个 ODE (Eq. 37):

最朴素的解法是欧拉法，在0->1的时间范围内采样N次，根据每次的速度步进一小步。N 越大，每步走得越短，离散化误差越小，结果越好。

$$x_{t_{k-1}} = x_{t_k} + (t_{k-1} - t_k) \cdot v_\theta(x_{t_k}, t_k) \tag{57}$$

那如果我们想 1 步搞定（$N = 1$）呢?

$$x_0 = x_T - T \cdot v_\theta(x_T, T) \tag{58}$$

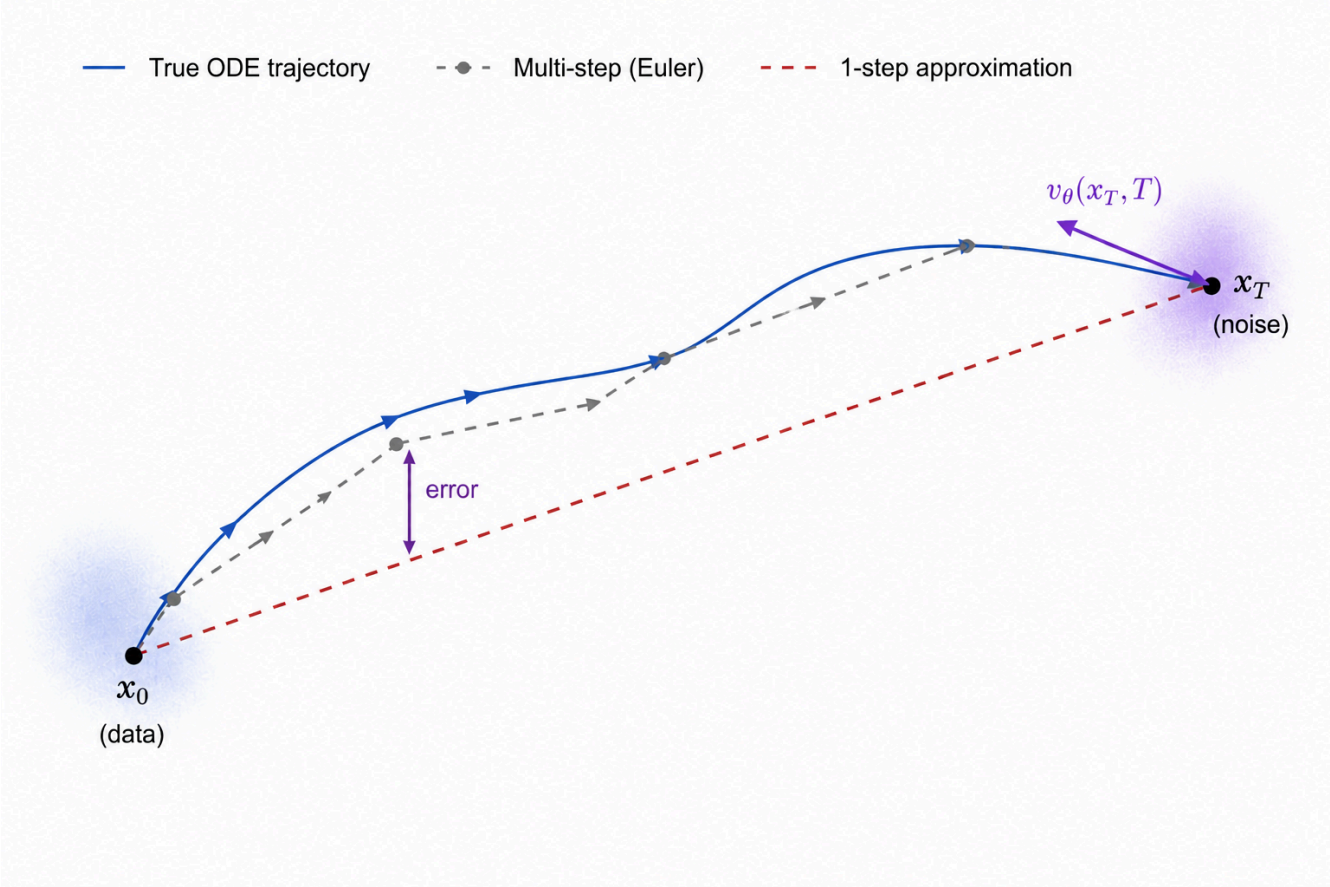
它变成了只依赖于初始位置的瞬时速度，直接拿这个方向乘以总时长 T 当位移用，相当于假设整条 ODE 是一条直线，必然会引入误差。

> **Caution**
>
> 这里可能会有疑问，我们之前在推导FM的时候，不已经默认ODE和sde那一套相比，是一条最快的直线了吗。但其实OT 路径下边缘速度场积分出来的实际轨迹也是弯的，可以参考：
>
> [2209.03003] Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow

那如果我们让网络直接学全局信息呢？也就是说，不让它预测"现在该往哪走"，而是让它直接告诉你"最终会到哪里"：

$$f(x_t, t) \to x_0 \tag{59}$$

这样每一步都consistent向同一个目标优化——最终的图片分布x_0。



最直觉的想法是直接监督，让 $f_\theta(\mathbf{x}_t, t)$ 回归真实的 $\mathbf{x}_0$，Loss就相应的变成了：

$$\mathcal{L}_{\mathcal{CD}} = \mathbb{E}_{t,x_0,x_1} \| f_\theta(x_t, t) - x_0 \|^2 \tag{60}$$

这会带来一个我们在Flow matching Section介绍过的问题:

- **在去噪过程的早期，x_t 几乎是纯噪声，给定同一个 $\mathbf{x}_t$，对应的 $\mathbf{x}_0$ 有无数种可能**

这样训练在t较大的时候必定是无效的，所以Consistency Models的做法是把绝对回归目标换成局部相对约束:

- **要求同一条 ODE 轨迹上的相邻两点，经过 f_θ 之后映射到同一个结果。**

根据Eq. (58)，我们可以写出如下的局部相对约束Loss:

$$\mathcal{L}_{\mathcal{CD}} = \mathbb{E} \left\| \underbrace{x_{t_{k+1}} - t_{k+1} \cdot v_\theta(x_{t_{k+1}}, t_{k+1})}_{\text{从 } t_{k+1} \text{ 预测的 } x_0} - \underbrace{(\hat{x}_{t_k} - t_k \cdot v_{\theta^-}(x_{t_k}, t_k))}_{\text{走一步后，从 } t_k \text{ 预测的 } x_0} \right\| \tag{61}$$

> 其中t_k的推导服从Eq. (57)

> **Tip**
>
> 在loss右边出现了一个没见过的v_{θ^-}，这其实是一个小trick，为了训练的稳定性。v_{θ^-}不参与梯度更新，类似是通过EMA的形式维护了一个旧policy，更新公式为:
>
> $$\theta^- \leftarrow \mu\theta^- + (1-\mu)\theta \tag{62}$$

如此，我们就训练出来一个从轨迹上任意一点直接预测 $\mathbf{x}_0$的网络，推理的时候只需要用Eq. (58)就能一步得到结果

(c) CD with two-step generation (FID=2.93)

> CM论文中一步生图的case

MeanFlow

尽管CM在少步生成上得到了成功，但训练上始终有一个说不清的地方：loss 的 target 来自 v_{θ^-}，而 θ^- 本身又是 θ 的 EMA（**网络在追一个由自己衍生出来的目标**）。接下来介绍的这篇MeanFlow则从v的拟合目标触发，给训练提供了一个有明确 ground-truth 的训练目标。

让我们重新回到Eq. (47)，我希望生成步数少，就是希望$(t_{k-1} - t_k)$的值大，但是反观我们训练的目标，一直是在优化瞬时速度v_θ。这就导致了我们一直在已有的框架上打补丁，让瞬时速度尽可能的能拟合跨步较大的真实速度。

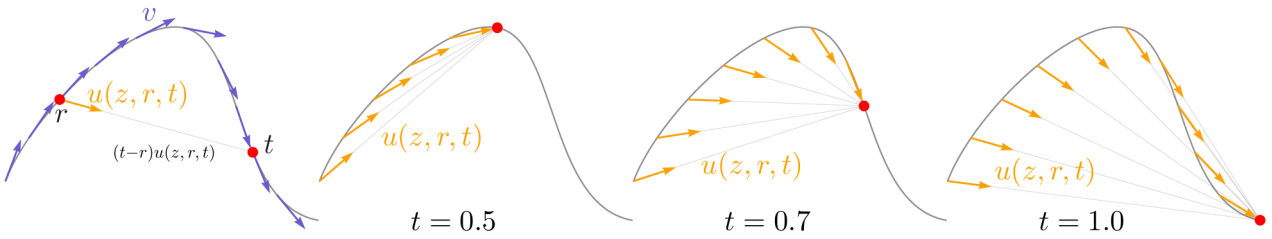


Figure 3: **The field of *average velocity*** $u(z,r,t)$. **Leftmost**: While the *instantaneous velocity* v determines the tangent direction of the path, the average velocity $u(z,r,t)$, defined in Eq. (3), is generally not aligned with v. The average velocity is aligned with the *displacement*, which is $(t-r)u(z,r,t)$. **Right three subplots**: The field $u(z,r,t)$ is conditioned on both r and t, and is shown here for $t=0.5$, 0.7, and 1.0.

> 直观上理解在velocity field上瞬时和平均的区别

自然，从直觉上讲，更适合这个任务目标的是平均速度，我们就从这个motivation出发，尝试构建一个**拟合到目标的平均速度**的生成模型

从Eq. (47)出发，我们原本是需要解一个如下ODE：

$$\frac{dx_t}{dt} = v_\theta(x_t, t) \tag{63}$$

把瞬时速度变成平均速度，其实只需要在两边都做下积分：

$$x_t - x_r = \int_r^t v_\theta(x_\tau,\tau)d\tau = (t-r) \times \frac{1}{t-r}\int_r^t v_\theta(x_\tau,\tau)d\tau \tag{64}$$

再化简下，左边就变成t到r的平均速度了：

$$u_\theta(x_t, r, t) = \frac{1}{t-r}\int_r^t v_\theta(x_\tau,\tau)d\tau \tag{65}$$

现在要做的事情其实已经比较明显了，因为在原来的损失函数中，我们对v_θ进行优化，**只要我们把v_θ用u_θ进行替换，自然也就得到了针对平均速度的优化目标。**

既然如此，把Eq. (65)化简之后得到：

$$\begin{aligned}\boldsymbol{v}_\theta(\boldsymbol{x}_t, t) &= \boldsymbol{u}_\theta(\boldsymbol{x}_t, r, t) + (t-r)\frac{d}{dt}\boldsymbol{u}_\theta(\boldsymbol{x}_t, r, t) \\ &= \boldsymbol{u}_\theta(\boldsymbol{x}_t, r, t) + (t-r)\left[\frac{d\boldsymbol{x}_t}{dt}\cdot\frac{\partial}{\partial \boldsymbol{x}_t}\boldsymbol{u}_\theta(\boldsymbol{x}_t, r, t) + \frac{\partial}{\partial t}\boldsymbol{u}_\theta(\boldsymbol{x}_t, r, t)\right]\end{aligned} \tag{66}$$

现在的式子右边，我们只剩下一个$\frac{d\boldsymbol{x}_t}{dt}$不知道了，但好巧不巧这个其实相当于t时刻的瞬时速度，也就是t时刻到t时刻的平均速度$\boldsymbol{u}_\theta(\boldsymbol{x}_t, t, t)$，带入Eq. (47)可得:

$$\mathbb{E}_{r,t,\boldsymbol{x}_0,\boldsymbol{x}_1}\left[\left\|\boldsymbol{u}_\theta(\boldsymbol{x}_t, r, t) + (t-r)\left[\boldsymbol{u}_\theta(\boldsymbol{x}_t, t, t)\cdot\frac{\partial}{\partial \boldsymbol{x}_t}\boldsymbol{u}_\theta(\boldsymbol{x}_t, r, t) + \frac{\partial}{\partial t}\boldsymbol{u}_\theta(\boldsymbol{x}_t, r, t)\right] - (\boldsymbol{x}_1 - \boldsymbol{x}_0)\right\|^2\right] \tag{67}$$

> 可能的疑问是为啥这里不直接换成$\boldsymbol{u}_\theta(\boldsymbol{x}_t, t, t) - (x_1 - x_0)$的格式，这样就不存在时间窗口长度不为零的情况了，模型也自然无法从训练数据里学习到这种情况，所以这样只是形式上把他变成了平均速度，实际上和瞬时速度没有任何区别。

理论来说，我们完全可以用Eq. (67)来训练模型，不过实际上为了减少训练时候的运算量，会选择下面这种训练目标，会少一次forward的cost （少算一次u）

$$\mathbb{E}_{r,t,\boldsymbol{x}_0,\boldsymbol{x}_1}\left[\left\|\boldsymbol{u}_\theta(\boldsymbol{x}_t, r, t) + (t-r)\text{sg}\left[(\boldsymbol{x}_1 - \boldsymbol{x}_0)\cdot\frac{\partial}{\partial \boldsymbol{x}_t}\boldsymbol{u}_\theta(\boldsymbol{x}_t, r, t) + \frac{\partial}{\partial t}\boldsymbol{u}_\theta(\boldsymbol{x}_t, r, t)\right] - (\boldsymbol{x}_1 - \boldsymbol{x}_0)\right\|^2\right] \tag{68}$$

Figure 5: **1-NFE Generation Results**. We show curated examples of class-conditional generation on ImageNet 256×256 using our 1-NFE model (MeanFlow-XL/2, 3.43 FID).

> meanflow 1-NFE（Number of Function Evaluations）的结果

> **Note**
>
> 其实刚刚说的两个工作都可以划分为在 ODE 轨迹上施加自洽约束，常见的加速方法还包括但不限于：
>
> 1. 对ODE/SDE运算的加速
>
>   > [2206.00927] DPM-Solver: A Fast ODE Solver for Diffusion Probabilistic Model Sampling in Around 10 Steps
>
> 2. 蒸馏加速
>
>   > [2404.04057] Score identity Distillation: Exponentially Fast Distillation of Pretrained Diffusion Models for One-Step Generation
>
> 等方向，因为近期关注度并不那么高就不展开了。

2. 图像生成与强化学习

> 这里就各选一个on-policy和off-policy的方法来做介绍

Diffusion DPO （off-policy）

既然要讲Diffusion DPO，就不可避免的需要介绍经典DPO；所以在正式开始之前，先简单的把DPO算法过一遍

> DPO (Direct Preference Optimization) 算法讲解_哔哩哔哩_bilibili（推导过程参考了这个视频，讲的很清晰）

- 为什么需要DPO：

  在post training中，DPO可以算是比较“轻量级”的一种方法了，他不需要reward model、verifier、value model。只需要一好一坏的pair data就能训起来。

他的具体公式是这样的：

$$\max_{\pi} \underbrace{\mathbb{E}_{x\sim D, y\sim\pi}[r(x,y)]}_{\text{得到尽可能多的奖励}} - \underbrace{\beta\mathbb{D}_{KL}[\pi(y|x)||\pi_{ref}(y|x)]}_{\text{新模型尽可能地和基准模型分布一致}} \tag{69}$$

其中：

> - **奖励函数**：$r(x,y)$（其中 x 为 prompt，y 为 response）
> - **基准模型**：$\pi_{ref}(y|x)$
> - **训练模型**：$\pi(y|x)$

直观上理解，就是要做到两个事情：

1. 让模型的策略往生成更好结果的方向走（得到更多reward）

2. 让模型不要变太多，导致崩溃和hacking（KL项）

> 可能会问，不是说没有reward model吗，这里的优化项为什么还有函数r（更好的结果会得到更高的分数），这是因为这个公式还没完全化简，可以在后面慢慢解出r的表达形式

首先把KL散度展开一下

$$\max_{\pi} \mathbb{E}_{x\sim D}\left[\sum_{y}\pi(y|x)r(x,y)-\beta\sum_{y}\pi(y|x)\log\frac{\pi(y|x)}{\pi_{ref}(y|x)}\right] \tag{70}$$

把求和变成期望：

$$\max_{\pi} \mathbb{E}_{x\sim D,y\sim\pi}[r(x,y)]-\mathbb{E}_{x\sim D,y\sim\pi}\left[\beta\log\frac{\pi(y|x)}{\pi_{ref}(y|x)}\right] \tag{71}$$

整理一下可以得到：

$$\min_{\pi} \mathbb{E}_{x\sim D,y\sim\pi}\left[\log\frac{\pi(y|x)}{\pi_{ref}(y|x)}-\frac{1}{\beta}r(x,y)\right] \tag{72}$$

再把后面一项放到log里面：

$$=\min_{\pi} \mathbb{E}_{x\sim D,y\sim\pi}\left[\log\frac{\pi(y|x)}{\pi_{ref}(y|x)\exp\left(\frac{1}{\beta}r(x,y)\right)}\right] \tag{73}$$

下面会用一个小trick，如果了解的话这在Soft Q-Learning里也出现过：

> Reinforcement Learning with Deep Energy-Based Policies

$$\begin{cases}\min_{\pi} \mathbb{E}_{x\sim D,y\sim\pi}\left[\log\frac{\pi(y|x)}{\pi_{ref}(y|x)\exp\left(\frac{1}{\beta}r(x,y)\right)\frac{1}{Z(x)}Z(x)}\right] \\ Z(x)=\sum_{y}\pi_{ref}(y|x)\exp\left(\frac{1}{\beta}r(x,y)\right)\end{cases} \tag{74}$$

这里添上的$Z(x)$做了两个事情：

1. 把log变成了两个概率分布的比值
2. 添加了一个和期望无关的，只和x有关的附加项

所以不那么准确的说，这一步把优化目标变成了：**两个分布的KL散度+无关常数项**

不妨令：

$$\pi^*(y|x)=\frac{1}{Z(x)}\pi_{ref}(y|x)\exp\left(\frac{1}{\beta}r(x,y)\right)=\frac{\pi_{ref}(y|x)\exp\left(\frac{1}{\beta}r(x,y)\right)}{\sum_{y}\pi_{ref}(y|x)\exp\left(\frac{1}{\beta}r(x,y)\right)} \tag{75}$$

进一步化简就看到很清晰的：

$$=\min_{\pi} \mathbb{E}_{x\sim D,y\sim\pi}\left[\log\frac{\pi(y|x)}{\pi^*(y|x)}-\log Z(x)\right] \tag{76}$$

$$=\min_{\pi} \mathbb{D}_{KL}(\pi(y|x)||\pi^*(y|x))\Rightarrow \boldsymbol{\pi(y|x)=\pi^*(y|x)=\frac{1}{Z(x)}\pi_{ref}(y|x)\exp\left(\frac{1}{\beta}r(x,y)\right)} \tag{77}$$

众所周知KL散度当且仅当两个分布重合的时候取值最小，所以可以把最小化KL变成俩分布相等，化简可得$r(x,y)$的表达式：

$$r(x,y)=\beta\ln\frac{\pi(y|x)}{\pi_{ref}(y|x)}+\beta\ln Z(x) \tag{78}$$

那么好策略和坏策略之间的差值就是：

$$r(x,y_w)-r(x,y_l)=\beta\ln\frac{\pi(y_w|x)}{\pi_{ref}(y_w|x)}-\beta\ln\frac{\pi(y_l|x)}{\pi_{ref}(y_l|x)} \tag{79}$$

这也就是为啥我们需要Pair Data来训练模型。

现在离找到训练目标只差一步了：**我如何让模型学会最大化这俩策略的差别。**

Bradley-Terry(BT) 模型

先看一个简单的例子：

假设 A、B、C 三个选手互相对战，结果如下：

对战	胜	负
A 对 B	8	4
A 对 C	3	5

我们想根据这些对战记录，估计每个选手的"实力"。BT 模型给出了一个非常简洁的假设：用一个正数 α_i 表示第 i 个元素的实力（必须大于 0），那么 i 战胜 j 的概率为：

$$P(i > j) = \frac{\alpha_i}{\alpha_i + \alpha_j} \tag{80}$$

这个式子的直觉非常自然：两个人对战，谁实力占的比例大，谁赢的概率就大。

有了概率模型，就可以用**对数最大似然估计**来反推 $\alpha_A, \alpha_B, \alpha_C$。把上面的对战记录代入：

$$\ln L = 8 \ln\left(\frac{\alpha_A}{\alpha_A + \alpha_B}\right) + 4 \ln\left(\frac{\alpha_B}{\alpha_A + \alpha_B}\right) + 3 \ln\left(\frac{\alpha_A}{\alpha_A + \alpha_C}\right) + 5 \ln\left(\frac{\alpha_C}{\alpha_A + \alpha_C}\right) \tag{81}$$

最大化这个似然，就能解出每个选手的相对实力。把它写成一般的 Loss 函数（取负号变成最小化，对数据分布取期望）：

$$\text{Loss} = -\mathbb{E}_{(\alpha_x, \alpha_y) \sim D}\left[\ln \frac{\alpha_x}{\alpha_x + \alpha_y}\right] \tag{82}$$

现在把 BT 模型搬到 DPO 的语境里。回答的"好坏"（也就是上面 BT 模型里的"实力"）是靠 Reward 模型 $r(x, y)$ 来评估的。直接代入 BT 公式：

$$P(y_1 \succ y_2) = \frac{r(x, y_1)}{r(x, y_1) + r(x, y_2)} \tag{83}$$

但有个小问题：BT 模型要求"实力"是正数，而 $r(x, y)$ 是神经网络的输出，**有可能返回负数。**最简单的办法就是套一个指数函数，强行变成正数：

$$P(y_1 \succ y_2) = \frac{\exp(r(x, y_1))}{\exp(r(x, y_1)) + \exp(r(x, y_2))} \tag{84}$$

###

再回忆一下 sigmoid 函数的定义：

$$\sigma(x) = \frac{1}{1 + \exp(-x)} \tag{85}$$

把Eq. (84) 做几步化简，就可以得到一个十分优雅的形式：

$$-\ln \sigma\big(r(x, y_w) - r(x, y_l)\big) \tag{86}$$

这就是我们要的训练目标——它告诉模型：**最大化两个策略之间的奖励差**，差越大，sigmoid 越接近 1，$\ln \sigma$ 越接近 0，loss 越小。

于是DPO的最终Loss函数可以写成：

$$-\ln \sigma\big(\beta \ln \frac{\pi(y_w|x)}{\pi_{ref}(y_w|x)} - \beta \ln \frac{\pi(y_l|x)}{\pi_{ref}(y_l|x)}\big) \tag{87}$$

从 DPO 到 Diffusion-DPO

现在轮到 Diffusion-DPO 登场了。问题非常自然：上面这套东西能不能直接搬到扩散模型上？

在DDPM语境下，Eq. (87)的y_w是生成出来的干净图x_0，条件x则是所有的训练集c，那么我们的Loss就可以写成:

$$Loss = -\ln\sigma\big(\beta\ln\frac{p(x_0^w|c)}{p_{ref}(x_0^w|c)} - \beta\ln\frac{p(x_0^{nl}|c)}{p_{ref}(x_0^l|c)}\big) \tag{88}$$

但是问题就出在我们这里引入的$p(x_0)$这里，试想DDPM是怎么得到最终图片的：

它从纯噪声 x_T 开始，一步步去噪，$x_T \to x_{T-1} \to \cdots \to x_0$，假设需要 1000 步。每一步 $p_\theta(x_{t-1}|x_t)$ 都是个简单的高斯，所以整条路径的联合概率是：

$$p_\theta(x_{0:T}|c) = p_\theta(x_T)\prod_{t=1}^{T} p_\theta(x_{t-1}|x_t, c) \tag{89}$$

但我们要的是 x_0 的边际似然——也就是不管中间走的是哪条路径，最终生成 x_0 的概率。这需要把所有中间步骤积掉：

$$p_\theta(x_0|c) = \int p_\theta(x_{0:T}|c)dx_{1:T} \tag{90}$$

这个 1000 维的积分，没法解析算，也没法数值近似。

所以不妨转换下思路：**每个策略最后采样出的不是一张干净的图，而是一条能最终变成干净图的轨迹**

这样公式就变成了：

$$Loss = -\ln\sigma\big(\beta\ln\frac{p(x_{0:T}^w|c)}{p_{ref}(x_{0:T}^w|c)} - \beta\ln\frac{p(x_{0:T}^l|c)}{p_{ref}(x_{0:T}^l|c)}\big) \tag{91}$$

把ln里面的轨迹变成条件概率相乘，得到：

$$Loss = -\mathbb{E}_{x_0^w,x_0^l}\left[\ln\sigma\left(\beta\mathbb{E}_{x_{1:T}^w,x_{1:T}^l\sim p_\theta}\left[\sum_{t=1}^{T}\ln\frac{p_\theta(x_{t-1}^w|x_t^w)}{p_{\text{ref}}(x_{t-1}^w|x_t^w)} - \sum_{t=1}^{T}\ln\frac{p_\theta(x_{t-1}^l|x_t^l)}{p_{\text{ref}}(x_{t-1}^l|x_t^l)}\right]\right)\right] \tag{92}$$

这个loss的问题是太不灵活了，我要得到一个loss，需要完整的计算出T个$p_\theta(x_{t-1}^w|x_t^w)$，这从计算效率上来说是不可行的。我们希望的是，有任意一步$p_\theta(x_{t-1}^w|x_t^w)$，都能给出一个梯度信号来训练模型。

一个自然的想法就是把Eq. (92)的$\sum_{t=1}^{T}$变成$T*\mathbb{E}()$的形式，由于−logσ(·) 函数是凸函数，直接用Jensen不等式：

$$-\log\sigma\left(\mathbb{E}[f]\right) \le \mathbb{E}\left[-\log\sigma(f)\right] \tag{93}$$

带入Eq. (92)，得到化简之后的损失函数：

$$\mathcal{L} \le -\mathbb{E}_{x_0^w,x_0^l,t}\left[\ln\sigma\left(\beta T\left(\mathbb{E}_{x_{t-1}^w,x_t^w\sim p_\theta}\ln\frac{p_\theta(x_{t-1}^w|x_t^w)}{p_{\text{ref}}(x_{t-1}^w|x_t^w)} - \mathbb{E}_{x_{t-1}^l,x_t^l\sim p_\theta}\ln\frac{p_\theta(x_{t-1}^l|x_t^l)}{p_{\text{ref}}(x_{t-1}^l|x_t^l)}\right)\right)\right] \tag{94}$$

式 (94) 虽然解决了"必须把 T 项全算出来才能给梯度"的问题,但**新的瓶颈立刻冒出来——在内层期望上。**

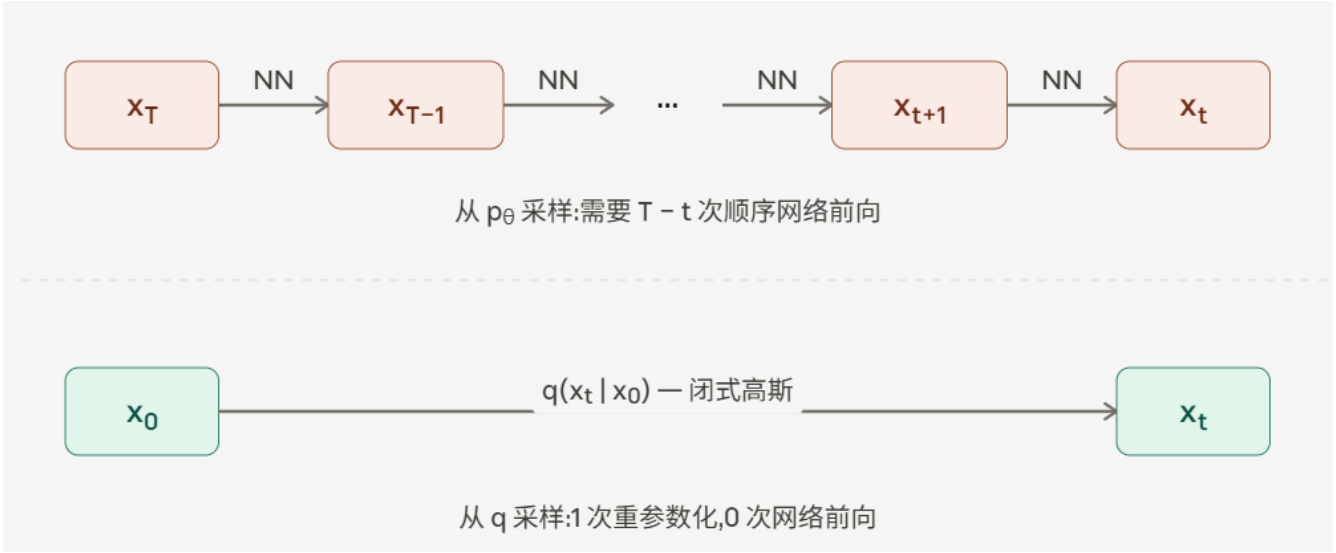
注意 (94) 里面的 $\mathbb{E}_{x_{t-1}^w,x_t^w\sim p_\theta}$:这个期望是相对于**模型自己的反向过程** $p_\theta(x_{t-1}, x_t \mid x_0^w)$ 取的。要从这个分布拿到一个 (x_{t-1}, x_t) 样本,你仍然得:

1. 从 $x_T \sim \mathcal{N}(0, I)$ 出发,
2. 一步步跑反向链 $x_T \to x_{T-1} \to \cdots \to x_t$,

中间每一步都是一次网络前向。也就是说——**Jensen 把损失"摊平"到了某个单一 t 上,但要拿到那个 t 处的 (x_{t-1}, x_t) 仍然需要 $O(T-t)$ 次前向**,平均下来还是 $O(T)$。采样开销并没被解决,只是从"求和"换了个地方藏到"内层期望"里。

所以 Step 3 的关键近似就呼之欲出了:**把内层的 $\mathbb{E}_{\sim p_\theta(\cdot|x_0)}$ 替换成 $\mathbb{E}_{\sim p(\cdot|x_0)}$**——也就是用预定义的前向过程代替模型的反向过程作为采样分布。

> 其实这样能work很重要的一个前提是DPO是post training方法，一个良好的起始ckpt的$p_\theta(\cdot \mid x_0)$是和$p(\cdot \mid x_0)$高度近似的。



> 这里的q指的是前向过程

再回头看Eq. (17)，可以很清晰的看出前向过程可以直接计算出：

$$p(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0) = \frac{p(\mathbf{x}_t|\mathbf{x}_{t-1},\mathbf{x}_0)p(\mathbf{x}_{t-1}|\mathbf{x}_0)}{p(\mathbf{x}_t|\mathbf{x}_0)} = \mathcal{N}\left(\mathbf{x}_{t-1}; \underbrace{\frac{\alpha_t\bar{\beta}_{t-1}^2}{\bar{\beta}_t^2}\mathbf{x}_t + \frac{\bar{\alpha}_{t-1}\beta_t^2}{\bar{\beta}_t^2}\mathbf{x}_0}_{\boldsymbol{\mu}_q(\mathbf{x}_t,\mathbf{x}_0)}, \underbrace{\frac{\bar{\beta}_{t-1}^2\beta_t^2}{\bar{\beta}_t^2}\mathbf{I}}_{\sigma_q^2(t)}\right) \tag{17}$$

这样损失函数进一步变成了：

$$\mathcal{L} \leq -\mathbb{E}_{x_0^w,x_0^l,t}\left[\ln\sigma\left(\beta T\left(\mathbb{E}_{x_{t-1}^w,x_t^w\sim p}\ln\frac{p_\theta(x_{t-1}^w|x_t^w)}{p_{\text{ref}}(x_{t-1}^w|x_t^w)} - \mathbb{E}_{x_{t-1}^l,x_t^l\sim p}\ln\frac{p_\theta(x_{t-1}^l|x_t^l)}{p_{\text{ref}}(x_{t-1}^l|x_t^l)}\right)\right)\right] \tag{95}$$

> **Important**
>
> 这里有个很有意思的想法，可以思考下：既然我都用前向过程来近似反向过程了，为啥不直接彻底一点把Loss变成这个形式呢，这样就彻底的可以直接计算出了：
>
> $$\mathcal{L} \leq -\mathbb{E}_{x_0^w,x_0^l,t}\left[\ln\sigma\left(\beta T\left(\mathbb{E}_{x_{t-1}^w,x_t^w\sim p}\ln\frac{p(x_{t-1}^w|x_t^w)}{p_{\text{ref}}(x_{t-1}^w|x_t^w)} - \mathbb{E}_{x_{t-1}^l,x_t^l\sim p}\ln\frac{p(x_{t-1}^l|x_t^l)}{p_{\text{ref}}(x_{t-1}^l|x_t^l)}\right)\right)\right]$$

再对Eq. (95)做个小trick，不等式右边的绿色部分可以化简：

$$\begin{aligned}\mathbb{E}\left[\log\frac{p_\theta(x_{t-1}|x_t)}{p_{\text{ref}}(x_{t-1}|x_t)}\right] &= \mathbb{E}\left[\log\frac{p(x_{t-1}|x_t,x_0)}{p_{\text{ref}}(x_{t-1}|x_t)} - \log\frac{p(x_{t-1}|x_t,x_0)}{p_\theta(x_{t-1}|x_t)}\right] \\ &= D_{KL}\left(p(x_{t-1}|x_t,x_0)\,\|\,p_{\text{ref}}(x_{t-1}|x_t)\right) - D_{KL}\left(p(x_{t-1}|x_t,x_0)\,\|\,p_\theta(x_{t-1}|x_t)\right)\end{aligned} \tag{96}$$

又因为三个分布的方差完全相同，所以KL散度只和均值之差相关。

$$D_{KL}\left(\mathcal{N}(\mu_1,\sigma^2I)\,\|\,\mathcal{N}(\mu_2,\sigma^2I)\right) = \frac{1}{2\sigma^2}\|\mu_1-\mu_2\|^2 \tag{97}$$

所以我们现在的目标就是求出这仨分布的均值就行:

1. 反向过程

   这个容易得到，直接参考Eq. (21):

   $$\mu_\theta = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t,t)\right) \tag{98}$$

2. 前向过程（包含p_{ref}和p）：

   Eq. (17)可以得到：

   $$\mu_p = \frac{\alpha_t\bar{\beta}_{t-1}^2}{\bar{\beta}_t^2}\mathbf{x}_t + \frac{\bar{\alpha}_{t-1}\beta_t^2}{\bar{\beta}_t^2}\mathbf{x}_0 \tag{99}$$

把Eq. (19)，带入Eq. (99)，替换掉x_0并化简可得：

$$\mu_p = \frac{x_t}{\bar{\beta}_t^2} \cdot \alpha_t \bar{\beta}_t^2 - \frac{\alpha_t \bar{\beta}_t \beta_t^2}{\bar{\beta}_t} \epsilon = \frac{1}{\alpha_t} \left(x_t - \frac{\beta_t^2}{\bar{\beta}_t} \epsilon \right) \tag{100}$$

> **Note**
> 化简的时候用到了这个约束$\alpha_t^2 \bar{\beta}_{t-1}^2 + \beta_t^2 = \bar{\beta}_t^2$

对比Eq. (98)和Eq. (100)，可以发现他们唯一的区别就是一个用真实噪声ϵ，一个用预测噪声ϵ_θ，所以这两者的均值之差也只和噪声相关了，既然如此，Eq. (94)可以等效变成：

$$\mathcal{L}(\theta) = -\mathbb{E}_{x_0^w, x_0^l, t, \epsilon^w, \epsilon^l} \left[\ln \sigma \left(-\beta \omega_t \left(\|\epsilon^w - \epsilon_\theta(x_t^w, t)\|^2 - \|\epsilon^w - \epsilon_{\text{ref}}(x_t^w, t)\|^2 - \|\epsilon^l - \epsilon_\theta(x_t^l, t)\|^2 + \|\epsilon^l - \epsilon_{\text{ref}}(x_t^l, t)\|^2 \right) \right) \right] \tag{101}$$

这也就得到了实际Diffusion DPO的损失。

这个最终损失看起来吓人，但意思非常直白。

对 winner 图：$\|\epsilon^w - \epsilon_\theta(x_t^w)\|^2 - \|\epsilon^w - \epsilon_{\text{ref}}(x_t^w)\|^2$ 是"模型相对参考模型的去噪差距"。**小于 0 意味着新模型在 winner 上去噪更准，大于 0 意味着更差。**

对 loser 图：我们希望反过来——新模型在 loser 上**比参考模型更差。**

两者一减，再套上 $-\ln \sigma$（单调递减），模型的训练方向就非常清楚了：

> **在 winner 图上比参考模型去噪得更准；在 loser 图上比参考模型去噪得更差。**

写成代码也就是这几行：

```
t = randint(0, 1000)
noise = randn_like(x_w)              # 同一个噪声同时加到 winner 和 loser
x_t_w = add_noise(x_w, noise, t)
x_t_l = add_noise(x_l, noise, t)

# 四次前向：模型 × 参考，winner × loser
err_w_model = ||model(x_t_w, c, t) - noise||^2
err_l_model = ||model(x_t_l, c, t) - noise||^2
err_w_ref   = ||ref(x_t_w, c, t)   - noise||^2
err_l_ref   = ||ref(x_t_l, c, t)   - noise||^2

inside = -beta * ((err_w_model - err_w_ref) - (err_l_model - err_l_ref))
loss = -log(sigmoid(inside))
```

比标准扩散训练**多算的，就是参考模型的一次前向。**代码上几乎没改，但优化目标已经从"拟合数据分布"变成了"对齐人类偏好"。

Flow GRPO（on-policy）

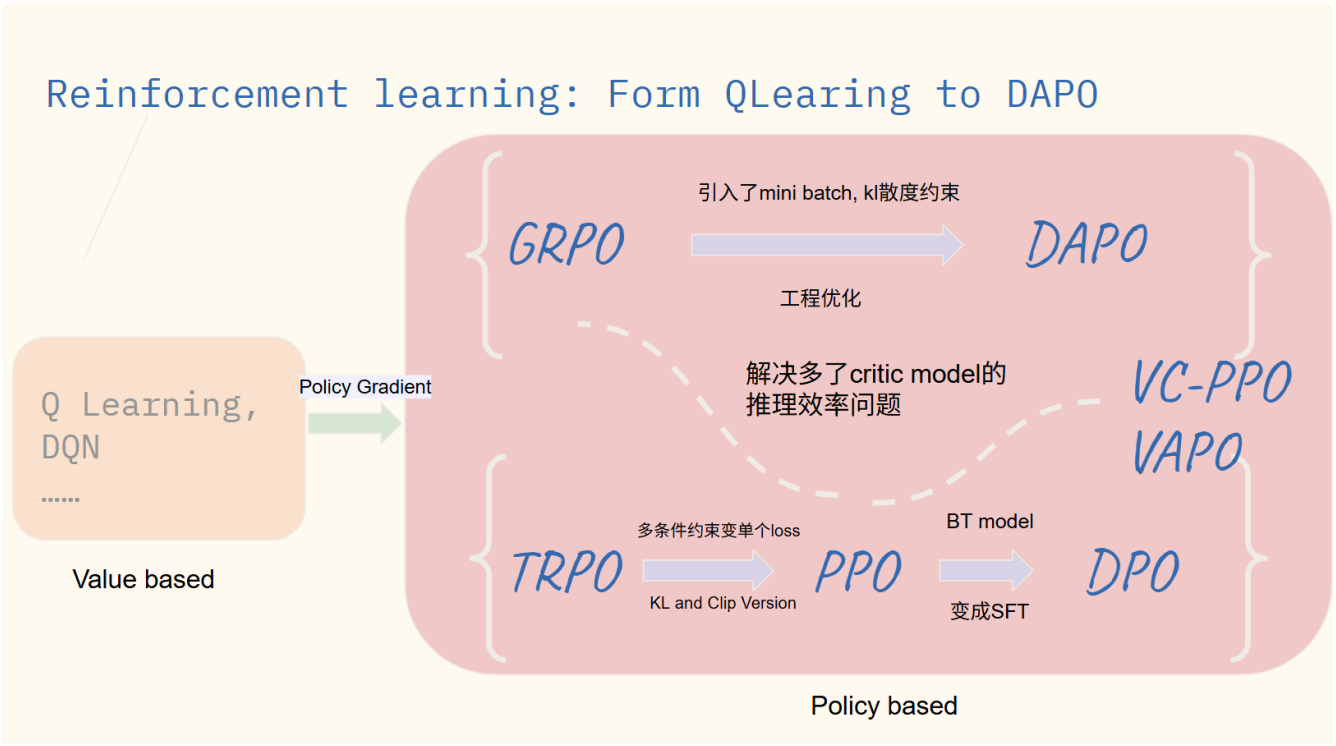
和DPO相对的，我们举一个on-policy的方法作为例子。从大家的研究兴趣来看，这种XXPO的on-policy方法显然也更受关注。

这类方法虽然在训练的cost上大于DPO，但是他的好处是对训练数据没有那么严格的要求，只需要一句能生成图片的prompt就行了。

经典GRPO

因为llm/vlm的相关工作太多了，这块就不多做赘述，这里放了一些相关链接，便于理解：

- Building DeepSeek R1 from Scratch Using Python | by Fareed Khan | Level Up Coding
- FareedKhan-dev/train-deepseek-r1: Building DeepSeek R1 from Scratch



Complex Original Equation

$$\max_{\theta} \mathbb{E}_{q,o_i} \left[\sum_{i=1}^{G} \min\left(r_i A_i, \text{clip}(r_i, 1-\epsilon, 1+\epsilon) A_i\right) - \beta D_{\text{KL}}(\pi_\theta || \pi_{\text{ref}}) \right]$$

Our Simplified Form

$$\text{OverallGoal} = \frac{1}{n} \sum_{i=1}^{n} (\text{RewardPart}_i - \text{StayStablePart}_i)$$

$$\text{RewardPart} = \text{SmallerOf}\,(\text{ChangeRatio} \times \text{Advantage}) \quad \text{and} \quad \text{SmallerOf}\,(\text{LimitedChangeRatio} \times \text{Advantage})$$

$$\frac{\text{Answer's Score} - \text{Average Score of Group}}{\text{Spread of Scores in Group}} - \text{How Much Better Score}$$

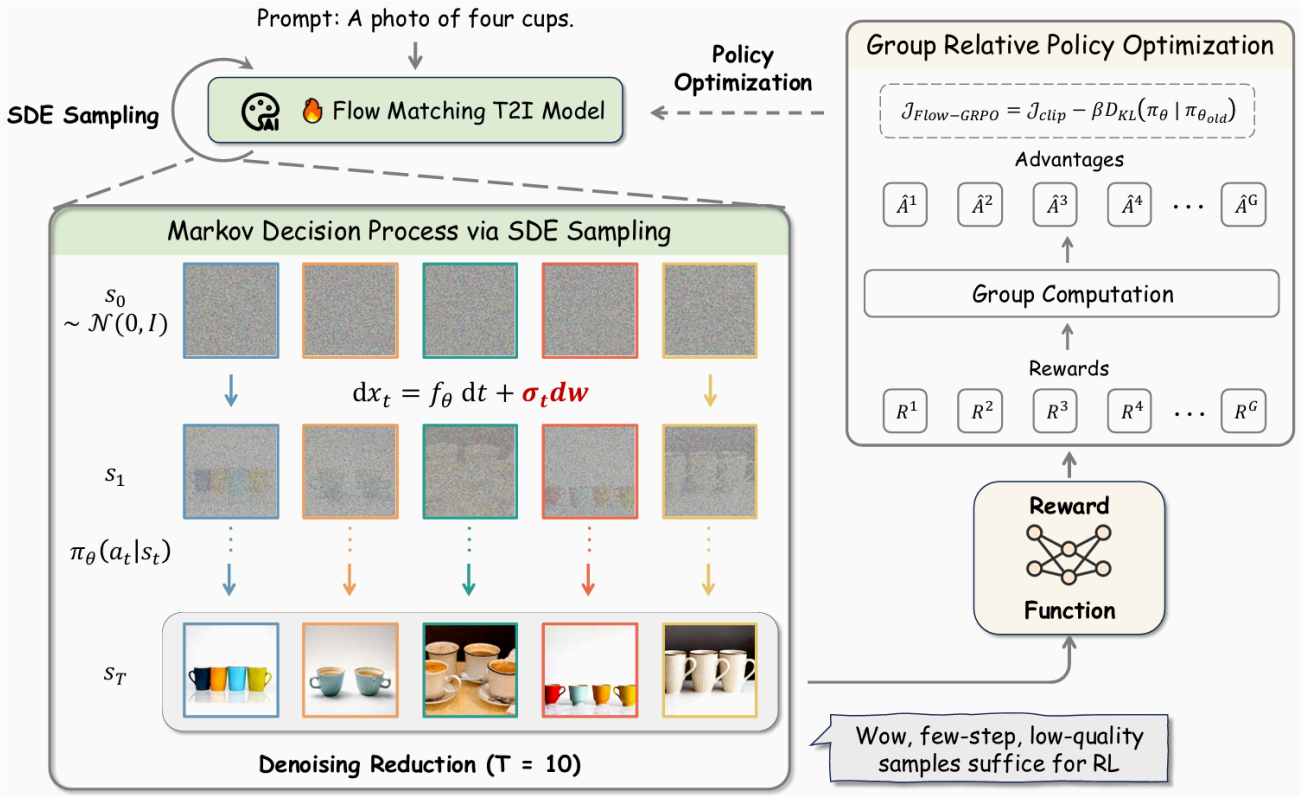
$$\frac{\text{Chance of Answer with Old Model}}{\text{Chance of Answer with New Model}} - \text{How Much Chance Changed}$$

$$\text{Limit}\,(\text{ChangeRatio}, \text{Between}\,(1-\epsilon, 1+\epsilon)) - \text{Change Ratio, but Limited}$$

$$\begin{cases} J_{GRPO}(\theta) = \mathbb{E}_{q\sim P(Q),\,\{o_i\}_{i=1}^{G}\sim\pi_{\theta_{old}}(\cdot|q)} \left[\frac{1}{G}\sum_{i=1}^{G} \frac{1}{|o_i|} \sum_{t=1}^{|o_i|} \left(\min(r_{i,t}(\theta)\hat{A}_{i,t},\ \text{clip}(r_{i,t}(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_{i,t}) - \beta\, D_{KL}[\pi_\theta || \pi_{ref}] \right) \right] \\ r_{i,t}(\theta) = \dfrac{\pi_\theta(o_{i,t} \mid q,\, o_{i,<t})}{\pi_{\theta_{old}}(o_{i,t} \mid q,\, o_{i,<t})} \\ \hat{A}_{i,t} = \dfrac{r_i - \text{mean}(\{r_1, r_2, \ldots, r_G\})}{\text{std}(\{r_1, r_2, \ldots, r_G\})} \end{cases} \tag{10}$$

Eq. (102)就是通用形式的GRPO写法了，在生成的框架里，各个符号的含义可以这样解释：

1. π_θ:最新模型的参数（策略）
2. $\pi_{\theta_{old}}$:这个gradient accumulate的batch中，最老的模型参数（策略）
3. π_{ref}:还没有开始训练的模型参数（这里是为了防止训完的模型和原始模型差太多）
4. r：每一个样本（生成图片）的评估分数，是一个常数



简单来说，Flow GRPO的整个训练流程就是:

1. 对于同一个prompt，生成好多张不一样的图片
2. 用一个Reward Function/Model来对这一组图片就行评分
3. 更新参数，让模型更容易生成评分高的图片

如此循环往复这三步。

其实从它的名字就可以看出，它是从flow matching那套理论框架往下做的，下面就细致的推导下他具体是怎么做的，比较核心的问题是:

- 我们如何根据同一个prompt，生成一组各不相同的图片。

其实这一点在Eq. (48)就已经给出了答案：

$$dx = v_\theta(x, t)dt + \sigma_t dw \tag{48}$$

Flow Matching有两种decode方式，一种是ODE，其结果是固定的；另外一种就是在ODE的基础上引入一定噪声，变成SDE，这样去噪的结果就自然带有了随机性。

但是直接拿Eq. (48)做ode->sde的转化，会带来两者并非在描述同一个velocity field：也就是说**不能保证任意时刻，每一个位置的粒子密度都一致**，使用我们尝试对它进行更精确的建模：

$$dx = f(x_t, t)dt + \sigma_t dw \tag{103}$$

于是我们的问题设置就变成了：

- **目标**：解出一个合适的$f(x_t, t)$
- **约束**：要和ode形式的velocity field任意时刻粒子密度都一致

我们不妨设：**无数个粒子组成的概率分布为**$p_t(x)$

对于ode:

那么在任意一个区域V中，概率的变化率 = 流入的概率 - 流出的概率：

$$\frac{d}{dt}\int_V p_t(x)dx = -\oint_{\partial V} p_t(x)v_t(x)\cdot \hat{n}dA \tag{104}$$

> $\hat{n}$ 是边界的外法向。

应用Gauss散度定理把面积分变成体积分：

$$\int_V \frac{\partial p_t}{\partial t}dx = -\int_V \nabla\cdot[p_t(x)v_t(x)]dx \tag{105}$$

又因为V是任意区域，所以积分可以丢掉：

$$\frac{\partial p_t}{\partial t} = -\nabla\cdot[p_t(x)v_t(x)] \tag{106}$$

> 其实就是流体力学里的连续性方程

对于sde:

沿用Eq. (29)的trick，先把sde离散化：

$$x_{t+\Delta t} = x_t + f(x_t,t)\Delta t + \sigma_t\sqrt{\Delta t}\epsilon,\quad \epsilon\sim\mathcal{N}(0,I) \tag{107}$$

所以给定 x_t，下一步 $x_{t+\Delta t}$ 的条件分布是（变成概率好推导）：

$$p(x_{t+\Delta t}|x_t) = \mathcal{N}(x_{t+\Delta t}; x_t + f\Delta t, \sigma_t^2\Delta t I) \tag{108}$$

再把条件概率积分成全概率，换元 $\delta = x - x_t$，即 $x_t = x - \delta$：：

$$p_{t+\Delta t}(x) = \int p(x|x-\delta)p_t(x-\delta)d\delta \tag{109}$$

当 $\Delta t\to 0$ 时，$\delta = x - x_t$ 主要集中在 $f\Delta t$ 附近（很小），所以把 $p_t(x_t) = p_t(x-\delta)$ 在 x 处展开：

$$p_t(x-\delta) \approx p_t(x) - \delta\cdot\nabla p_t(x) + \frac{1}{2}\delta\delta^T\cdot\nabla^2 p_t(x)$$

代入积分：

$$p_{t+\Delta t}(x) \approx \underbrace{\int p(x|x-\delta)d\delta}_{1}\cdot p_t(x) - \underbrace{\int \delta p(x|x-\delta)d\delta}_{\mathbb{E}[\delta]}\cdot\nabla p_t(x) + \frac{1}{2}\underbrace{\int \delta\delta^T p(x|x-\delta)d\delta}_{\mathbb{E}[\delta\delta^T]} : \nabla^2 p_t(x) \tag{110}$$

所以 $\delta = x - x_t = f\Delta t + \sigma_t\sqrt{\Delta t}\,\epsilon$，所以可以算出：

$$\begin{cases}\mathbb{E}[\delta] = f\Delta t \\ \mathbb{E}[\delta\delta^T] = \sigma_t^2\Delta t\, I\end{cases} \tag{111}$$

再带入Eq. (110)，可以得到：

$$p_{t+\Delta t}(x) \approx p_t(x) - f\Delta t\cdot\nabla p_t(x) + \frac{1}{2}\sigma_t^2\Delta t\,\nabla^2 p_t(x) \tag{112}$$

此时右边完成了对x的泰勒展开，我们再对左边进行对于p的泰勒展开：

$$p_t(x) + \frac{\partial p_t}{\partial t}\Delta t \approx p_t(x) - f\Delta t\cdot\nabla p_t(x) + \frac{1}{2}\sigma_t^2\Delta t\,\nabla^2 p_t(x) \tag{113}$$

化简得到：

$$\frac{\partial p_t}{\partial t} = -\nabla\cdot[f(x,t)p_t] + \frac{1}{2}\sigma_t^2\nabla^2 p_t \tag{114}$$

我们要找的SDE和原来的ODE共享同样的 p_t，所以它们的演化方程右边必须相等，联立Eq. (114) 和 Eq. (106)：

$$-\nabla\cdot[fp_t] + \frac{1}{2}\sigma_t^2\nabla^2 p_t + \nabla\cdot[v_t p_t] = 0 \tag{115}$$

这个式子就可以接出来$f(x_t, t)$了：

$$f = v_t + \frac{1}{2}\sigma_t^2 \nabla \log p_t \tag{116}$$

Eq. (103)的正向SDE就可以写成：

$$dx_t = \left[v_t(x_t) + \frac{1}{2}\sigma_t^2 \nabla \log p_t(x_t)\right] dt + \sigma_t \, dw \tag{117}$$

有了正向sde的推导，自然能想到把Eq. (34)用一下，直接得到反向sde（也就是去噪生成图片过程）的公式：

$$dx_t = \left[v_t(x_t) - \frac{1}{2}\sigma_t^2 \nabla \log p_t(x_t)\right] dt + \sigma_t \, d\bar{w} \tag{118}$$

Eq. (118) 里出现了 $\nabla \log p_t(x_t)$，但 Flow Matching 模型学的是 velocity field v_θ，**根本没训练过 score。要么再训一个 score 网络，要么直接把 score 用 velocity 表达出来。**

那用什么工具找这个关系？

只能从两者**共同的根源**出发—— x_t 的分布。具体说：

- score 是 "$\log p_t(x_t)$ 对 x_t 求梯度"
- velocity 是 "粒子在 x_t 这一点的瞬时速度的期望"

两者都依赖 $p_t(x_t)$。所以我们的策略是：

> **把 $p_t(x_t)$ 写明白，然后从同一个分布出发，分别推出 score 的表达式和 velocity 的表达式，最后联立消去中间量。**

在概率视角DDPM中，我们曾经有Eq. (15)这个等式

$$p(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \alpha_t \mathbf{x}_0, \beta_t^2 \mathbf{I}) \tag{15}$$

于是条件score可以用α_t，β_t表示出来。

$$\nabla_{x_t} \log p_{t|0}(x_t|x_0) = -\frac{1}{2\beta_t^2} \cdot 2(x_t - \alpha_t x_0) = -\frac{x_t - \alpha_t x_0}{\beta_t^2} = -\frac{\beta_t x_1}{\beta_t^2} = -\frac{x_1}{\beta_t} \tag{119}$$

不过我们想要得到的是**边际score**而不是**条件score**，一个比较直觉的做法是，把所有可能的x_0做积分，那不就得到边际score了吗

$$\nabla \log p_t(x_t) = \mathbb{E}_{x_0 \sim p(x_0|x_t)} \left[\nabla \log p_{t|0}(x_t \mid x_0)\right] \tag{120}$$

> 这里只是从直觉上写出了Eq. (120)，具体证明不赘述

联立Eq. (119)与Eq. (120)可得：

$$\nabla log p_t(x_t) = \mathbb{E}_{x_1 \sim p(x_1|x_t)}[-\frac{x_1}{\beta_t} \mid x_t] = -\frac{1}{\beta_t}\mathbb{E}_{x_1 \sim p(x_1|x_t)}[x_1 \mid x_t] \tag{121}$$

现在我们已经找到了关联score和velocity的中介$\mathbb{E}_{x_1 \sim p(x_1|x_t)}[x_1 \mid x_t]$，下面就尝试一下把velocity的表达形式推导一下：

回忆 Flow Matching 里 velocity field 的定义，在线性插值 $x_t = \alpha_t x_0 + \beta_t x_1$ 下，target velocity 是**沿 ODE 轨迹的瞬时速度的条件期望**：

> **Caution**
>
> 为啥是条件期望？因为任何一个位置都可能在好几个samples的轨迹上，那么这个点的瞬时速度就应该是所有samples的gt速度加权平均

$$v_t(x_t) = \mathbb{E}_{x_1 \sim p(x_1|x_t)} \left[\frac{dx_t}{dt} \mid x_t\right] \tag{122}$$

不妨定义$\dot{\alpha}_t \equiv d\alpha_t/dt$，同理 $\dot{\beta}_t$。

$$v_t(x_t) = \dot{\alpha}_t \, \mathbb{E}_{x_1 \sim p(x_1|x_t)}[x_0 \mid x_t] + \dot{\beta}_t \, \mathbb{E}_{x_1 \sim p(x_1|x_t)}[x_1 \mid x_t] \tag{123}$$

似乎Eq. (123)里出现了我们不希望看到的$\mathbb{E}_{x_1 \sim p(x_1|x_t)}[x_0 \mid x_t]$，不过根据Eq. (5)，$x_0$是可以被$x_t$和$x_1$联合表示出来的：

$$\mathbb{E}_{x_1\sim p(x_1|x_t)}[x_0 \mid x_t] = \frac{x_t - \beta_t\, \mathbb{E}_{x_1\sim p(x_1|x_t)}[x_1 \mid x_t]}{\alpha_t} \tag{124}$$

带入Eq. (123)整理之后得到：

$$v_t(x_t) = \frac{\dot\alpha_t}{\alpha_t}x_t + \left(\dot\beta_t - \frac{\dot\alpha_t\beta_t}{\alpha_t}\right)\mathbb{E}_{x_1\sim p(x_1|x_t)}[x_1 \mid x_t] \tag{125}$$

接下来就可以联立Eq. (125)和Eq. (121)消元了，而且由于Flow Matching选择的是**OT路径**，还可以用Eq. (43)进一步化简：

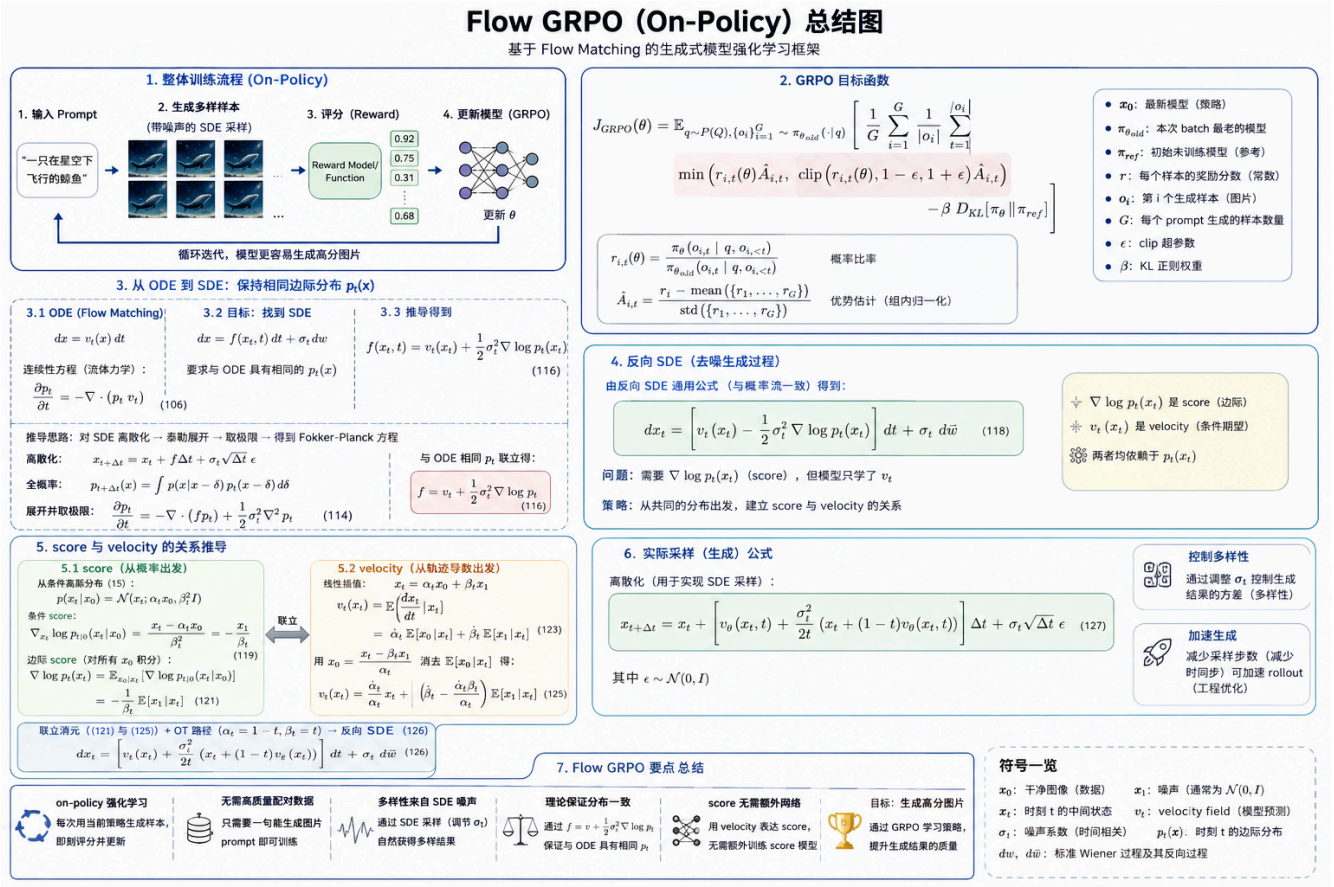
$$dx_t = \left[v_t(x_t) + \frac{\sigma_t^2}{2t}\Big(x_t + (1-t)\,v_t(x_t)\Big)\right]dt + \sigma_t\, d\bar w \tag{126}$$

同样的离散化技巧：

$$x_{t+\Delta t} = x_t + \left[v_\theta(x_t,t) + \frac{\sigma_t^2}{2t}(x_t + (1-t)v_\theta(x_t,t))\right]\Delta t + \sigma_t\sqrt{\Delta t}\,\epsilon \tag{127}$$

这就是我们sde采样的最终公式！可以通过调整σ_t来控制生成图片batch之内的方差。

最后还有一个工程上的小优化，rollout的时间可以通过减少采样步骤来加速。



3. Diffusion OPD

> 生成有个趋势是会把llm里验证过比较有用的东西搬过来再做一遍，大约半年前开始llm领域开始出现大量OPD相关文章，最近生成相关的迁移工作也慢慢多起来了，不过基本都大同小异，这里以DiffusionOPD为例简单介绍一下

DiffusionOPD

和之前的一样，可以先看一眼常规OPD的式子：

$$\mathcal{L}_{\mathrm{OPD}}^{\mathrm{LLM}}(\theta) = \mathbb{E}_{x\sim\pi_\theta}\left[\sum_{t=1}^{T}\mathrm{KL}\left(\pi_\theta(\cdot|x_{<t})\,\|\,\pi^\star(\cdot|x_{<t})\right)\right] \tag{128}$$

直观上理解很简单，这里的π_θ代表学生模型（能力强），π^*代表教师模型（能力弱），如果我们在同一输入的情况下，让学生模型输出尽可能的靠近教师模型，那自然就能提升学生模型的能力了，所以这里的$\mathcal{L}_{\text{OPD}}^{\text{LLM}}(\theta)$存在一个学生-教师之间的KL散度。

那换到生成的语境下，这里的策略π的实际涵义就变成了：

- $\pi_\theta(\cdot|x_{<t})$: 学生模型去噪到第t步的时候，接下来去噪图像的概率
- $\pi^*(\cdot|x_{<t})$: **教师模型拿到学生模型去噪到第t步的条件的时候**，接下来去噪图像的概率

形式化一下，可以写成：

$$\mathcal{L}_{\text{OPD}}(\theta) = \mathbb{E}_{x_{0:N}\sim p_S}\left[\sum_{j=0}^{N-1} \text{KL}\left(p_S(\cdot|x_{t_j}) \,\|\, p_T(\cdot|x_{t_j})\right)\right] \tag{129}$$

在上个section中，我们推导出的好多结论其实在这里可以直接用，这个损失函数的化简就会变得比较清晰：

首先，根据Eq. (107)可以知道方差只和我们选择的超参σ_t，也就是scheduler相关，不妨假设教师模型和学生模型共享同一个scheduler，那么$p_S(\cdot|x_{t_j})$和$p_T(\cdot|x_{t_j})$就变成了同方差的两个分布。

计算同方差分布的KL散度的trick在Eq. (97)用过一次，之和其均值相关，那么就可以化简成：

$$\mathcal{L}_{\text{OPD}}^{\text{diffusion-ODE}}(\theta) = \mathbb{E}_{x_{0:N}\sim p_{S,\theta}}\left[\sum_{j=0}^{N-1} \frac{1}{2}\|\mu_S(x_{t_j};\theta) - \mu_T(x_{t_j})\|_2^2\right] \tag{130}$$

可以直接用来训练了。

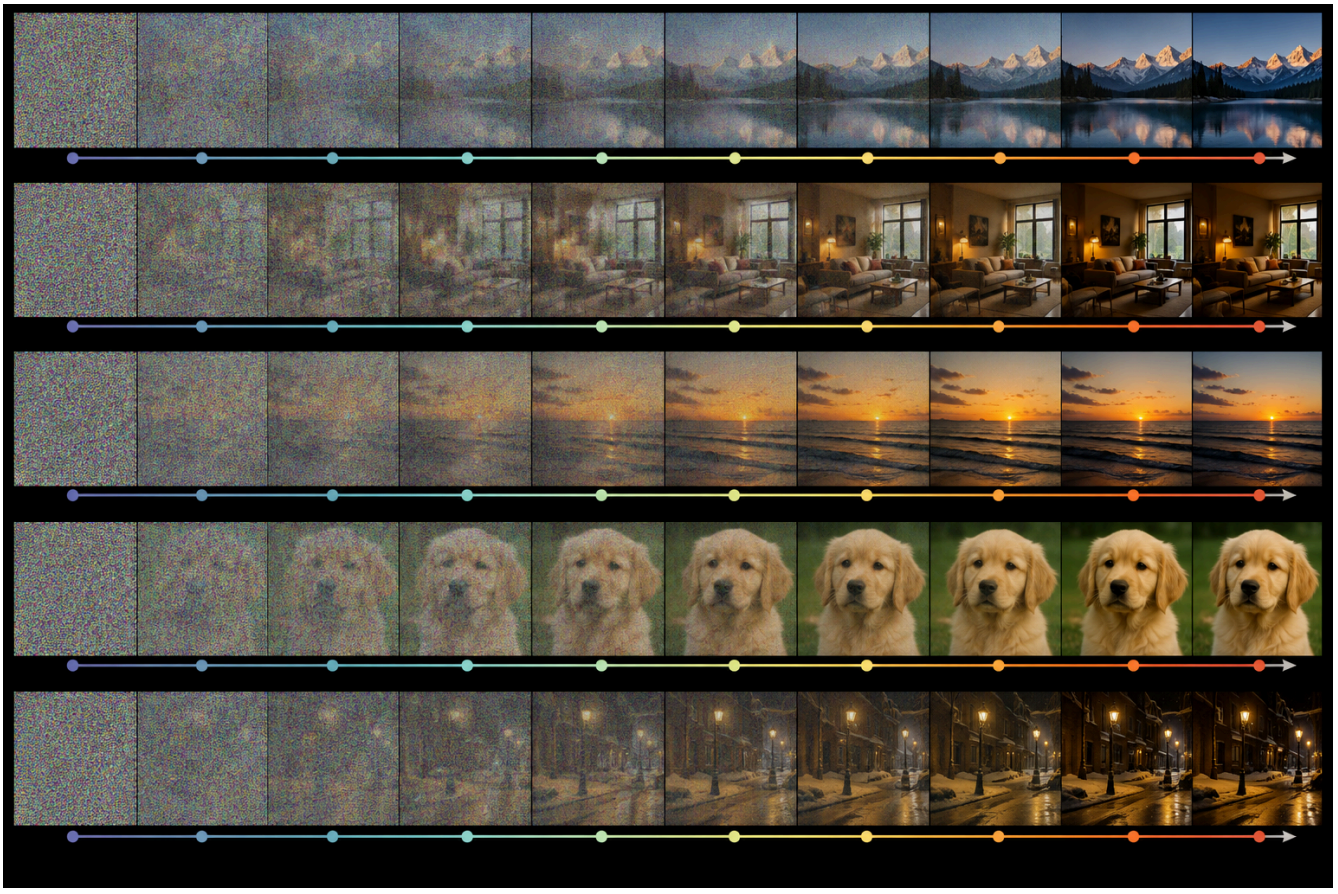
> **Tip**
>
> 个人感觉这里假设教师模型和学生模型共享同一个scheduler的限制是可以通过进一步推导绕过去的，比如从flow grpo那块开始，让student model的sde推导出的p和teacher model的ode推导出的p一致，可能是更本质的做法。

4. Diffusion Model 中的 Caching

> 下面的两个section就不涉及形式化的推导了

teacache

首先让我们观察下一次常规多步去噪过程中，这张噪声图是怎么一步步演化的：



可以看到其实中间去噪步骤并不是"均匀"的，一个自然而然的想法是，我能不能在这些变化比较小的地方采样少一点，从而用更少的总步数来得到得到质量相似的最终图片。

具体来说，就是用$t+k$的velocity space来代替t时刻的velocity space

现在唯一要解决的问题就是：

- **怎么判断哪些去噪步数是值得被跳过的呢**

比较直觉的方法是对比第t步和第t+1步的距离：

$$\mathcal{L}_{\text{rel}}^{1}(O,t)=\frac{\|O_t-O_{t+1}\|}{\|O_{t+1}\|}$$

如果这个误差累积到一定程度，那就说明我们不得不再进行一次前向传播了：

$$\sum_{t=t_a}^{t_b-1}\mathrm{L1}_{\text{rel}}(\mathbf{O},t)\leq\delta<\sum_{t=t_a}^{t_b}\mathrm{L1}_{\text{rel}}(\mathbf{O},t) \tag{131}$$

不过，想想$\mathbf{O}$是怎么来的，他是**上一次的latent经过一次完整forward得到的结果。**如果真用这个做判断的话，实际上不会有任何跳步的加速，因为我们必须做每一次中间step推导。

原文给了个simple yet effective的方案：

> 假设F是带有时序信息的input embedding，O是transformer 跑完后预测的速度场

1. 采样一堆prompts跑完整的原始推理
2. 在每一步 t，同时记录两个标量
    - $x_t=L1_{rel}(F,t)$ # 调制后输入的相对 L1 差异
    - $y_t=L1_{rel}(O,t)$ # 模型输出的相对 L1 差异
3. 用多项式函数f拟合：输入为F，输出为O的函数

这样在实际inference的时候，就可以用:

$$\sum_{t=t_a}^{t_b-1} f(\mathrm{L1}_{\mathrm{rel}}(\mathbf{F},t)) \le \delta < \sum_{t=t_a}^{t_b} f(\mathrm{L1}_{\mathrm{rel}}(\mathbf{F},t)) \tag{132}$$

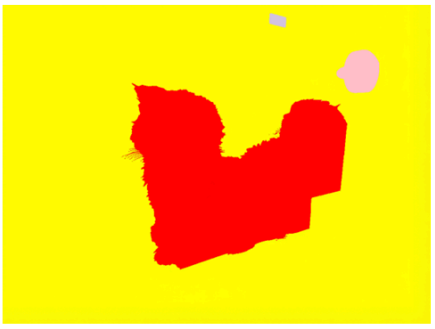
来判断是否要跳过这一步denoise了

5. 生成模型的通用视觉理解能力

VisionBanana

这篇没有架构的创新，但是指出了一个新方向，生成模型不单单能生成美学意义上的图片，他也能通过少量数据的微调做实力分割，空间理解等下游任务。

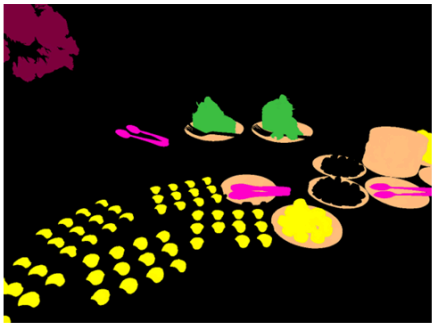
(a) Example 1



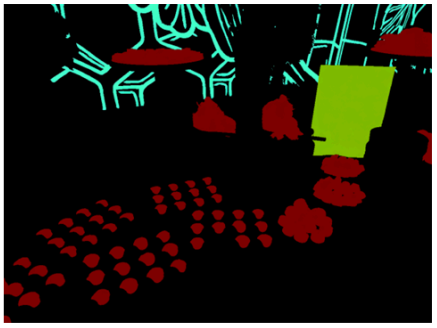
“Generate a semantic segmentation visualization image, using this color mapping: {"cat": "red", "lock": "pink", "exit sign": "light purple", "background": yellow}.”

“Generate a visualization image of semantic segmentation, using this color mapping: {"cat ears": <255, 165, 0>, "exit sign": <0, 0, 255>, "background":<125,0, 125>}”

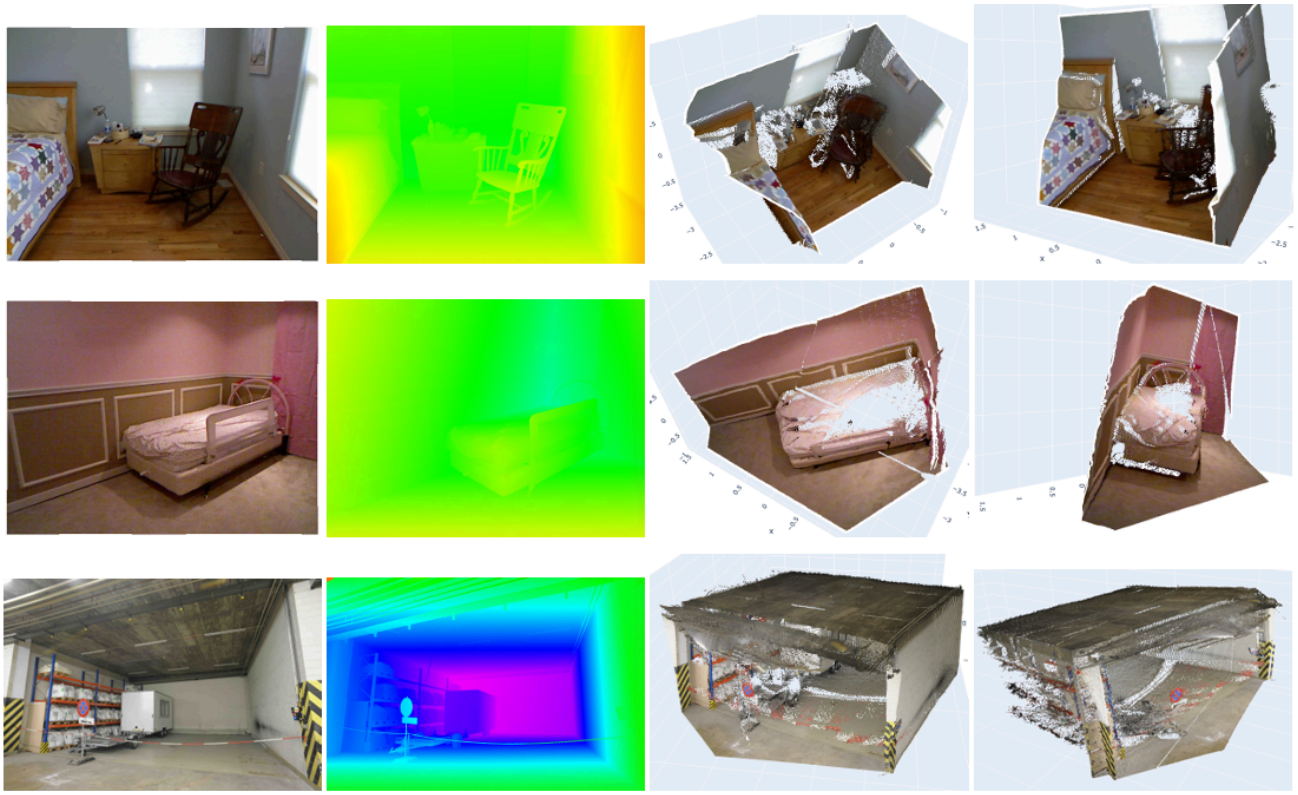
(b) Example 2



“This image is a per-pixel class labeling of the input. The macaron cakes are represented by (255, 255, 0). The round plates are represented by (255, 192, 128). The slice cakes are depicted in (64, 192, 64). The flowers are shown in (128, 0, 64). The tongs are (255, 0, 192).”



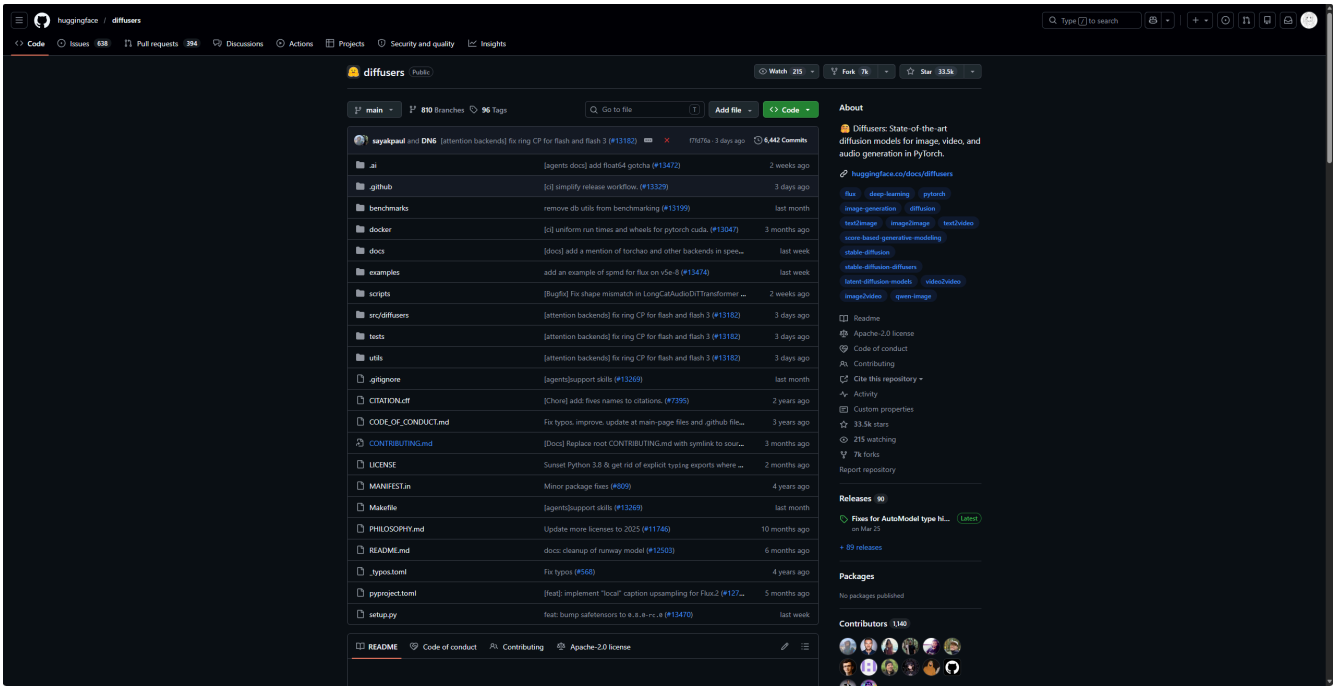
“Generate a semantic segmentation visualization of the input. The menu is #80C000. The dessert is #800000. The patterns on the wall is #40FFC0”



常用代码库与工具

1. Diffusers

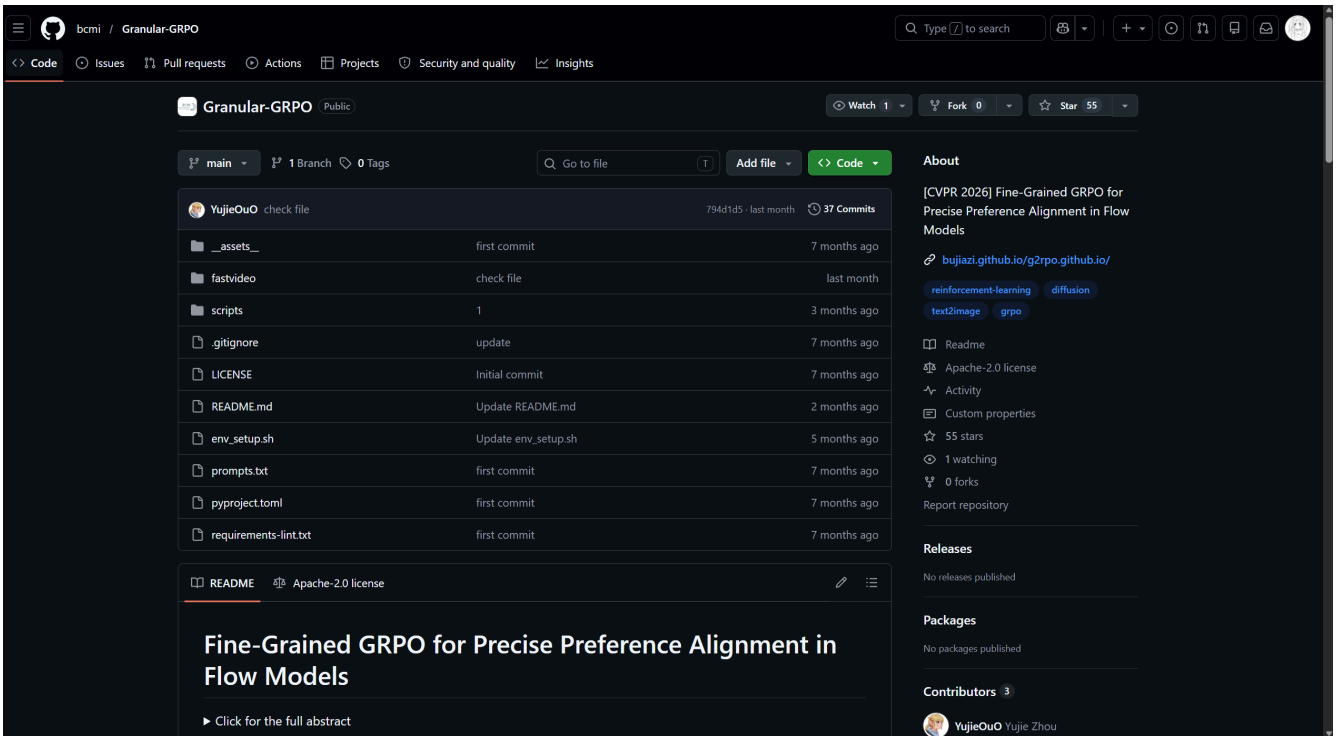
> **Important**
>
> huggingface/diffusers: 🤗 Diffusers: State-of-the-art diffusion models for image, video, and audio generation in PyTorch.



常用的sft codebase，对多卡多节点训练支持的也不错

2. G2RPO

> **Important**
>
> [bcmi/Granular-GRPO: CVPR 2026] Fine-Grained GRPO for Precise Preference Alignment in Flow Models



做rl的学长推荐的rl相关codebase

3. 科学空间

> **Important**
>
> 科学空间|Scientific Spaces

很不错的博客，写的时候也参考了很多。

Reference

1. [2006.11239] Denoising Diffusion Probabilistic Models
2. [2010.02502] Denoising Diffusion Implicit Models
3. [2011.13456] Score-Based Generative Modeling through Stochastic Differential Equations
4. [2210.02747] Flow Matching for Generative Modeling
5. [2207.12598] Classifier-Free Diffusion Guidance
6. [2403.03206v1] Scaling Rectified Flow Transformers for High-Resolution Image Synthesis
7. [2505.13447] Mean Flows for One-step Generative Modeling
8. [2512.05150] TwinFlow: Realizing One-step Generation on Large Models with Self-adversarial Flows
9. [2202.00512] Progressive Distillation for Fast Sampling of Diffusion Models
10. [2303.01469] Consistency Models
11. [2209.03003] Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow
12. [2311.12908] Diffusion Model Alignment Using Direct Preference Optimization
13. [2505.05470] Flow-GRPO: Training Flow Matching Models via Online RL
14. [2411.19108] Timestep Embedding Tells: It's Time to Cache for Video Diffusion Model
15. [2604.20329] Image Generators are Generalist Vision Learners
16. [2605.08063] Flow-OPD: On-Policy Distillation for Flow Matching Models
17. [2605.15055] DiffusionOPD: A Unified Perspective of On-Policy Distillation in Diffusion Models