扩散模型之DiT：纯Transformer架构

扩散模型大部分是采用**UNet架构** 来进行建模，UNet可以实现输出和输入一样维度，所以天然适合扩散模型。扩散模型使用的UNet除了包含基于残差的卷积模块，同时也往往采用self-attention。自从ViT之后，transformer架构已经大量应用在图像

任务上，随着扩散模型的流行，也已经有工作尝试采用transformer架构来对扩散模型建模，这篇文章我们将介绍Meta的工作**DiT**：[Scalable Diffusion Models with Transformers]，它是**完全基于transformer架构的扩散模型**，这个工作不仅将transformer成功应用在扩散模型，还探究了**transformer架构在扩散模型上的scalability能力，其中最大的模型DiT-XL/2在ImageNet 256x256的类别条件生成上达到了SOTA（FID为2.27）**。

在介绍DiT模型架构之前，我们先来看一下DiT所采用的扩散模型。 首先，DiT并没有采用常规的pixel diffusion，而是**采用了latent diffusion架构** ，这也是Stable Diffusion所采用的架构。latent diffusion采用一个autoencoder来将图像压缩为低维度的latent，扩散模型用来生成latent，然后再采用autoencoder来重建出图像。DiT采用的autoencoder是SD所使用的KL-f8，对于256x256x3的图像，其压缩得到的latent大小为32x32x4，这就降低了扩散模型的计算量（后面我们会看到这将减少transformer的token数量）。另外，这里扩散过程的nosie scheduler采用简单的linear scheduler（timesteps=1000，beta_start=0.0001，beta_end=0.02），这个和SD是不同的。其次，DiT所使用的扩散模型沿用了OpenAI的**Improved DDPM**，相比原始DDPM一个重要的变化是不再采用固定的方差，而是**采用网络来预测方差** 。在DDPM中，生成过程的分布采用一个参数化的高斯分布来建模：

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t)) \quad (1)$$

不过DDPM采用固定的方差，即$\boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t)$采用固定值，在生成过程中设置为$\beta_t$或者$\tilde{\beta}_t$，而$\beta_t$和$\tilde{\beta}_t$其实是一个上下限。而Improved DDPM采用网络来预测方差：

$$\boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t) = \exp\left(\mathbf{v} \log \beta_t + (1-\mathbf{v})\tilde{\beta}_t\right) \quad (2)$$

这里网络并不是直接预测方差，而是预测一个系数$\mathbf{v}$，并通过在β_t和$\tilde{\beta}_t$之间插值来计算最终的方差。DDPM的$\boldsymbol{\mu}_\theta(\mathbf{x}_t, t)$通过$L_{simple}$来进行优化，但是这个损失函数并不依赖$\boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t)$，为了优化$\boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t)$，Improved DDPM采用了一个组合损失函数：

$$L_{hybrid} = L_{simple} + \lambda L_{vlb} \tag{3}$$

这里的L_{vlb}是扩散模型原始的VLB损失，注意这里在计算VLB时，要截断$\boldsymbol{\mu}_\theta(\mathbf{x}_t, t)$的梯度，即$L_{vlb}$只负责优化$\boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t)$，而不会影响$\boldsymbol{\mu}_\theta(\mathbf{x}_t, t)$，这里的系数$\lambda$默认取0.001。关于VLB的计算，可以参考OpenAI开源的原始代码。 要注意的一点是，预测方差不需要再训练一个网络，而是直接在原来的网络上增加一倍的输出即可，比如对于32x32x4的latent，让网络输出32x32x8就可以了，其中一半用来预测噪音，一半用来预测方差系数$\mathbf{v}$。

上面介绍完了DiT所采用的扩散模型设置，然后我们来介绍DiT所设计的transformer架构，这才是这个工作的核心。其实DiT基本沿用了ViT的设计，如下图所示，首先采用一个**patch embedding来将输入进行patch化**，即得到一系列的tokens。其中patch size属于一个超参数，它直接决定了tokens的数量，这会影响模型的计算量。DiT的patch size共选择了三种设置：$p = 2, 4, 8$。注意token化之后，这里还要加上positional embeddings，这里采用非学习的sin-cosine位置编码。

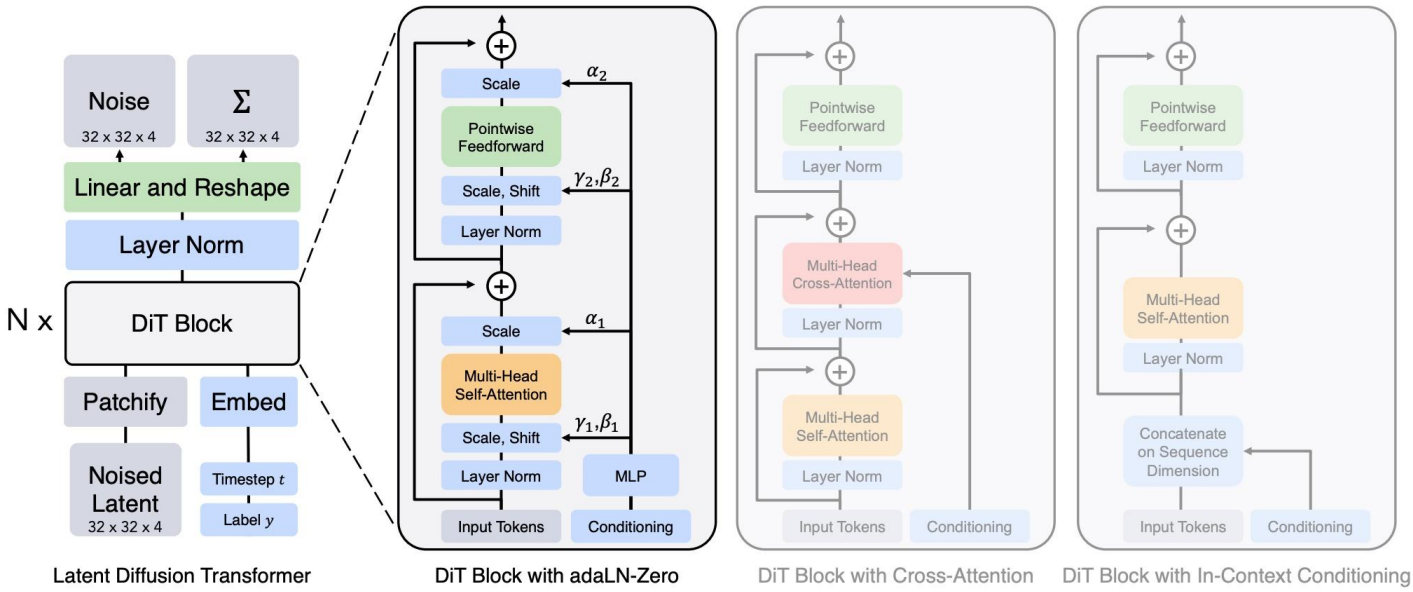

Figure 3. **The Diffusion Transformer (DiT) architecture.** *Left:* We train conditional latent DiT models. The input latent is decomposed into patches and processed by several DiT blocks. *Right:* Details of our DiT blocks. We experiment with variants of standard transformer blocks that incorporate conditioning via adaptive layer norm, cross-attention and extra input tokens. Adaptive layer norm works best.

将输入token化之后，就可以像ViT那样接transformer blocks了。但是对于扩散模型来说，往往还需要在网络中嵌入额外的条件信息，这里的条件包括timesteps以及类别标签（如果是文生图就是文本，但是DiT这里并没有涉及）。要说明的一点是，无论是timesteps还是类别标签，都可以采用一个embedding来进行编码。DiT共设计了四种方案来实现两个额外embeddings的嵌入，具体如下：

1. **In-context conditioning**：将两个embeddings看成两个tokens合并在输入的tokens中，这种处理方式有点类似ViT中的cls token，实现起来比较简单，也不基本上不额外引入计算量。

2. **Cross-attention block**： 将两个embeddings拼接成一个数量为2的序列，然后在transformer block中插入一个cross attention，条件embeddings作为cross attention的key和value；这种方式也是目前文生图模型所采用的方式，它需要额外引入15%的Gflops。
3. **Adaptive layer norm (adaLN) block**： 采用adaLN，这里是将time embedding和class embedding相加，然后来回归scale和shift两个参数，这种方式也基本不增加计算量。
4. **adaLN-Zero block**： 采用zero初始化的adaLN，这里是将adaLN的linear层参数初始化为zero，这样网络初始化时transformer block的残差模块就是一个identity函数；另外一点是，这里除了在LN之后回归scale和shift，还在每个残差模块结束之前回归一个scale，如上图所示。

论文对四种方案进行了对比试验，发现采用**adaLN-Zero** 效果是最好的，所以DiT默认都采用这种方式来嵌入条件embeddings。

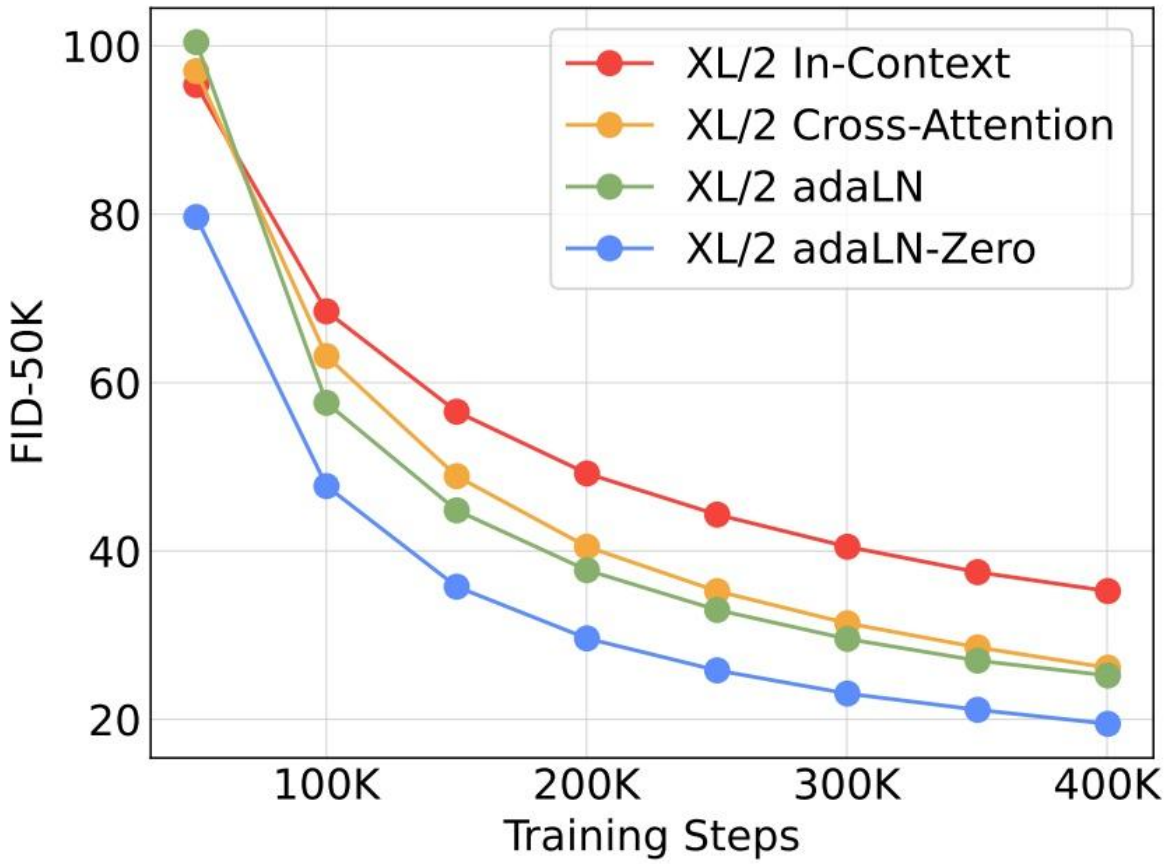

Figure 5. **Comparing different conditioning strategies.** adaLN-Zero outperforms cross-attention and in-context conditioning at all stages of training.

这里也贴一下基于**adaLN-Zero** 的DiT block的具体实现代码：

```
class DiTBlock(nn.Module):
    """
    A DiT block with adaptive layer norm zero
(adaLN-Zero) conditioning.
    """
    def __init__(self, hidden_size, num_heads,
mlp_ratio=4.0, **block_kwargs):
```

```
        super().__init__()
        self.norm1 = nn.LayerNorm(hidden_size,
elementwise_affine=False, eps=1e-6)
        self.attn = Attention(hidden_size,
num_heads=num_heads, qkv_bias=True,
**block_kwargs)
        self.norm2 = nn.LayerNorm(hidden_size,
elementwise_affine=False, eps=1e-6)
        mlp_hidden_dim = int(hidden_size *
mlp_ratio)
        approx_gelu = lambda:
nn.GELU(approximate="tanh")
        self.mlp = Mlp(in_features=hidden_size,
hidden_features=mlp_hidden_dim,
act_layer=approx_gelu, drop=0)
        self.adaLN_modulation = nn.Sequential(
            nn.SiLU(),
            nn.Linear(hidden_size, 6 *
hidden_size, bias=True)
        )

        # zero init

 nn.init.constant_(adaLN_modulation[-1].weight, 0)

 nn.init.constant_(adaLN_modulation[-1].bias, 0)
```

```
    def forward(self, x, c):
        shift_msa, scale_msa, gate_msa, shift_mlp,
scale_mlp, gate_mlp =
self.adaLN_modulation(c).chunk(6, dim=1)
        x = x + gate_msa.unsqueeze(1) *
self.attn(modulate(self.norm1(x), shift_msa,
scale_msa))
        x = x + gate_mlp.unsqueeze(1) *
self.mlp(modulate(self.norm2(x), shift_mlp,
scale_mlp))
        return x
```

虽然DiT发现**adaLN-Zero** 效果是最好的，但是这种方式只适合这种只有类别信息的简单条件嵌入，因为只需要引入一个class embedding；但是对于文生图来说，其条件往往是序列的text embeddings，采用cross-attention方案可能是更合适的。 由于对输入进行了token化，所以在网络的最后还需要一个decoder来恢复输入的原始维度，DiT采用一个简单的linear层来实现，直接将每个token映射为$p\times p\times 2C$的tensor，然后再进行reshape来得到和原始输入空间维度一样的输出，但是特征维度大小是原来的2倍，分别用来预测噪音和方差。具体实现代码如下所示：

```
class FinalLayer(nn.Module):
    """
```

```
    The final layer of DiT.
    """
    def __init__(self, hidden_size, patch_size,
out_channels):
        super().__init__()
        self.norm_final =
nn.LayerNorm(hidden_size,
elementwise_affine=False, eps=1e-6)
        self.linear = nn.Linear(hidden_size,
patch_size * patch_size * out_channels, bias=True)
        self.adaLN_modulation = nn.Sequential(
            nn.SiLU(),
            nn.Linear(hidden_size, 2 *
hidden_size, bias=True)
        )


nn.init.constant_(self.adaLN_modulation[-1].weight
, 0)

 nn.init.constant_(self.adaLN_modulation[-1].bias,
0)
        nn.init.constant_(self.linear.weight, 0)
        nn.init.constant_(self.linear.bias, 0)

    def forward(self, x, c):
```

```
        shift, scale =
self.adaLN_modulation(c).chunk(2, dim=1)
        x = modulate(self.norm_final(x), shift,
scale)
        x = self.linear(x)
        return x
```

注意这里先进行LayerNorm，同时也引入了zero adaLN，并且decoder的linear层也采用zero初始化。 仿照ViT，DiT也设计了4种不同规模的模型，分别是DiT-S、DiT-B、DiT-L和DiT-XL，其中最大的模型DiT-XL参数量为675M，计算量Gflops为29.1（256x256图像，patch size=4时）。四个模型的具体配置如下所示：

Model	Layers N	Hidden size d	Heads	Gflops (I=32, p=4)
DiT-S	12	384	6	1.4
DiT-B	12	768	12	5.6
DiT-L	24	1024	16	19.7
DiT-XL	28	1152	16	29.1

Table 1. **Details of DiT models.** We follow ViT [10] model configurations for the Small (S), Base (B) and Large (L) variants; we also introduce an XLarge (XL) config as our largest model.

论文重点探究了不同规模的DiT的性能，即模型的scalability能力，不同模型的性能对比如下所示：

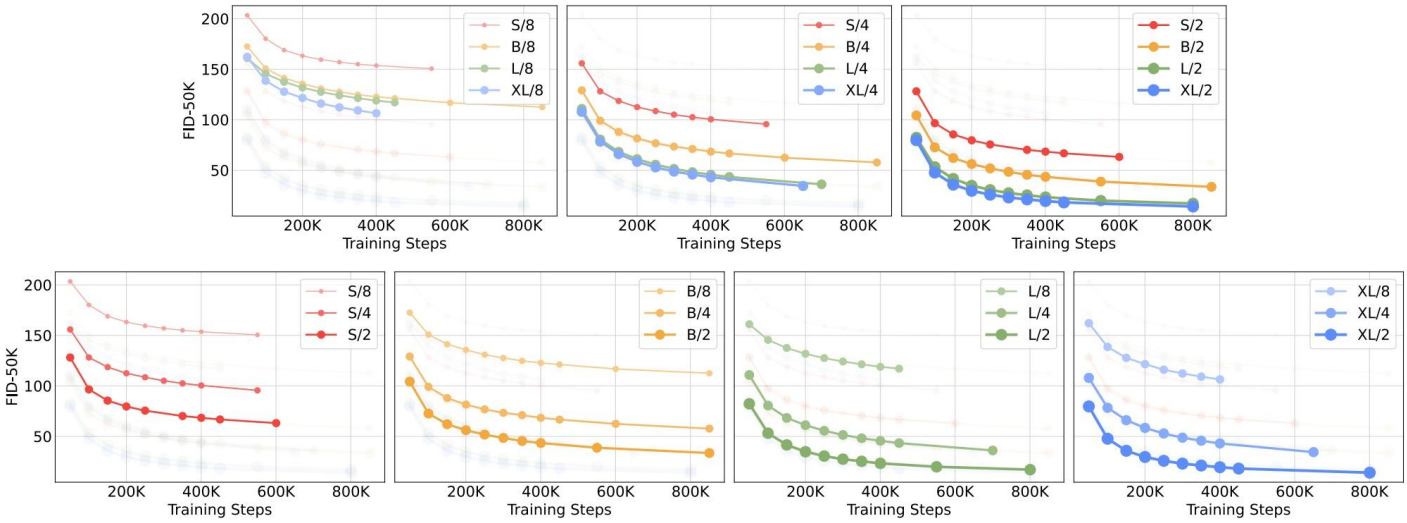

Figure 6. **Scaling the DiT model improves FID at all stages of training.** We show FID-50K over training iterations for 12 of our DiT models. *Top row:* We compare FID holding patch size constant. *Bottom row:* We compare FID holding model size constant. Scaling the transformer backbone yields better generative models across all model sizes and patch sizes.

注意对于DiT来说，除了模型参数会影响计算量，patch size也会影响计算量。可以看到无论是固定patch size增大模型参数，还是固定模型参数降低patch size，均能够提升生成质量，两个的共性都是增大了计算量。所以论文进一步绘制了模型Gflops和生成质量（FID）之间的关系，如下图所示，可以看到两者的正相关关系，这说明模型Gflops对最终的生成效果是至关重要的。

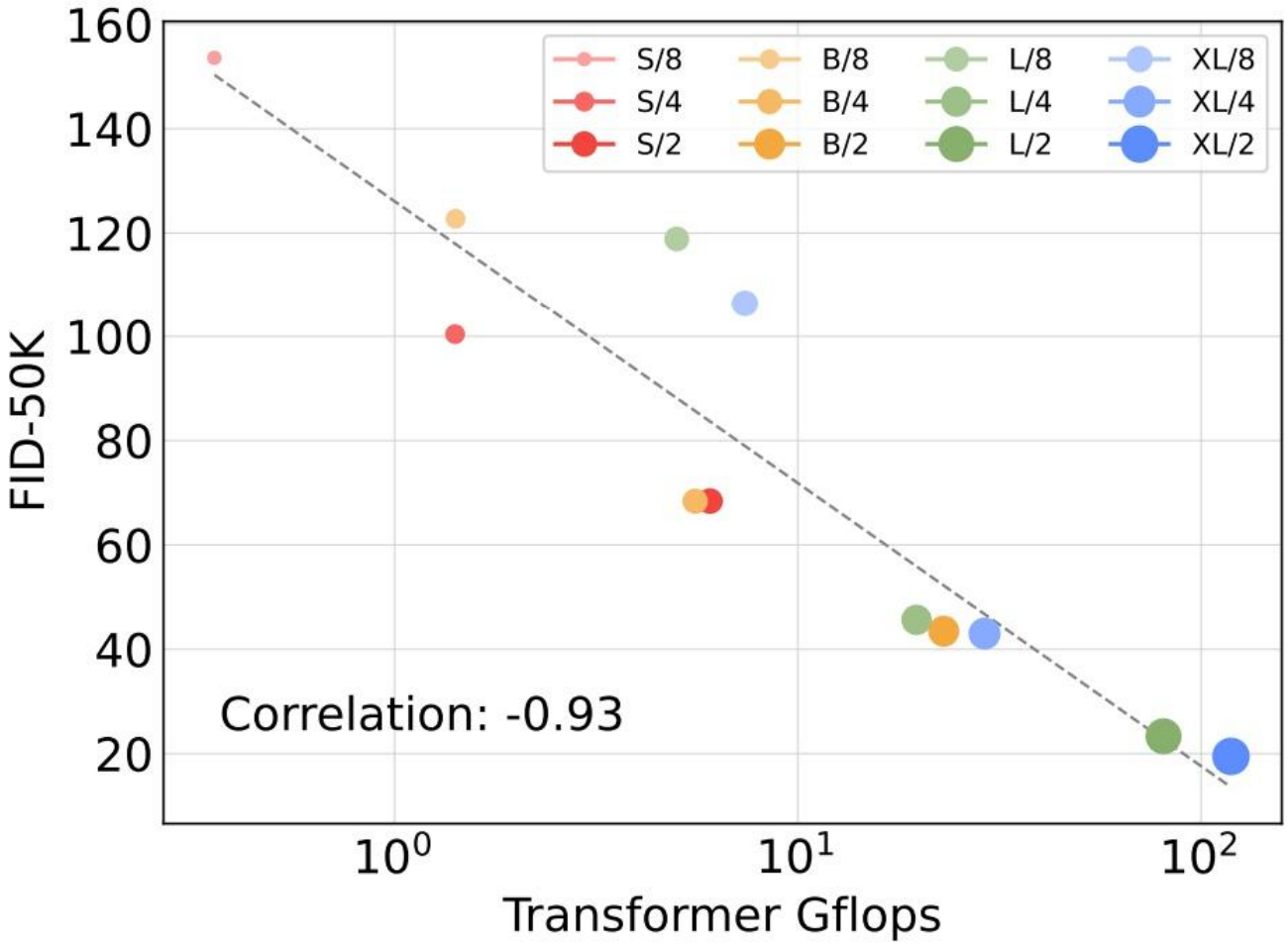

Figure 8. **Transformer Gflops are strongly correlated with FID.** We plot the Gflops of each of our DiT models and each model's FID-50K after 400K training steps.

在具体性能上，最大的模型DiT-XL/2采用classifier free guidance可以在class-conditional image generation on ImageNet 256×256任务上实现当时的sota。

Class-Conditional ImageNet 256×256					
Model	FID↓	sFID↓	IS↑	Precision↑	Recall↑
BigGAN-deep [2]	6.95	7.36	171.4	0.87	0.28
StyleGAN-XL [53]	2.30	4.02	265.12	0.78	0.53
ADM [9]	10.94	6.02	100.98	0.69	0.63
ADM-U	7.49	5.13	127.49	0.72	0.63
ADM-G	4.59	5.25	186.70	0.82	0.52
ADM-G, ADM-U	3.94	6.14	215.84	0.83	0.53
CDM [20]	4.88	-	158.71	-	-
LDM-8 [48]	15.51	-	79.03	0.65	0.63
LDM-8-G	7.76	-	209.52	0.84	0.35
LDM-4	10.56	-	103.49	0.71	0.62
LDM-4-G (cfg=1.25)	3.95	-	178.22	0.81	0.55
LDM-4-G (cfg=1.50)	3.60	-	247.67	**0.87**	0.48
DiT-XL/2	9.62	6.85	121.50	0.67	**0.67**
DiT-XL/2-G (cfg=1.25)	3.22	5.28	201.77	0.76	0.62
DiT-XL/2-G (cfg=1.50)	**2.27**	**4.60**	**278.24**	0.83	0.57

Table 2. **Benchmarking class-conditional image generation on ImageNet 256×256.** DiT-XL/2 achieves state-of-the-art FID.

Class-Conditional ImageNet 512×512					
Model	FID↓	sFID↓	IS↑	Precision↑	Recall↑
BigGAN-deep [2]	8.43	8.13	177.90	0.88	0.29
StyleGAN-XL [53]	2.41	4.06	267.75	0.77	0.52
ADM [9]	23.24	10.19	58.06	0.73	0.60
ADM-U	9.96	5.62	121.78	0.75	**0.64**
ADM-G	7.72	6.57	172.71	**0.87**	0.42
ADM-G, ADM-U	3.85	5.86	221.72	0.84	0.53
DiT-XL/2	12.03	7.12	105.25	0.75	**0.64**
DiT-XL/2-G (cfg=1.25)	4.64	5.77	174.77	0.81	0.57
DiT-XL/2-G (cfg=1.50)	**3.04**	**5.02**	**240.82**	0.84	0.54

Table 3. **Benchmarking class-conditional image generation on ImageNet 512×512.** Note that prior work [9] measures Precision and Recall using 1000 real samples for 512 × 512 resolution; for consistency, we do the same.

虽然DiT看起来不错，但是只在ImageNet上生成做了实验，并没有扩展到大规模的文生图模型。而且在DiT之前，其实也有基于transformer架构的扩散模型研究工作，比如U-ViT，目前也已经有将transformer应用在大规模文生图（基于扩散模型）的工作，比如UniDiffuser，但是其实都没有受到太大的关注。目前主流的文生图模型还是采用基于UNet，UNet本身也混合了卷积和attention，它的优势一方面是高效，另外一方面是不需要位置编码比较容易实现变尺度的生成，这些对具体落地应用都是比较重要的。

参考

- Scalable Diffusion Models with Transformers
- https://github.com/facebookresearch/DiT

- High-Resolution Image Synthesis with Latent Diffusion Models
- Improved Denoising Diffusion Probabilistic Models