Background:

1. unaffordable computational cost for majority

2. evaluation cost is quite expensive as well, need to run the model sequentially for a large numbers of steps .

Key to enhance DM accessibility: reduce the computational complexity without impairing their performance.

Contributions:

1. Ours can (a) work on a compression level which provides more faithful and detailed reconstructions than previous work (see Fig. 1) and (b) can be efficiently applied to high-resolution synthesis of megapixel images.
2. (ii) We achieve competitive performance on multipletasks (unconditional image synthesis, inpainting, stochastic super-resolution) and datasets while significantly lowering computational costs. Compared to pixel-based diffusion ap- proaches, we also significantly decrease inference costs.
3. (iii) Our approach does not require a delicate weighting of reconstruction and generative abilities. This ensures extremely faithful reconstructions and requires very little regularization of the latent space.
4. (iv) We find that for densely conditioned tasks such as super-resolution, inpainting and semantic synthesis, our model can be applied in a convolutional fashion and render large, consistent images of ~ 1024^2 px.
5. (v) Moreover, **we design a general-purpose conditioning mechanism based on cross-attention, enabling multi-modal training.** We use it to train class-conditional, text-to-image and layout-to-image models.
6. (vi) Finally, we release pretrained latent diffusion and autoencoding models at https : / / github . com/CompVis/latent-diffusion which might be reusable for a various tasks besides training of DMs [81].

Method:

1. Perceptional compression:
   - Model: an auto-encoder trained by a perceptual loss + path-based adversarial objective. (Enforce Local realism & avoid bluriness)
   - KL-reg & VQ-reg(avoid to generate high-variance Latency spaces) & 2D representation for mild compression (rather than 1D)
2. Latent diffusion model, neutral backbone is time-conditional UNet

$$L_{LDM} := \mathbb{E}_{\mathcal{E}(x),\epsilon\sim\mathcal{N}(0,1),t}\Big[\|\epsilon - \epsilon_\theta(z_t,t)\|_2^2\Big].$$

3. Conditioning mechanisms

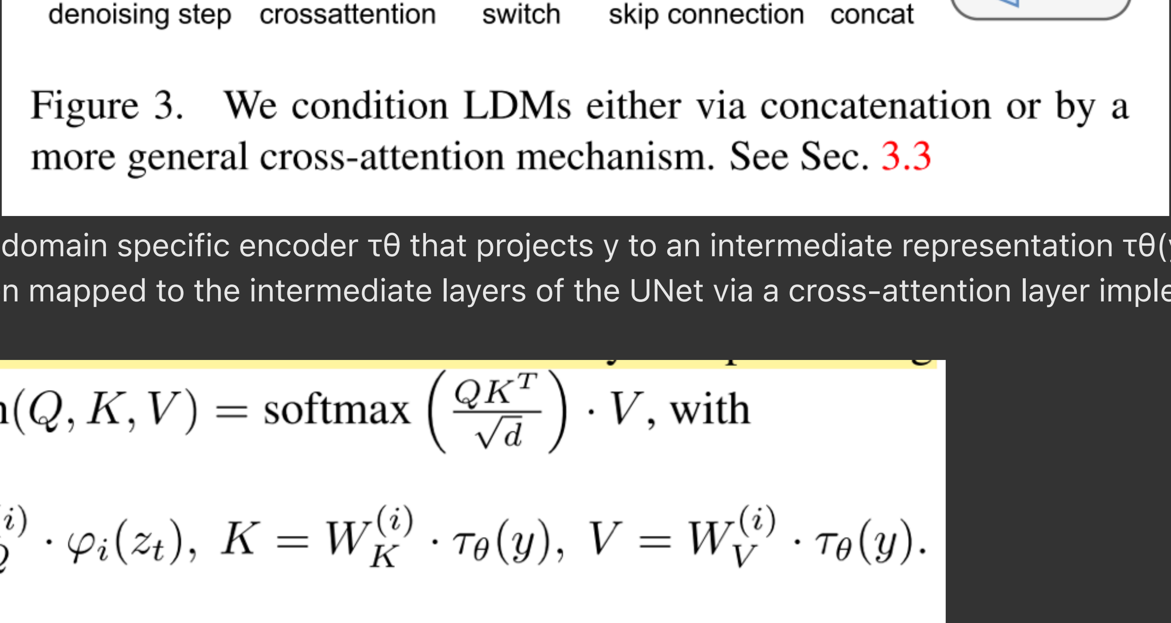


Figure 3. We condition LDMs either via concatenation or by a more general cross-attention mechanism. See Sec. 3.3

introduce a domain specific encoder τθ that projects y to an intermediate representation τθ(y) ∈ RM×dτ, which is then mapped to the intermediate layers of the UNet via a cross-attention layer implementing

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)\cdot V, \text{ with}$$

$$Q = W_Q^{(i)}\cdot\varphi_i(z_t),\ K = W_K^{(i)}\cdot\tau_\theta(y),\ V = W_V^{(i)}\cdot\tau_\theta(y).$$

Here, $\varphi_i(z_t) \in \mathbb{R}^{N\times d_\epsilon^i}$ denotes a (flattened) intermediate representation of the UNet implementing ϵ_θ and $W_V^{(i)} \in \mathbb{R}^{d\times d_\epsilon^i}$, $W_Q^{(i)} \in \mathbb{R}^{d\times d_\tau}$ & $W_K^{(i)} \in \mathbb{R}^{d\times d_\tau}$ are learnable projection matrices [36, 97]. See Fig. 3 for a visual depiction.

$$L_{LDM} := \mathbb{E}_{\mathcal{E}(x),y,\epsilon\sim\mathcal{N}(0,1),t}\Big[\|\epsilon-\epsilon_\theta(z_t,t,\tau_\theta(y))\|_2^2\Big], \quad (3)$$

where both τ_θ and ϵ_θ are jointly optimized via Eq. 3. This

Usage:
Image generation, super-resolution, image inpainting

Sample Codes from HuggingFace:

```
import torch
from diffusers import AutoencoderKL, UNet2DConditionModel, DDPMScheduler
from transformers import CLIPTextModel, CLIPTokenizer
import torch.nn.functional as F

# 加载autoencoder
vae = AutoencoderKL.from_pretrained("runwayml/stable-diffusion-v1-5", subfolder="vae")
# 加载text encoder
text_encoder = CLIPTextModel.from_pretrained("runwayml/stable-diffusion-v1-5", subfolder="text
tokenizer = CLIPTokenizer.from_pretrained("runwayml/stable-diffusion-v1-5", subfolder="tokeniz
# 初始化UNet
unet = UNet2DConditionModel(**model_config) # model_config为模型参数配置
# 定义scheduler
noise_scheduler = DDPMScheduler(
    beta_start=0.00085, beta_end=0.012, beta_schedule="scaled_linear", num_train_timesteps=100
)

# 冻结vae和text_encoder
vae.requires_grad_(False)
text_encoder.requires_grad_(False)

opt = torch.optim.AdamW(unet.parameters(), lr=1e-4)

for step, batch in enumerate(train_dataloader):
    with torch.no_grad():
        # 将image转到latent空间
        latents = vae.encode(batch["image"]).latent_dist.sample()
        latents = latents * vae.config.scaling_factor # rescaling latents
        # 提取text embeddings
        text_input_ids = tokenizer(
            batch["text"],
            padding="max_length",
            max_length=tokenizer.model_max_length,
            truncation=True,
            return_tensors="pt"
  ).input_ids
  text_embeddings = text_encoder(text_input_ids)[0] ## every batch has only one image-text pai

    # 随机采样噪音
    noise = torch.randn_like(latents)
    bsz = latents.shape[0]
    # 随机采样timestep
    timesteps = torch.randint(0, noise_scheduler.num_train_timesteps, (bsz,), device=latents.d
    timesteps = timesteps.long()

    # 将noise添加到latent上，即扩散过程
    noisy_latents = noise_scheduler.add_noise(latents, noise, timesteps)

    # 预测noise并计算loss
    model_pred = unet(noisy_latents, timesteps, encoder_hidden_states=text_embeddings).sample
    loss = F.mse_loss(model_pred.float(), noise.float(), reduction="mean")

 opt.step()
    opt.zero_grad()
Where, beta is nonlinear.

  betas = torch.linspace(beta_start**0.5, beta_end**0.5, num_train_timesteps, dtype=to
```

训练扩散模型时， 采用CFG（conditional free guidance）

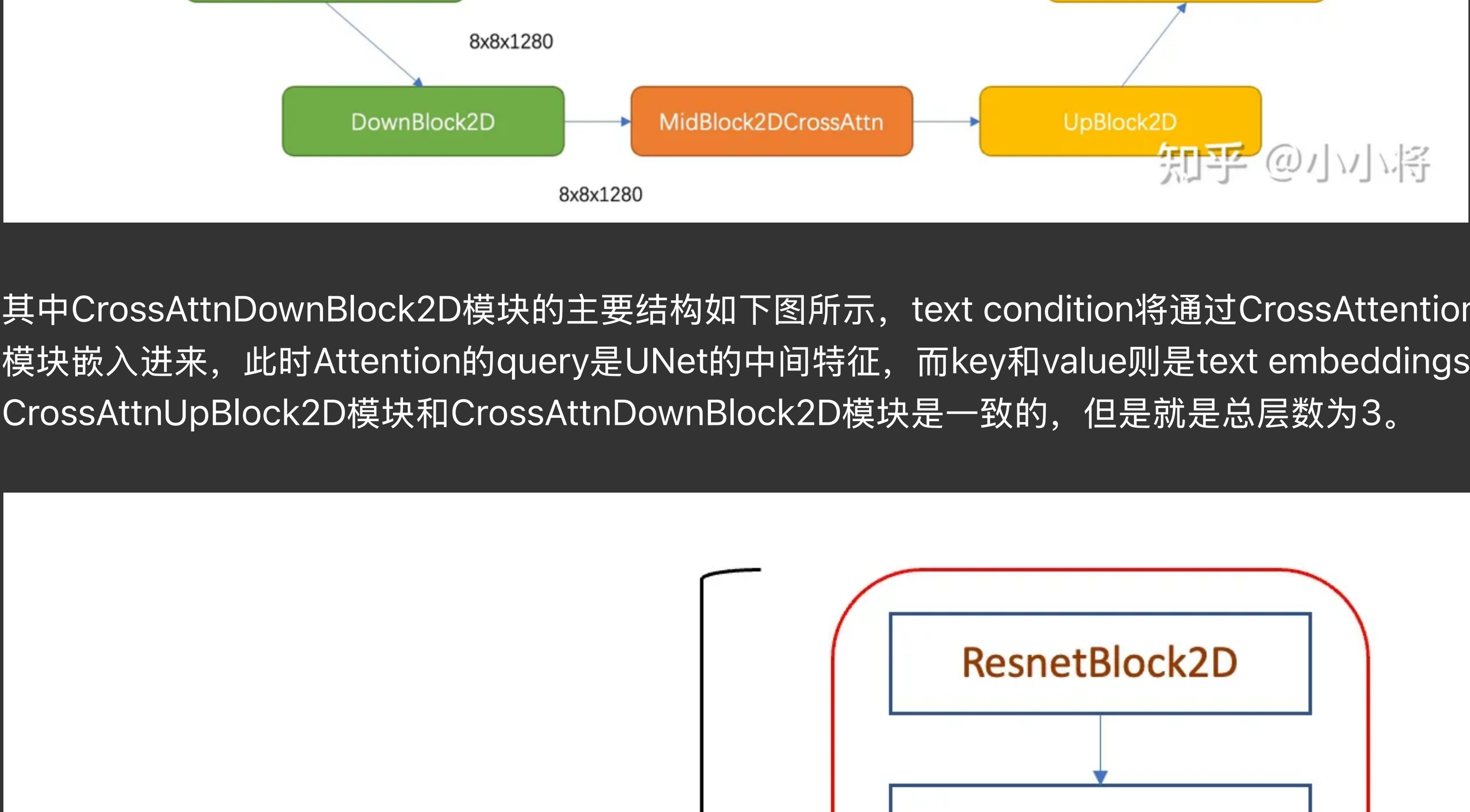
$$\begin{aligned}\text{pred\_noise} &= w\cdot\text{cond\_pred\_noise} + (1-w)\cdot\text{uncond\_pred\_noise}\\ &= w\epsilon_\theta(\mathbf{x}_t,t,\mathbf{c}) + (1-w)\epsilon_\theta(\mathbf{x}_t,t)\end{aligned}$$

[HuggingFace— Stable diffusionv1.5]

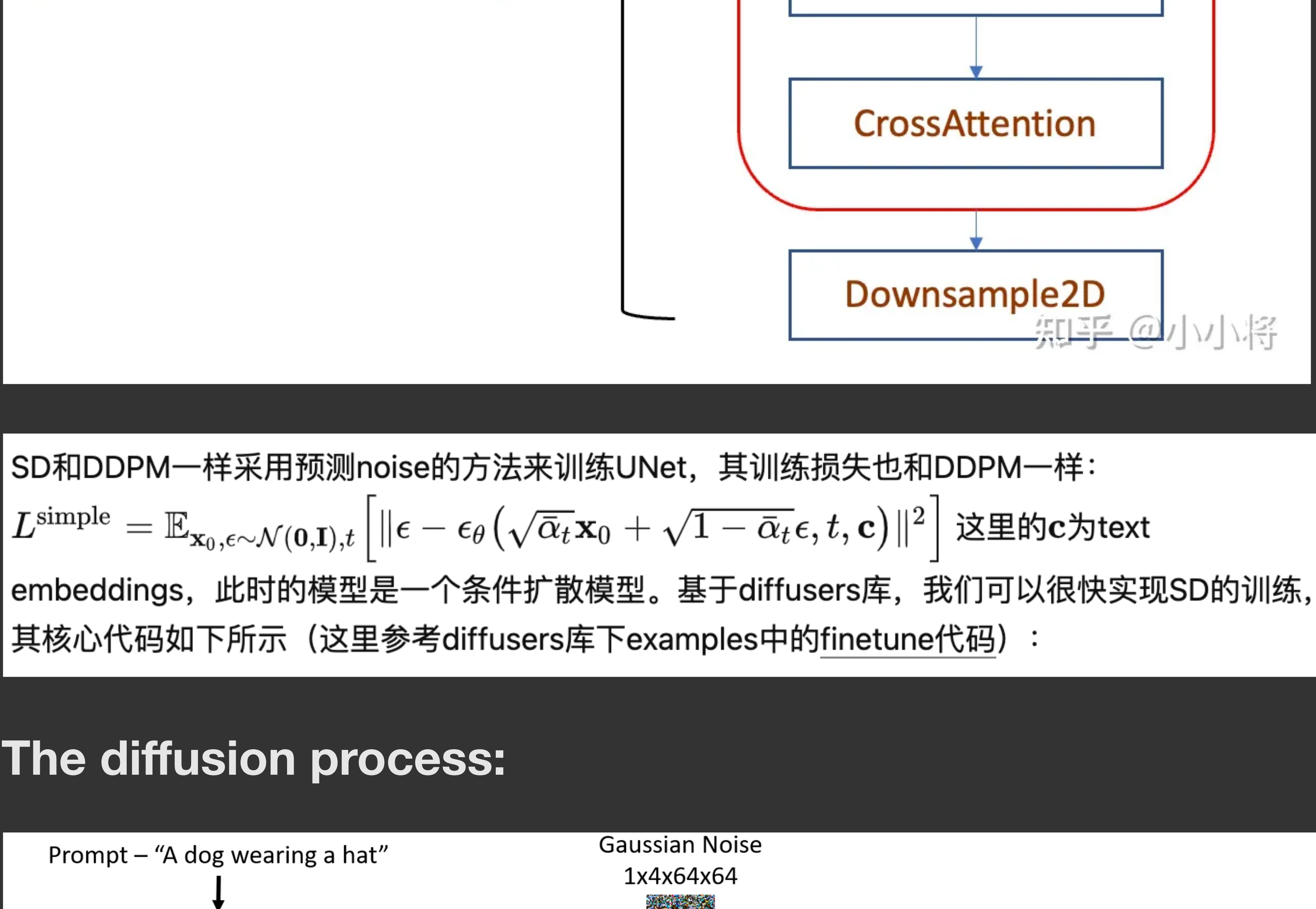
[文生图模型之Stable Diffusion]

[Diffusers: State-of-the-art diffusion models for image and audio generation in PyTorch]

UNET

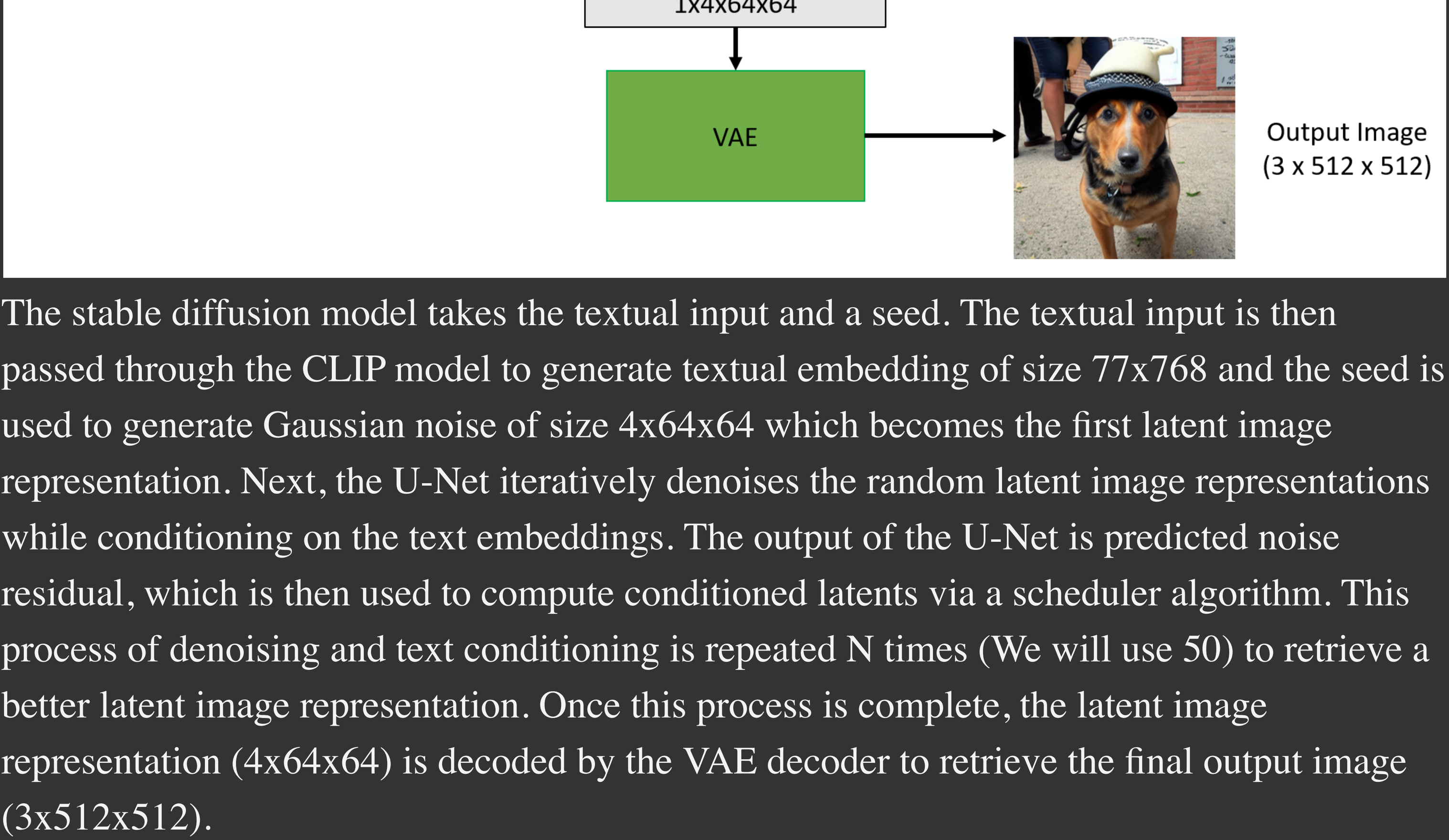


其中CrossAttnDownBlock2D模块的主要结构如下图所示，text condition将通过CrossAttention模块嵌入进来，此时Attention的query是UNet的中间特征，而key和value则是text embeddings。CrossAttnUpBlock2D模块和CrossAttnDownBlock2D模块是一致的，但是就是总层数为3。



SD和DDPM一样采用预测noise的方法来训练UNet，其训练损失也和DDPM一样：$L^{\text{simple}} = \mathbb{E}_{\mathbf{x}_0,\epsilon\sim\mathcal{N}(\mathbf{0},\mathbf{I}),t}\Big[\|\epsilon - \epsilon_\theta\big(\sqrt{\bar\alpha_t}\mathbf{x}_0 + \sqrt{1-\bar\alpha_t}\epsilon, t, \mathbf{c}\big)\|^2\Big]$ 这里的$\mathbf{c}$为text embeddings，此时的模型是一个条件扩散模型。基于diffusers库，我们可以很快实现SD的训练，其核心代码如下所示（这里参考diffusers库下examples中的finetune代码）：

The diffusion process:



The stable diffusion model takes the textual input and a seed. The textual input is then passed through the CLIP model to generate textual embedding of size 77x768 and the seed is used to generate Gaussian noise of size 4x64x64 which becomes the first latent image representation. Next, the U-Net iteratively denoises the random latent image representations while conditioning on the text embeddings. The output of the U-Net is predicted noise residual, which is then used to compute conditioned latents via a scheduler algorithm. This process of denoising and text conditioning is repeated N times (We will use 50) to retrieve a better latent image representation. Once this process is complete, the latent image representation (4x64x64) is decoded by the VAE decoder to retrieve the final output image (3x512x512).