Practical Implementation Details

In previous chapters, we have treated the score estimator $\widehat{s}(x,t) \approx \nabla \log p_t(x)$ as an abstract function from a concept class $\mathcal{F}$. In practice, diffusion models succeed largely because $\mathcal{F}$ is instantiated by carefully designed neural architectures that can represent *multi-scale* denoising maps and can be *conditioned* on the noise level (time) in a stable way. By primarily focusing on image-based diffusion models, we introduce several practical aspects: (i) why most large-scale image diffusion models operate in a *latent space*, (ii) how the score network is implemented in practice (U-Nets and Transformer-style backbones), and (iii) how *time embeddings* inject diffusion-time information into these networks.

1 Latent Diffusion Models (LDMs)

The key practical bottleneck in pixel-space diffusion is computational: a score network must repeatedly process high-dimensional tensors (e.g., 512×512 or 1024×1024 RGB images) for hundreds of denoising steps. Latent diffusion models [4] address this by inserting a learned *autoencoder*. The encoder down scales the input image by

$$X \in \mathbb{R}^{H \times W \times 3} \xrightarrow{\text{encoder } E} Z \in \mathbb{R}^{h \times w \times c} \quad (\text{typically } h = H/f,\ w = W/f),$$

and a diffusion model is then trained on the lower-dimensional latent z. A typical choice of the downsampling factor is $f = 8$. The dimension c is the number of output channels in the autoencoder; a canonical choice is $c = 4$.

Sampling in latent diffusion model proceeds as

$$Z_T \sim \mathsf{N}(0, I) \xrightarrow{\text{reverse diffusion in latent}} Z_0 \xrightarrow{\text{decoder } D} \widehat{X}.$$

Conceptually, the diffusion model is unchanged; only the state space is different. Practically, this yields a large speedup and enables higher resolutions with comparable compute. This design is the backbone of popular open-source systems such as Stable Diffusion.

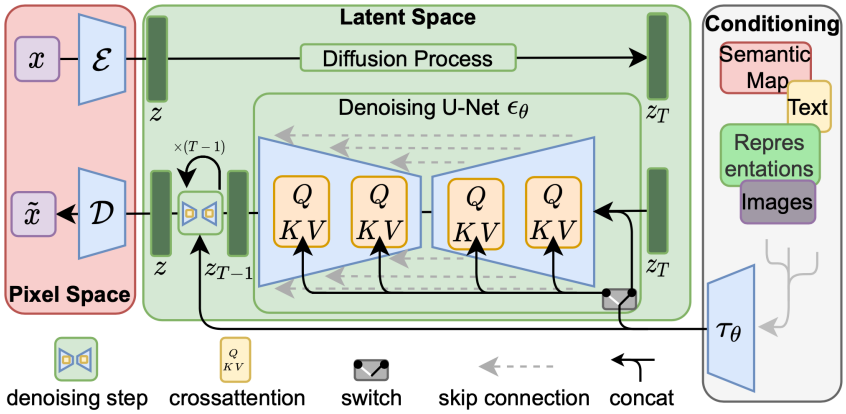

Figure 1: Latent diffusion model architecture in [4]. Data in the pixel space are transformed into a latent space first. In the latent space, a diffusion model is trained.

2 Score Network Architectures

In continuous-time formulations, the score network aims to approximate $s(x,t) \approx \nabla \log p_t(x)$ (or an equivalent parameterization such as noise-prediction). In implementations, the network takes as input:

$$\text{noisy (latent) image} \quad x_t \quad \text{and} \quad \text{time index} \quad t.$$

Two dominant architecture families are U-Net backbones [5] and Transformer-style diffusion backbones [2].

2.1 U-Net Score Networks

A diffusion U-Net is a convolutional network with skip connections across resolutions. They consist of several important elements:

- **Down path:** progressively downsample to low resolution.
- **Up path:** progressively upsample to recover spatial detail.
- **Skip connections:** direct interaction between the down path and up path.

Modern diffusion U-Nets also incorporate *attention blocks* (self-attention and/or cross-attention), which substantially improves global coherence and enables text-conditioned image generation.

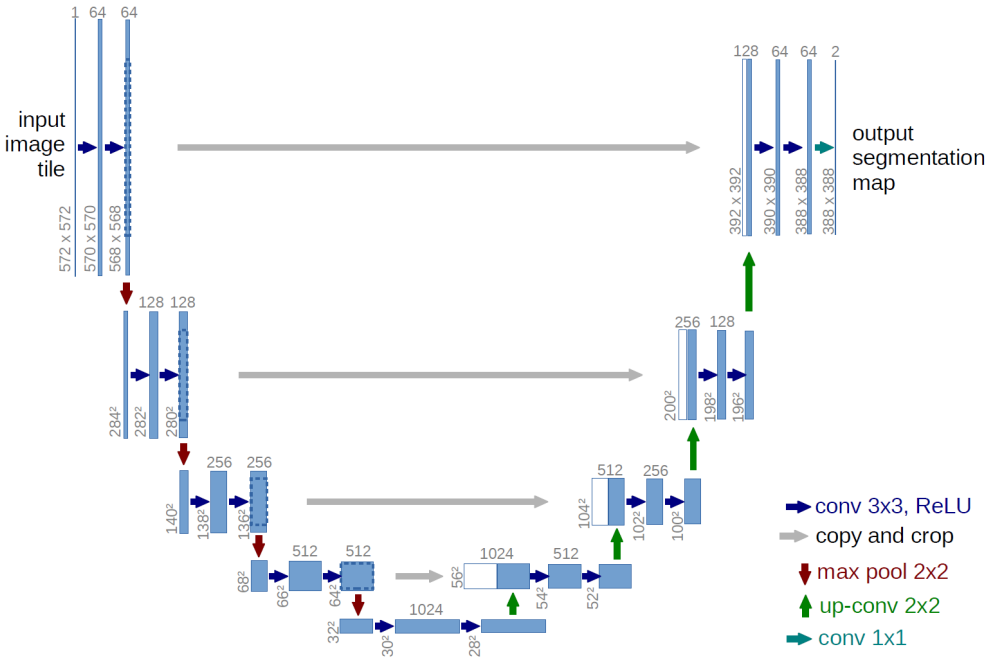

Figure 2: U-Net architecture in [5].

Scale and parameter counts. A typical Stable Diffusion v1.4/1.5 configuration uses a latent-space U-Net with about 860M parameters, coupled with text encoder for conditional generation. Stable Diffusion Extra Large (SDXL) scales the U-Net further (2.6B parameters) and also expands the text-encoding stack.

System	Model size (not accurate)	FID (reference only, not accurate)
Imagen [6]	$\approx$ 2B	COCO zero-shot FID: 7.27
DALL·E 2	$\approx$ 3.5B	COCO zero-shot FID: 10.39
GLIDE [1]	$\approx$ 3.5B	COCO zero-shot FID: 12.24
Stable Diffusion	$\approx$ 860M	(not standardized in release)
SDXL [3]	$\approx$ 2.6B	(varies by protocol)

Table 1: A high-level comparison of popular diffusion systems. FID values are listed only when stated in the corresponding primary source.

2.2 Transformer Score Networks

An alternative is to replace the U-Net with a Transformer-like backbone, often by representing the image/latent as a sequence of patches/tokens. These models can be competitive at scale, especially when trained on very large data sets [2]. Indeed, U-Nets remain a robust default for high-quality image synthesis with limited compute; transformer-style diffusion becomes increasingly attractive when training scale and data scale are very large and one wants a uniform sequence model backbone. In recent applications, transformer-based diffusion models are slightly preferred and demonstrate the capability in highly complex generation tasks such as video generation.

The key idea in transformer-based diffusion models is to (i) *patchify* the input image into tokens, (ii) run a stack of transformer blocks, and (iii) *unpatchify* back to the spatial image.

Patchification. Let $x_t \in \mathbb{R}^{H\times W\times C}$ denote a noisy image/latent at time t. Choose a patch size p and reshape into $N = (H/p)(W/p)$ patches. Each patch is flattened and linearly projected into a token:

$$x_t \;\mapsto\; \{u_i\}_{i=1}^{N}, \qquad u_i \in \mathbb{R}^{p^2C}, \qquad y_i = W_{\text{in}}u_i \in \mathbb{R}^d.$$

Positional information is added via fixed or learned embeddings (e.g., 2D sin-cos or learned position embeddings), giving the initial token sequence $\{y_i^{(0)}\}_{i=1}^{N}$.

Transformer blocks. A transformer comprises a series of blocks and each block encompasses a *multi-head attention* layer and a *feedforward* layer. Let $Y = [y_1, \ldots, y_N] \in \mathbb{R}^{d\times N}$ be the (column) stacking matrix of N patches. In a transformer block, multi-head attention computes

$$\texttt{Attn}(Y) = Y + \sum_{m=1}^{M} V^m Y \cdot \sigma\left((Q^m Y)^\top K^m Y\right), \tag{2.1}$$

where V^m, Q^m, and K^m are weight matrices of corresponding sizes in the m-th attention head, and σ is an activation function. The attention layer is followed by a feedforward layer, which computes

$$\texttt{FFN}(Y) = Y + W_1 \cdot \mathrm{ReLU}(W_2 Y + b_2\mathbf{1}^\top) + b_1\mathbf{1}^\top.$$

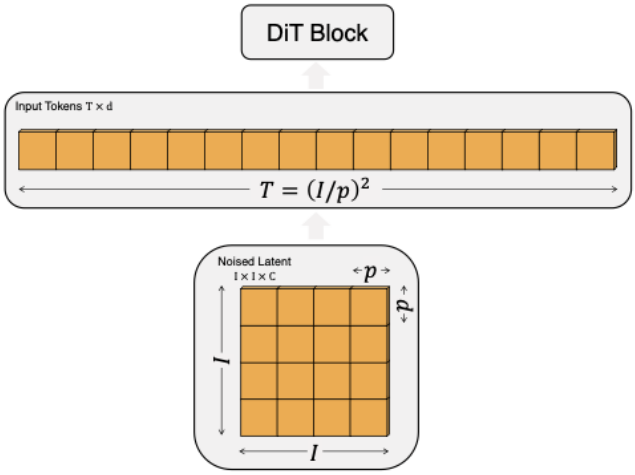

Figure 3: An example of patchification in diffusion transformer [2].

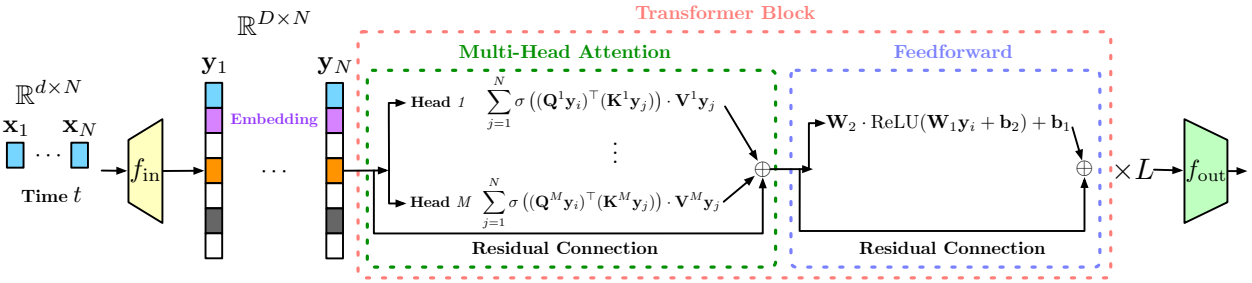

Figure 4: Demonstration of transformer architecture.

Here, W_1, W_2 are weight matrices, b_1 and b_2 are offset vectors, $\mathbf{1}$ denotes a vector of ones, and the ReLU activation function is applied entrywise.

The attention defined in (2.1) is often called self-attention. For text-conditioned generation, one can modify the self-attention to cross-attention. In particular, the text prompt will be mapped to tokens from a separate text encoder. We denote the text token as T. Then we modify the multi-head attention as

$$\mathrm{Attn}(Y,T) = Y + \sum_{m=1}^{M} V^m T \cdot \sigma\left((Q^m Y)^\top K^m T\right).$$

3 Time Embedding: Injecting Diffusion Time into Networks

In almost all implementations of the score network, the scalar time t (or an integer timestep index) is first mapped to a vector embedding, then injected into the score network *at many layers*. We describe the standard mechanism used in diffusion U-Nets.

Step 1: Fourier/sinusoidal timestep features. A common first step is to transform the time t into a sinusoidal (Fourier-feature) embedding, analogous to positional encodings in Transformers.

We fix an even dimension m and define

$$\phi(t) = \big(\sin(\omega_1 t), \cos(\omega_1 t), \ldots, \sin(\omega_{m/2} t), \cos(\omega_{m/2} t)\big) \in \mathbb{R}^m,$$

where frequencies are typically log-spaced, e.g.

$$\omega_k = \omega_{\min} \Big(\frac{\omega_{\max}}{\omega_{\min}}\Big)^{\frac{k-1}{m/2-1}} \quad \text{for } k = 1, \ldots, m/2.$$

(For discrete timestep index $n \in \{0, \ldots, T\}$, we can use $\phi(n)$; the mechanism is the same.)

Step 2: MLP produces a global time embedding. The raw Fourier features are then passed through a small MLP (two linear layers with a nonlinearity) to produce a *global* time embedding,

$$e(t) = \mathrm{MLP}(\phi(t)) \in \mathbb{R}^{d_t}.$$

Typical choices are $m = 128$ or 256, and d_t matching the channel width used inside the score network.

Step 3: Embedding $e(t)$ is used inside a diffusion U-Net. Modern diffusion U-Nets are largely built from *time-conditioned residual blocks*, with occasional attention blocks inserted at selected resolutions. The standard injection mechanism is:

1. Convert $e(t)$ into channel-wise parameters via a learned linear map. For example, we compute

   $$(\gamma(t), \beta(t)) = W_t e(t),$$

   where W_t is learnable.

2. Use $\gamma(t), \beta(t)$ to modulate normalization layers. The detailed implementation is complicated and may vary among different networks.

4 Code Bases (PyTorch) and a Minimal Text-to-Image Example

In practice, most readers will interact with diffusion models through mature PyTorch code bases rather than re-implementing training and sampling from scratch. Below we list several widely used repositories and then provide a minimal text-to-image example that illustrates the end-to-end objects discussed in this note: VAE (latent space), denoiser (U-Net/Transformer), scheduler (sampler), and conditioning (text encoder + cross-attn).

4.1 Recommended code bases and documentation (URLs)

- **Hugging Face `diffusers` (PyTorch).** A modular library covering diffusion pipelines (text-to-image, img2img, inpainting), schedulers (DDPM/DDIM/Euler/Heun/DPM-Solver), U-Net backbones, ControlNet/LoRA add-ons, and training utilities.
  `https://github.com/huggingface/diffusers`
  `https://huggingface.co/docs/diffusers/index`

- **OpenAI guided diffusion (reference implementation for diffusion training/sampling).** `https://github.com/openai/guided-diffusion`
- **DiT (Diffusion Transformers) reference code (Transformer diffusion backbones).** `https://github.com/facebookresearch/DiT`

4.2 A minimal PyTorch text-to-image example (`diffusers`)

The following example demonstrates the standard components:

- a *text encoder* produces text token embeddings T;
- a *latent* z_t is evolved by a *scheduler*;
- a *denoiser* (typically a latent U-Net with self-/cross-attention) predicts $\widehat{\epsilon}(z_t, t, T)$;
- a *VAE decoder* maps z_0 back to pixels.

This is the simplest path to reproduce text-to-image sampling in a modern stack.

Installation (one-time).

```
pip install -U diffusers transformers accelerate safetensors torch
```

Inference script (text-to-image).

```
import torch
from diffusers import StableDiffusionPipeline

# Pick a model repo / checkpoint on the Hugging Face Hub.
# You may need to accept the model's license terms on its model card page.
model_id = "runwayml/stable-diffusion-v1-5"

device = "cuda" if torch.cuda.is_available() else "cpu"
dtype = torch.float16 if device == "cuda" else torch.float32

pipe = StableDiffusionPipeline.from_pretrained(
    model_id,
    torch_dtype=dtype,
)
pipe = pipe.to(device)

prompt = "a photo of a corgi wearing sunglasses, cinematic lighting"
neg_prompt = "blurry, low quality"  # optional negative prompt
guidance_scale = 7.5                # classifier-free guidance strength
num_steps = 30                      # sampling steps
```

```
# Run the pipeline. Internally this executes:
# (i) text encoding, (ii) latent noise init z_T,
# (iii) iterative scheduler updates using the denoiser UNet,
# (iv) VAE decoding back to pixels.
image = pipe(
    prompt=prompt,
    negative_prompt=neg_prompt,
    guidance_scale=guidance_scale,
    num_inference_steps=num_steps,
).images[0]

image.save("sample.png")
print("Saved to sample.png")
```

References

[1] Alex Nichol, Prafulla Dhariwal, Aditya Ramesh, Pranav Shyam, Pamela Mishkin, Bob McGrew, Ilya Sutskever, and Mark Chen. Glide: Towards photorealistic image generation and editing with text-guided diffusion models. *arXiv preprint arXiv:2112.10741*, 2021.

[2] William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 4195–4205, 2023.

[3] Dustin Podell, Zion English, Kyle Lacey, Andreas Blattmann, Tim Dockhorn, Jonas Müller, Joe Penna, and Robin Rombach. Sdxl: Improving latent diffusion models for high-resolution image synthesis. *arXiv preprint arXiv:2307.01952*, 2023.

[4] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022.

[5] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[6] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily L Denton, Kamyar Ghasemipour, Raphael Gontijo Lopes, Burcu Karagol Ayan, Tim Salimans, et al. Photorealistic text-to-image diffusion models with deep language understanding. *Advances in neural information processing systems*, 35:36479–36494, 2022.