MO433 - Unsupervised Learning
Image Generation by Stable Diffusion

Alexandre Xavier Falcão

Institute of Computing - UNICAMP

afalcao@ic.unicamp.br

From Diffusion Models to Stable Diffusion

We learned diffusion models that work directly in pixel space.

Challenge with pixel-space diffusion

High-resolution images (e.g., $512 \times 512 \times 3$) require enormous computational resources.

- Memory: $\sim$768K dimensions per image.
- Training: Days/weeks on multiple high-end GPUs.
- Inference: Slow generation (1000 denoising steps).

From Diffusion Models to Stable Diffusion

We learned diffusion models that work directly in pixel space.

Challenge with pixel-space diffusion

High-resolution images (e.g., $512 \times 512 \times 3$) require enormous computational resources.

- Memory: $\sim$768K dimensions per image.
- Training: Days/weeks on multiple high-end GPUs.
- Inference: Slow generation (1000 denoising steps).

Solution: Stable Diffusion (Rombach et al., 2022) perform diffusion in a compressed latent space instead of pixel space!

$$\text{Pixel Space } (512 \times 512 \times 3) \xrightarrow{\text{VAE}} \text{Latent Space } (64 \times 64 \times 4).$$

Result: $48\times$ smaller dimensionality $\Rightarrow$ Much faster training and inference!

Stable Diffusion = Latent Diffusion + Text Conditioning

Component 1: VAE (Variational Autoencoder)
Encoder: Compress images to latent space

$$\begin{aligned}
\text{Image } \mathrm{x} &\in \mathbb{R}^{H\times W\times 3} \\
&\downarrow \text{ Conv layers (stride 2, multiple times)} \\
\text{Features } \mathrm{h} &\in \mathbb{R}^{h\times w\times C} \quad (\text{e.g., } h = \frac{H}{8}, w = \frac{W}{8}) \\
&\downarrow \text{ Flatten + Linear projections} \\
\boldsymbol{\mu} &\in \mathbb{R}^{d} \quad \text{(mean vector)} \\
\log \boldsymbol{\sigma}^2 &\in \mathbb{R}^{d} \quad \text{(log variance vector)}
\end{aligned}$$

Reparameterization trick to sample latent:

$$\mathrm{z}_0 = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathrm{I})$$

Then reshape z_0 back to spatial: $\mathrm{z}_0 \in \mathbb{R}^{h\times w\times 4}$
Decoder: Reconstruct from latents: $\mathbb{R}^{h\times w\times 4} \to \mathbb{R}^{H\times W\times 3}$

Stable Diffusion = Latent Diffusion + Text Conditioning

Component 2: UNet (Denoising Network)

- Operates in latent space (not pixel space!)
- Predicts noise: $\epsilon_\theta(\mathrm{z}_t, t, \mathrm{c})$
  - Input: Noisy latent z_t at timestep t
  - Output: Predicted noise ϵ
- Conditioned on text via cross-attention
- **Much faster** than pixel-space diffusion (8× smaller!)

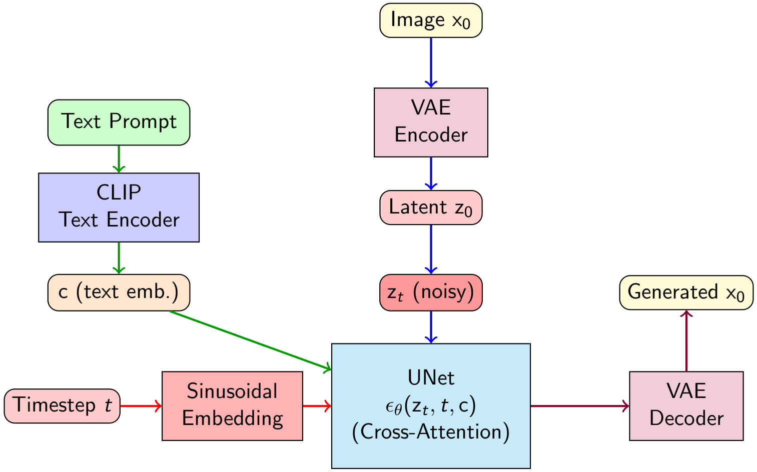
Component 3: CLIP Text Encoder

- Converts text prompts to embeddings:

$$\mathrm{c} = \mathrm{CLIP}(\text{“a photo of a royal guard”}) \in \mathbb{R}^{77\times768}$$

- Guides generation through cross-attention in UNet
- Enables text-to-image synthesis

Stable Diffusion: Architecture Overview



Training: Compress to latents, add noise, learn $\epsilon_\theta(z_t, t, c)$ to predict noise.

Inference: Start from $z_T \sim \mathcal{N}(0, I)$, iteratively denoise guided by text and timestep.

Why Stable Diffusion Works Better

Pixel-Space Diffusion:

Disadvantages:

- High dimensional (786,432D)
- Slow training
- Slow inference
- Expensive memory

Stable Diffusion:

Advantages:

- Low dimensional (16,384D)
- Fast training (48$\times$ faster)
- Fast inference
- Memory efficient
- High quality via VAE

Insight

The VAE learns to compress images into a perceptually meaningful latent space, discarding imperceptible details while preserving semantic information.

Agenda

- VAE: Compressing images to latent space.
- Text Conditioning with CLIP.
- Cross-Attention: Injecting text into UNet.
- Training Stable Diffusion.
- Inference and Sampling.
- Applications and fine-tuning.

VAE: Compressing Images to Latent Space

Variational Autoencoder (VAE) compresses images while preserving perceptual information.

Encoder ϕ: Maps image to latent distribution

$$q_\phi(\mathrm{z}|\mathrm{x}) = \mathcal{N}(\mathrm{z}; \mu_\phi(\mathrm{x}), \sigma^2_\phi(\mathrm{x}) I)$$

Sample latent: $\mathrm{z} = \mu_\phi(\mathrm{x}) + \sigma_\phi(\mathrm{x}) \odot \epsilon, \quad \epsilon \sim \mathcal{N}(0, I)$

Decoder ψ: Reconstructs image from latent

$$p_\psi(\mathrm{x}|\mathrm{z}) = \mathcal{N}(\mathrm{x}; \mu_\psi(\mathrm{z}), I)$$

Training objective (negative ELBO - Evidence Lower Bound):

$$\mathcal{L}_{\mathrm{VAE}} = \underbrace{\|\mathrm{x} - \hat{\mathrm{x}}\|^2}_{\text{reconstruction}} + \underbrace{\beta \cdot \mathrm{KL}(q_\phi(\mathrm{z}|\mathrm{x}) \| p(\mathrm{z}))}_{\text{regularization}}$$

where $p(\mathrm{z}) = \mathcal{N}(0, I)$ and β controls the trade-off.

VAE Architecture for Stable Diffusion

Scaling factor: Latents are scaled by $s = 0.18215$ for numerical stability.

Encoder: $\phi : \mathbb{R}^{512\times512\times3} \to \mathbb{R}^{64\times64\times4}$

- Downsampling factor: $8\times$ in spatial dimensions.
- Compression ratio: $48\times$ (from 786,432 to 16,384 dimensions).
- Architecture: Convolutional layers with residual blocks.

Encoding process:

$$\mathrm{x} \in [-1,1]^{512\times512\times3} \quad \text{(normalized image)}.$$
$$\mathrm{x}' = \phi(\mathrm{x}) \in \mathbb{R}^{64\times64\times4}.$$
$$[\mu_\phi(\mathrm{x}), \log\sigma^2_\phi(x)] \leftarrow [\mathit{Linear}(\mathrm{x}'), \mathit{Linear}(\mathrm{x}')]$$
$$\mathrm{z} \leftarrow s \cdot [\mu_\phi(\mathrm{x}) + \sigma_\phi(\mathrm{x}) \odot \epsilon] \quad \text{(for diffusion)}.$$

Decoder: $\psi : \mathbb{R}^{64\times64\times4} \to \mathbb{R}^{512\times512\times3}$

$$\hat{\mathrm{x}} = \psi(\mathrm{z}/s) \in [-1,1]^{512\times512\times3}.$$

VAE Properties

Properties that make VAE suitable for diffusion:

1. **Perceptual compression:**
   - Removes imperceptible high-frequency details.
   - Preserves semantic and structural information.
   - Low reconstruction error: $\|x - \hat{x}\|^2 < 0.01$.
2. **Smooth latent space:**
   - Similar images map to nearby latents.
   - Enables smooth interpolation.
   - Suitable for Gaussian diffusion.
3. **Computational efficiency:**
   - $48\times$ fewer dimensions than pixels.
   - Faster forward/backward passes in UNet.
   - Lower memory requirements.

Trade-off: Slight loss of fine details (assuming large training set) vs. massive speedup.

Text Conditioning with CLIP

CLIP (Contrastive Language-Image Pre-training) encodes text into semantic embeddings.

Architecture: Transformer-based text encoder, pre-trained on 400M image-text pairs, maps text to fixed-dimensional space: $\mathbb{R}^{77\times 768}$.

Text encoding process:

$$\text{text} \xrightarrow{\text{Tokenizer}} [t_1, t_2, \ldots, t_{77}] \in \mathbb{N}^{77}$$

$$[t_1, \ldots, t_{77}] \xrightarrow{\text{Embedding}} [\mathrm{e}_1, \ldots, \mathrm{e}_{77}] \in \mathbb{R}^{77\times 768}$$

$$[\mathrm{e}_1, \ldots, \mathrm{e}_{77}] \xrightarrow{\text{Transformer}} \mathrm{c} \in \mathbb{R}^{77\times 768}$$

Properties:

- Maximum sequence length: 77 tokens.
- Padding/truncation for variable-length prompts.
- Similar meanings $\Rightarrow$ similar embeddings.

Classifier-Free Guidance

Problem: How to control the strength of text conditioning?

Solution: Classifier-Free Guidance (CFG) - interpolate between conditional and unconditional predictions.

Training: Randomly drop text conditioning ($p = 0.1$)

$$\epsilon_\theta(z_t, t, c) \quad \text{and} \quad \epsilon_\theta(z_t, t, \emptyset)$$

Inference: Combine both predictions

$$\tilde{\epsilon}_\theta = \underbrace{\epsilon_\theta(z_t, t, \emptyset)}_{\text{unconditional}} + w \cdot \underbrace{(\epsilon_\theta(z_t, t, c) - \epsilon_\theta(z_t, t, \emptyset))}_{\text{guidance direction}}$$

where $w > 0$ is the guidance scale.

Effects of guidance scale:

- $w = 1$: Standard conditional generation.
- $w > 1$: Stronger adherence to prompt (typical: $w = 7.5$).
- $w < 1$: Weaker conditioning, more creative.

UNet with Cross-Attention

Modified ResBlock with cross-attention:

$$\begin{aligned}
\mathrm{h}^{(0)} &= \mathrm{ResBlock}(\mathrm{z}_t, \mathrm{e}_t) \quad \text{(spatial conv + time embed)} \\
\mathrm{h}^{(1)} &= \mathrm{SelfAttn}(\mathrm{h}^{(0)}) + \mathrm{h}^{(0)} \quad \text{(self-attention)} \\
\mathrm{h}^{(2)} &= \mathrm{CrossAttn}(\mathrm{h}^{(1)}, \mathrm{c}) + \mathrm{h}^{(1)} \quad \text{(text conditioning)} \\
\mathrm{h}^{(3)} &= \mathrm{FFN}(\mathrm{h}^{(2)}) + \mathrm{h}^{(2)} \quad \text{(feed-forward)}
\end{aligned}$$

Three inputs to UNet:

1. z_t: Noisy latent at timestep t.
2. t: Timestep (via sinusoidal embedding e_t).
3. c: Text embeddings from CLIP.

Output: Predicted noise $\epsilon_\theta(\mathrm{z}_t, t, \mathrm{c})$.

Key: Apply attention only at lower resolutions ($\leq 32 \times 32$ pixels).

Attention Mechanism: Core Concept

Goal: Allow each position to attend to relevant information.

Single-Head Attention:

$$
\begin{aligned}
\mathrm{Q} &= \mathrm{ZW}_Q \in \mathbb{R}^{n \times d_k} \quad &\text{(queries)} \\
\mathrm{K} &= \mathrm{ZW}_K \in \mathbb{R}^{n \times d_k} \quad &\text{(keys)} \\
\mathrm{V} &= \mathrm{ZW}_V \in \mathbb{R}^{n \times d_v} \quad &\text{(values)}
\end{aligned}
$$

Typically: $d_k = d_v$.

$$\mathrm{Attention}(\mathrm{Q}, \mathrm{K}, \mathrm{V}) = \mathrm{softmax}\left(\frac{\mathrm{QK}^T}{\sqrt{d_k}}\right)\mathrm{V}$$

- $\mathrm{QK}^T \in \mathbb{R}^{n \times n}$: attention scores (who attends to whom).
- Softmax: normalize scores to probabilities.
- Multiply by V: weighted combination of values.

Limitation: Single attention pattern solved by multiple attention heads.

Multi-Head Attention

Run h attention operations in parallel (different "heads").

For each head $i = 1, \ldots, h$:

1. **Project to head i** using independent W_Q^i, W_K^i, W_V^i.

$$Q_i = ZW_Q^i \in \mathbb{R}^{n \times d_k}$$
$$K_i = ZW_K^i \in \mathbb{R}^{n \times d_k}$$
$$V_i = ZW_V^i \in \mathbb{R}^{n \times d_v}$$

2. **Compute attention:**

$$\text{head}_i = \text{softmax}\left(\frac{Q_i K_i^T}{\sqrt{d_k}}\right) V_i \in \mathbb{R}^{n \times d_v}$$

Combine all heads:

$$\text{MultiHead}(Z) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h) W^O \in \mathbb{R}^{n \times d}$$

where $W^O \in \mathbb{R}^{(h \cdot d_v) \times d}$ mixes information across heads - e.g., $d_k = d_v = d/h$ so concatenation gives dimension d (from the model.

Self-Attention in UNet: Spatial Coherence

Goal: Allow spatial locations (pixels) to attend to each other.

Self-Attention: Q, K, V all come from the same source Z.

$$\begin{aligned} \mathrm{Q}_i &= \mathrm{Z}\mathrm{W}_Q^i \in \mathbb{R}^{(H\cdot W)\times d_k} && \text{(queries from latents)} \\ \mathrm{K}_i &= \mathrm{Z}\mathrm{W}_K^i \in \mathbb{R}^{(H\cdot W)\times d_k} && \text{(keys from latents)} \\ \mathrm{V}_i &= \mathrm{Z}\mathrm{W}_V^i \in \mathbb{R}^{(H\cdot W)\times d_v} && \text{(values from latents)} \end{aligned}$$

where $\mathrm{Z} \in \mathbb{R}^{(H\cdot W)\times d}$ are flattened UNet features.

Each head i:

$$\mathrm{head}_i = \mathrm{softmax}\left(\frac{\mathrm{Q}_i\mathrm{K}_i^T}{\sqrt{d_k}}\right)\mathrm{V}_i \in \mathbb{R}^{(H\cdot W)\times d_v}$$

Output: $\mathrm{Concat}(\mathrm{head}_1, \ldots, \mathrm{head}_h)\mathrm{W}^O \in \mathbb{R}^{(H\cdot W)\times d}$

Key: $\mathrm{Q}_i\mathrm{K}_i^T \in \mathbb{R}^{(H\cdot W)\times(H\cdot W)}$ — each pixel attends to all pixels!

Cross-Attention in UNet: Text Conditioning

Goal: Condition each spatial location on the text prompt.

Cross-Attention: Q from latents, K,V from text.

$$\begin{aligned} Q_i &= ZW_Q^i \in \mathbb{R}^{(H\cdot W)\times d_k} \quad \text{(queries from latents)} \\ K_i &= CW_K^i \in \mathbb{R}^{77\times d_k} \quad \text{(keys from text)} \\ V_i &= CW_V^i \in \mathbb{R}^{77\times d_v} \quad \text{(values from text)} \end{aligned}$$

where $Z \in \mathbb{R}^{(H\cdot W)\times d}$ (UNet), $C \in \mathbb{R}^{77\times 768}$ (CLIP text).

Each head i:

$$\text{head}_i = \text{softmax}\left(\frac{Q_i K_i^T}{\sqrt{d_k}}\right) V_i \in \mathbb{R}^{(H\cdot W)\times d_v}$$

Output: $\text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O \in \mathbb{R}^{(H\cdot W)\times d}$

Key: $Q_i K_i^T \in \mathbb{R}^{(H\cdot W)\times 77}$ — each pixel attends to text tokens!

Implementation Note: Efficient Multi-Head Attention

Conceptual: h independent projections $\mathrm{W}_Q^1, \ldots, \mathrm{W}_Q^h$.
Efficient implementation: Combine into single large matrix.

$$\mathrm{W}_Q = \begin{bmatrix} \mathrm{W}_Q^1 & \mathrm{W}_Q^2 & \cdots & \mathrm{W}_Q^h \end{bmatrix} \in \mathbb{R}^{d \times (h \cdot d_k)}$$

Steps:

1. **Single projection:** $\mathrm{Q} = \mathrm{Z}\mathrm{W}_Q \in \mathbb{R}^{n \times (h \cdot d_k)}$.
2. **Reshape:** Split last dimension into heads: $\mathbb{R}^{n \times h \times d_k}$.
3. **Transpose:** $\mathbb{R}^{h \times n \times d_k}$ for parallel computation.
4. **Attention:** Compute h attentions in parallel (batched).
5. **Concatenate:** Reshape back to $\mathbb{R}^{n \times (h \cdot d_k)}$.
6. **Output projection:** W^O.

Result: Mathematically equivalent, computationally efficient in GPUs!

Training Stable Diffusion

Training objective: Learn to predict noise in latent space.

Loss function:

$$\mathcal{L} = \mathbb{E}_{\mathrm{z}_0,\mathrm{c},\epsilon,t}\left[\|\epsilon - \epsilon_\theta(\mathrm{z}_t, t, \mathrm{c})\|^2\right]$$

where:

- $\mathrm{z}_0 = \phi(\mathrm{x}_0)$: Encoded latent from image.
- $\mathrm{c} = \mathrm{CLIP}(\mathrm{prompt})$: Text embedding.
- $t \sim \mathrm{Uniform}(1, T)$: Random timestep.
- $\epsilon \sim \mathcal{N}(0, I)$: Random noise.
- $\mathrm{z}_t = \sqrt{\bar{\alpha}_t}\mathrm{z}_0 + \sqrt{1 - \bar{\alpha}_t}\epsilon$: Noisy latent.

Frozen components:

- VAE encoder and decoder.
- CLIP text encoder.

Trainable component:

- UNet denoising network only.

Training Algorithm

Algorithm 1 Training Stable Diffusion

```
Require: Dataset of image-text pairs $\{(x^{(i)}, \text{prompt}^{(i)})\}$, VAE encoder $\phi$, CLIP text encoder, UNet $\epsilon_\theta$.
1: Freeze VAE and CLIP text encoders.
2: repeat
3:     Sample batch $\{(x^{(i)}, \text{prompt}^{(i)})\}_{i=1}^{B}$.
4:     // Encode to latent space.
5:     $z_0^{(i)} \leftarrow s \cdot \phi(x^{(i)})$ for all $i$ {VAE encoding}.
6:     // Encode text prompts.
7:     $c^{(i)} \leftarrow \text{CLIP}(\text{prompt}^{(i)})$ for all $i$ {Text encoding}.
8:     // Randomly drop conditioning (10% of samples).
9:     With probability 0.1: $c^{(i)} \leftarrow \emptyset$ {For CFG}.
10:    // Sample random timesteps and noise.
11:    Sample $t^{(i)} \sim \text{Uniform}(\{1, \dots, T\})$ for all $i$.
12:    Sample $\epsilon^{(i)} \sim \mathcal{N}(0, I)$ for all $i$.
13:    // Add noise to latents.
14:    $z_t^{(i)} \leftarrow \sqrt{\bar{\alpha}_{t^{(i)}}} z_0^{(i)} + \sqrt{1 - \bar{\alpha}_{t^{(i)}}} \epsilon^{(i)}$.
15:    // Predict noise.
16:    $\hat{\epsilon}^{(i)} \leftarrow \epsilon_\theta(z_t^{(i)}, t^{(i)}, c^{(i)})$ {UNet forward}.
17:    // Compute loss and update.
18:    $\mathcal{L} \leftarrow \frac{1}{B} \sum_{i=1}^{B} \|\epsilon^{(i)} - \hat{\epsilon}^{(i)}\|^2$.
19:    Update $\theta$ using $\nabla_\theta \mathcal{L}$.
20: until converged.
```

Sampling Algorithm (Inference)

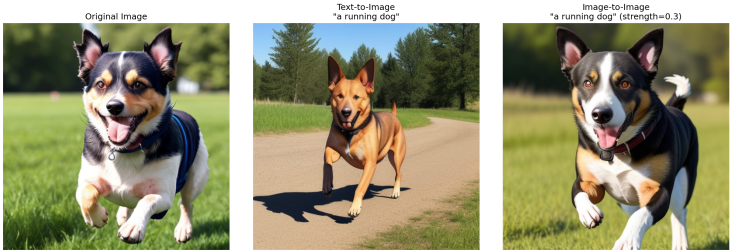
Algorithm 2 Sampling from Stable Diffusion

```
Require: Trained UNet $\epsilon_\theta$, VAE decoder $\psi$, CLIP text encoder, text prompt, guidance scale $w$
1: // Encode text prompt
2: c ← CLIP(prompt) {Conditional embedding}
3: $c_\emptyset$ ← CLIP(“”) {Unconditional (empty) embedding}
4: // Start from random noise
5: Sample $z_T \sim \mathcal{N}(0, I)$ {Pure noise in latent space}
6: // Denoising loop
7: for $t = T, T-1, \ldots, 1$ do
8:     // Predict noise with and without conditioning
9:     $\epsilon_{\text{uncond}} \leftarrow \epsilon_\theta(z_t, t, c_\emptyset)$
10:    $\epsilon_{\text{cond}} \leftarrow \epsilon_\theta(z_t, t, c)$
11:    // Apply classifier-free guidance
12:    $\hat{\epsilon} \leftarrow \epsilon_{\text{uncond}} + w \cdot (\epsilon_{\text{cond}} - \epsilon_{\text{uncond}})$
13:    // Denoise one step
14:    $z_{t-1} \leftarrow$ scheduler.step($\hat{\epsilon}, t, z_t$)
15: end for
16: // Decode to image
17: $x_0 \leftarrow \psi(z_0/s)$ {VAE decoding}
18: return $x_0$
```

Inference: Visual Demonstration

Text-to-Image Generation (see code1-stable-diffusion.py)

Prompt: *"a running dog"*



Left: Original image for reference.
Center: Text-to-image (from pure noise).
Right: Image-to-image (original + noise, then denoise with text guidance).

Image-to-Image Generation

Start from a noisy version of an existing image to preserve its structure rather than generating a new image from pure noise.

Process:

1. Encode source image: $\mathsf{z}_0 = \phi(\mathsf{x}_{\text{source}})$.
2. Add controlled noise:

$$\mathsf{z}_{t_0} = \sqrt{\bar{\alpha}_{t_0}}\mathsf{z}_0 + \sqrt{1-\bar{\alpha}_{t_0}}\epsilon$$

3. Denoise from t_0 to 0 (instead of T to 0).
4. Decode: $\mathsf{x}_{\text{output}} = \psi(\mathsf{z}_0)$.

Strength parameter $s \in [0, 1]$, $t_0 = s \cdot T$.

- $s = 0$: No change (skip denoising).
- $s = 0.3$: Subtle modifications, preserves image structure.
- $s = 0.7$: Major changes, follows prompt more.
- $s = 1.0$: Equivalent to text-to-image.

Applications of Stable Diffusion

1. **Text-to-Image:** Generate images from text descriptions (e.g., art generation).
2. **Image-to-Image:** Transform existing images (e.g., style transfer).
3. **Inpainting:** Fill masked regions (e.g., object removal, image completion).
4. **Super-resolution:** Upscale low-res images (e.g., detail enhancement, quality improvement).

Fine-tuning Stable Diffusion

Why fine-tune?

- Adapt to specific domains (e.g., medical images, art styles).
- Learn new concepts or objects.
- Improve quality on specific types of prompts.

Full Fine-tuning:

- Train entire UNet on custom dataset.
- Requires: Many images (1000+), high compute (GPU days).
- Best quality but expensive.

Efficient Alternative: LoRA (Low-Rank Adaptation).

- Freeze original weights, add small trainable layers.
- Requires: Few images (10-100), low compute (GPU hours).
- Good quality with minimal resources.

Let's understand LoRA...

LoRA: Low-Rank Adaptation

Instead of updating all weights $W \in \mathbb{R}^{d\times k}$, add a low-rank perturbation.

Original layer:

$$y = Wx$$

LoRA layer:

$$y = \underbrace{W}_{\text{frozen}} x + \underbrace{BA}_{\text{trainable}} x$$

where:

- $W \in \mathbb{R}^{d\times k}$: Original frozen weights.
- $B \in \mathbb{R}^{d\times r}$, $A \in \mathbb{R}^{r\times k}$: Trainable low-rank matrices.
- $r \ll \min(d, k)$: Rank (typically $r = 4, 8, 16$).

Parameter reduction:

$$\text{Original: } d \times k \quad \Rightarrow \quad \text{LoRA: } r(d + k)$$

Ex: $d = k = 1024$, $r = 8$: $1,048,576 \Rightarrow 16,384$ ($64\times$ reduction!).

LoRA: Low-Rank Adaptation

Intrinsic Dimensionality Hypothesis:
It works when the updates to weights during fine-tuning lies in a low-dimensional subspace.

Mathematical intuition:
Full update: $\mathrm{W}_{\mathrm{new}} = \mathrm{W} + \Delta\mathrm{W}$
LoRA approximation: $\Delta\mathrm{W} \approx \mathrm{BA}$ where $\mathrm{rank}(\mathrm{BA}) = r$

Benefits for Stable Diffusion fine-tuning:

1. **Memory efficient:** Only train $\sim$1% of parameters.
2. **Fast training:** Fewer parameters to update.
3. **Modular:** Can swap LoRA weights without changing base model.
4. **Composable:** Can combine multiple LoRAs.
5. **No catastrophic forgetting:** Base model stays intact.

Where to apply LoRA: Attention projection layers in UNet.

Fine-tuning: Training Setup

Full fine-tuning (see code2-finetune-stable-model.py):

Frozen components:

- VAE encoder/decoder.
- CLIP text encoder.

Trainable:

- Entire UNet (or with LoRA: only LoRA layers).

Training hyperparameters:

- Learning rate: 1×10^{-5} (lower than pretraining).
- Batch size: 1-4 (limited by GPU memory).
- Resolution: 384×384 or 512×512.
- Epochs: 10-100 (depends on dataset size).

Memory optimizations:

- Gradient checkpointing (trades compute for memory).
- 8-bit optimizer (50% memory reduction).
- Mixed precision training (FP16).

Fine-tuning Algorithm

Algorithm 3 Fine-tuning Stable Diffusion (Full or LoRA)

```
Require: Custom dataset $\{(x^{(i)}, \text{prompt}^{(i)})\}$, pretrained model
1: Load pretrained VAE, CLIP, UNet
2: Freeze VAE and CLIP
3: if using LoRA then
4:     Freeze UNet base weights
5:     Add LoRA layers to attention projections
6:     Initialize $A \sim \mathcal{N}(0, \sigma^2)$, $B = 0$
7: end if
8: repeat
9:     Sample batch $\{(x^{(i)}, \text{prompt}^{(i)})\}$
10:     Encode: $z_0^{(i)} \leftarrow \mathcal{E}(x^{(i)})$, $c^{(i)} \leftarrow \text{CLIP}(\text{prompt}^{(i)})$
11:     Sample $t^{(i)}$, $\epsilon^{(i)}$
12:     Compute $z_t^{(i)} = \sqrt{\bar{\alpha}_{t^{(i)}}} z_0^{(i)} + \sqrt{1 - \bar{\alpha}_{t^{(i)}}} \epsilon^{(i)}$
13:     Predict: $\hat{\epsilon}^{(i)} \leftarrow \epsilon_\theta(z_t^{(i)}, t^{(i)}, c^{(i)})$
14:     Loss: $\mathcal{L} = \frac{1}{B} \sum_i \|\epsilon^{(i)} - \hat{\epsilon}^{(i)}\|^2$
15:     if using LoRA then
16:         Update only LoRA weights (A, B)
17:     else
18:         Update all UNet weights
19:     end if
20: until converged
21: Save fine-tuned model (or LoRA weights)
```

Fine-tuned Model Inference

Inference with fine-tuned model.

Process:

1. Load pretrained VAE and CLIP (unchanged).
2. Load fine-tuned UNet (or base UNet + LoRA weights).
3. Generate images using standard sampling algorithm.

Example prompt: *"a jumping dog with sunglasses wearing a bear hat"*

(Image generated using code3-stable-model-inference.py with user prompt)

What We Learned

1. **Stable Diffusion = Latent Diffusion + Text Conditioning**
   - VAE compresses to 48$\times$ smaller latent space.
   - Diffusion operates in latent space (much faster).
2. **Three key components:**
   - VAE: Compression and reconstruction.
   - CLIP: Text understanding.
   - UNet: Denoising with cross-attention.
3. **Training:** Predict noise in latent space conditioned on text.
4. **Inference:** Iterative denoising with classifier-free guidance.
5. **Fine-tuning:** Adapt to custom domains.
   - Full: High quality, high cost.
   - LoRA: Good quality, low cost (recommended).

See also other techniques: DreamBooth, IP-Adapter, Textual Inversion, and ControlNet.

Take-home Message

- VAE (see code4-train-vae.py) requires several thousands of samples at least to reduce blurring and generate clear images in stable diffusion.
- Training a simple diffusion model from scratch also requires many samples (code5-train-diffusion-from-scratch.py), but stable diffusion can considerably amend the problem (code5-train-stable-diffusion-from-scratch.py).
- Real life problems may present a few images from domain difficult to adapt the model via fine-tuning.
- LoRA is an alternative (code6-train-model-with-lora.py and code7-generate-image-with-lora.py), but

  can it adapt a pretrained stable diffusion model to a distinct domain with a few images?