Diffusion Models — Practice

Lecture 22 · ES 667: Deep Learning

Prof. Nipun Batra
IIT Gandhinagar · Aug 2026

Learning outcomes

By the end of this lecture you will be able to:

1. Condition a diffusion model on **text** via cross-attention.
2. Derive the **classifier-free guidance** update rule and pick w.
3. Explain why **latent diffusion** made Stable Diffusion shippable.
4. Describe **DDIM** sampling and skip 95% of steps with no retraining.
5. Recognize **DiT** (diffusion Transformer) as U-Net's replacement.
6. Place **flow matching** and **consistency models** as the 2026 frontier.

Where we are

Last lecture · **DDPM** — forward noise, learn reverse, predict ε. Works on MNIST, toy 2D. But how do we get from there to **Stable Diffusion** and **Sora**?

> REFERENCE
>
> Today maps to **Prince Ch 18 (later sections)** + HF `diffusers` docs + Rombach 2022 (Stable Diffusion) + Ho & Salimans 2022 (CFG).

Four questions:

1. How do we **condition** on a text prompt?
2. What is **classifier-free guidance**?
3. Why does **latent diffusion** matter?
4. What about **faster** sampling — DDIM and DiT?

PART 1

Conditioning · from unconditional to text-to-image

How to condition a diffusion model

We want $\epsilon_\theta(x_t, t, c)$ — predict noise given a **condition** c (class label, text embedding, image).

Three common ways to inject c:

1. **Concatenate** c to the input channels (for class labels).
2. **Add** c to the time embedding (simple but loses detail).
3. **Cross-attention** between c and intermediate features (Stable Diffusion, most modern).

Cross-attention · the painter-following-instructions analogy

KEY IDEA

How does an image model **listen** to a text prompt?

Imagine a painter following instructions. For every brushstroke they ask, *"Which part of the instruction applies right here?"*

Cross-attention is exactly this · each spatial location in the image listens to the most relevant words in the prompt.

The result · a dynamic, **spatial** link between text and pixels. The "cat" word strongly influences cat-shaped regions; the "background" word influences corners.

Cross-attention · how an image listens to text

INTUITION

Analogy · team of painters. Each painter handles a small patch. Before painting they ask: *"which word in the prompt is most relevant to **my patch**?"* The painter near the top pays attention to "hat"; the furry-patch painter pays attention to "cat".

The mechanism — for each image region:

1. **Get word vectors** from a frozen text encoder (CLIP) → keys K and values V.
2. **Get region "questions"** from the U-Net's intermediate features → queries Q.
3. **Score relevance** · $\mathrm{sim}(Q, K)$, then softmax.
4. **Final instruction** · weighted sum of V → used to denoise that region.

```
text_emb = clip_text_encoder("a cat astronaut")  # [1, 77, 768]

class CrossAttention(nn.Module):
    def forward(self, spatial_features, text_emb):
        Q = self.q_proj(spatial_features)
        K = self.k_proj(text_emb)
        V = self.v_proj(text_emb)
```

Cross-attention · worked numeric

Prompt: "a red square". Two word vectors: $V_{\text{red}} = [1, 0]$, $V_{\text{square}} = [0, 1]$.

Top-left pixel's current state: $Q_{\text{TL}} = [0.1, 0.9]$ (more square-like than red-like).

Step 1 · scores.

- $\text{sim}(Q, V_{\text{red}}) = 0.1 \cdot 1 + 0.9 \cdot 0 = 0.1$
- $\text{sim}(Q, V_{\text{square}}) = 0.1 \cdot 0 + 0.9 \cdot 1 = 0.9$

"Square" is much more relevant.

Step 2 · softmax → weights. Suppose $w_{\text{red}} = 0.3$, $w_{\text{square}} = 0.7$.

Step 3 · final instruction.
$0.3 \cdot [1, 0] + 0.7 \cdot [0, 1] = [\mathbf{0.3, 0.7}]$

Used to update this pixel — **slightly more red, much more square**.

Cross-attention · why it works for text conditioning

Self-attention inside the U-Net mixes spatial positions. **Cross**-attention *between spatial features and text features* is where the prompt enters.

KEY IDEA

At a high-resolution block · each pixel asks "which prompt tokens matter for me?". A cat pixel pays attention to "cat" in the prompt; a sky pixel to "sky". The result is **spatially localized conditioning** — a single prompt can steer different regions differently.

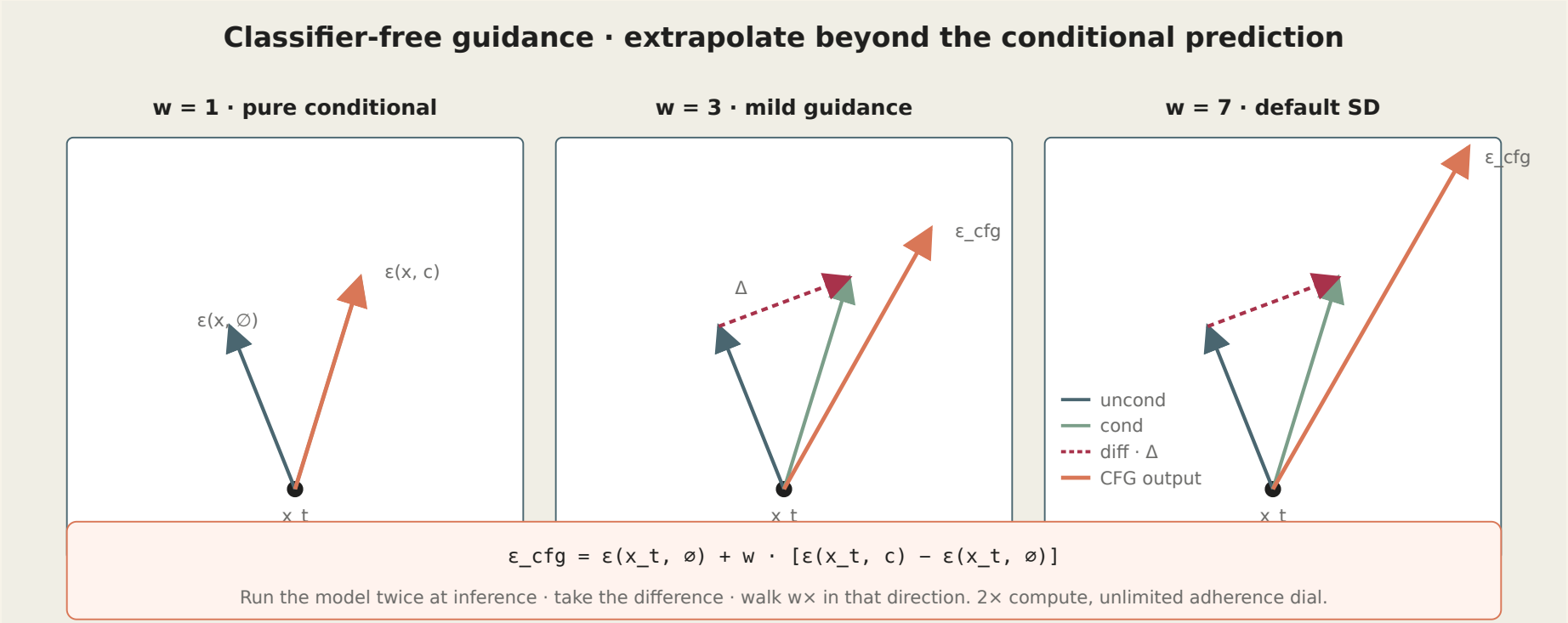
Visualizing the cross-attention maps reveals exactly this — each word has a blob of pixels that listened most to it. This is the mechanism behind DreamBooth, Prompt-to-Prompt, and every prompt-editing technique.

PART 2

Classifier-Free Guidance

The trick that makes generation feel "on-prompt"

CFG · the geometry



CFG · the two-compass analogy

INTUITION

You're lost in a forest with two compasses.

1. **Generic compass** · always points to "average image" (unconditional prediction $\epsilon_\emptyset$).
2. **Special compass** · points toward "cabin in the woods" (prompt-conditional ϵ_c).

The **difference vector** $\epsilon_c - \epsilon_\emptyset$ is the **pure influence of the prompt**. CFG · start at the generic point and walk w times that vector $\rightarrow$ over-shoot the conditional prediction in the prompt direction.

CFG · derive the formula step by step

1. **Generic direction.** $\epsilon_{\text{uncond}} = \epsilon_\theta(x_t, \emptyset)$ (null prompt).
2. **Prompt direction.** $\epsilon_{\text{cond}} = \epsilon_\theta(x_t, c)$.
3. **Pure prompt influence.**
   $\Delta = \epsilon_{\text{cond}} - \epsilon_{\text{uncond}}$
4. **Extrapolate** by guidance scale w:

$$\epsilon_{\text{CFG}} = \epsilon_{\text{uncond}} + w \cdot \Delta = \epsilon_\theta(x_t, \emptyset) + w\big(\epsilon_\theta(x_t, c) - \epsilon_\theta(x_t, \emptyset)\big)$$

- $w = 0$ → pure unconditional (ignore prompt).
- $w = 1$ → exactly the conditional prediction.
- $w > 1$ → **amplify** prompt adherence by overshooting.
  Default $w = 7$ in Stable Diffusion.

CFG in pictures

IN PRACTICE

▶️

Interactive: slide w from 0 to 30, watch prompt adherence vs artifacts — cfg-scale-visualizer.

Training with CFG-ready dropout

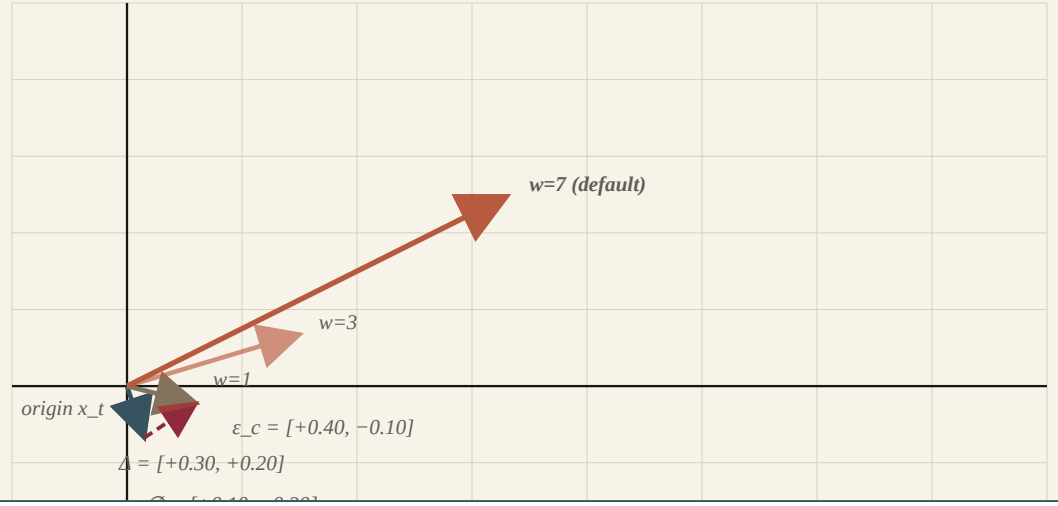
To enable CFG at inference, the training needs both conditional and unconditional examples:

```
def training_step(x0, prompt):
    t = sample_t()
    noise = torch.randn_like(x0)
    x_t = add_noise(x0, noise, t)

    # Dropout 10% of prompts to null (enables unconditional path later)
    if random.random() < 0.10:
        c = null_embedding
    else:
        c = clip_text_encoder(prompt)

    pred_noise = model(x_t, t, c)
    return F.mse_loss(pred_noise, noise)
```

Same network learns both modes. At inference, you run it twice and extrapolate. Cost · 2× compute per step. Benefit · any w at generation time.

CFG · vectors in 2D

CFG · same prompt, three guidance strengths · vectors in 2D



ε_cfg = ε_∅ + w · Δ · take the difference vector, walk w× along it.

Higher w · longer overshoot · stronger prompt adherence · saturated colors / artifacts at w > 12.

Worked example · CFG arithmetic at one step

DERIVATION

At denoising step t, the U-Net runs twice on the noisy latent x_t:

- $\epsilon_\emptyset = \epsilon_\theta(x_t, \text{null prompt}) = [+0.10, -0.30]$ (toy 2D)
- $\epsilon_c = \epsilon_\theta(x_t, \text{"cat"}) = [+0.40, -0.10]$

Difference vector · $\Delta = \epsilon_c - \epsilon_\emptyset = [+0.30, +0.20]$.

w	$\epsilon_{\text{cfg}} = \epsilon_\emptyset + w\Delta$	MEANING
0	$[+0.10, -0.30]$	unconditional
1	$[+0.40, -0.10]$	pure conditional
3	$[+1.00, +0.30]$	overshoot 2×
7	$[+2.20, +1.10]$	strong overshoot

Larger w pushes the noise prediction further along the "more cat-like" direction. Subtract that bigger noise → step lands further toward a strongly cat-shaped image.

Picking a CFG scale · practical guide

w	WHAT YOU GET
1	pure conditional · ignores extrapolation
3	subtle prompt adherence · natural-looking
7	Stable Diffusion default · balanced
12+	strong adherence · saturated colors, artifacts
25+	cartoonish oversaturation; often broken

INTUITION

CFG trades **diversity for prompt adherence**. Low w produces many different interpretations; high w produces a narrower, more on-prompt distribution. Treat w as a hyperparameter you sweep for each task.

PART 3

Latent diffusion

The one trick that made Stable Diffusion ship

Why diffuse in latent space · the intuition

Most pixels in an image are *correlated*. A patch of blue sky has hundreds of near-identical pixel values. Diffusing each one independently is wasteful.

KEY IDEA

A pretrained VAE has already absorbed the **perceptually redundant** structure · the latent is a near-minimal representation. Diffusion then only has to model the *interesting* (high-entropy) dimensions.

Result · 48× fewer dimensions, same perceptual quality, ~10× faster sampling. This is the single change that made Stable Diffusion runnable on consumer GPUs.

Latent diffusion · the zip-the-file analogy

INTUITION

A 512×512 photo of blue sky has 262,144 pixels — most are nearly identical shades of blue. Asking a network to predict "this blue pixel is followed by a blue pixel" 1000× is wasteful.

Like emailing a 100MB doc. Instead of attaching it raw, you zip to 1MB. The zip captures the important info in less space. Latent diffusion does the same · zip the image with a VAE, run diffusion in the small zipped space, then unzip at the end.

The dimension math

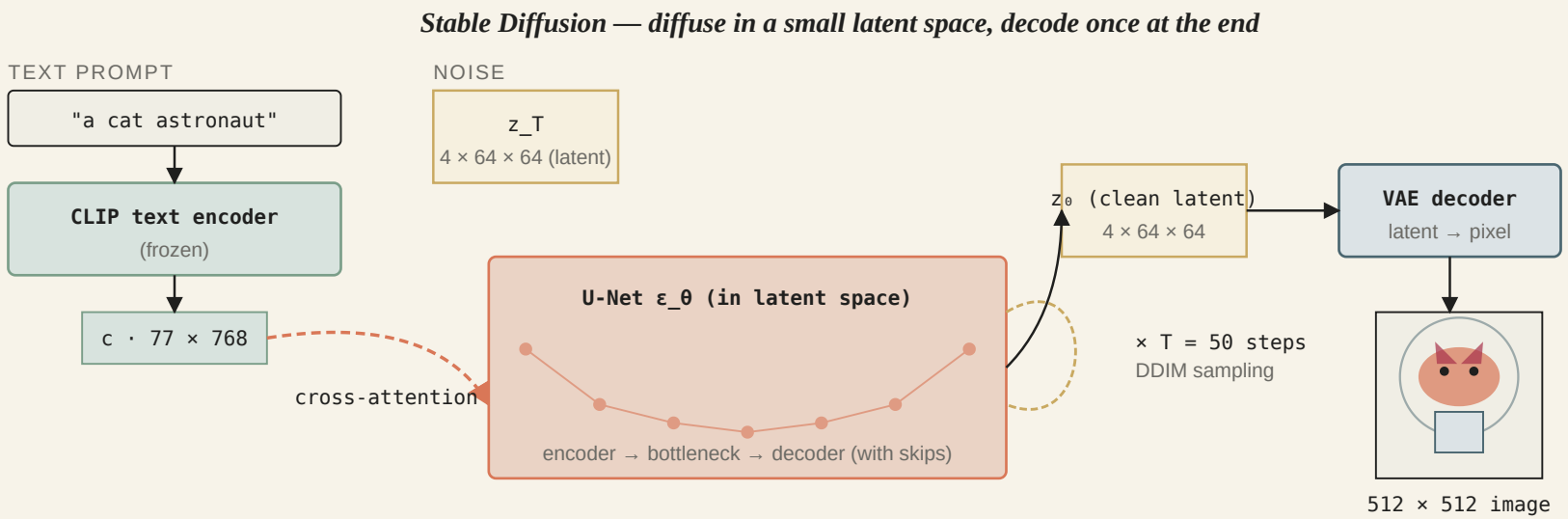
512×512 RGB = $512 \cdot 512 \cdot 3 = \mathbf{786{,}432}$ dimensions.

Stable Diffusion's VAE compresses to 64×64×4 = $64 \cdot 64 \cdot 4 = \mathbf{16{,}384}$ dimensions.

Saving · $786{,}432 / 16{,}384 = \mathbf{48}\times$ **fewer dimensions** for the expensive U-Net to process.

The diffusion loop runs in latent space; the VAE encoder runs once at the start, decoder once at the end. Single biggest reason Stable Diffusion is shippable on consumer GPUs.

Stable Diffusion architecture



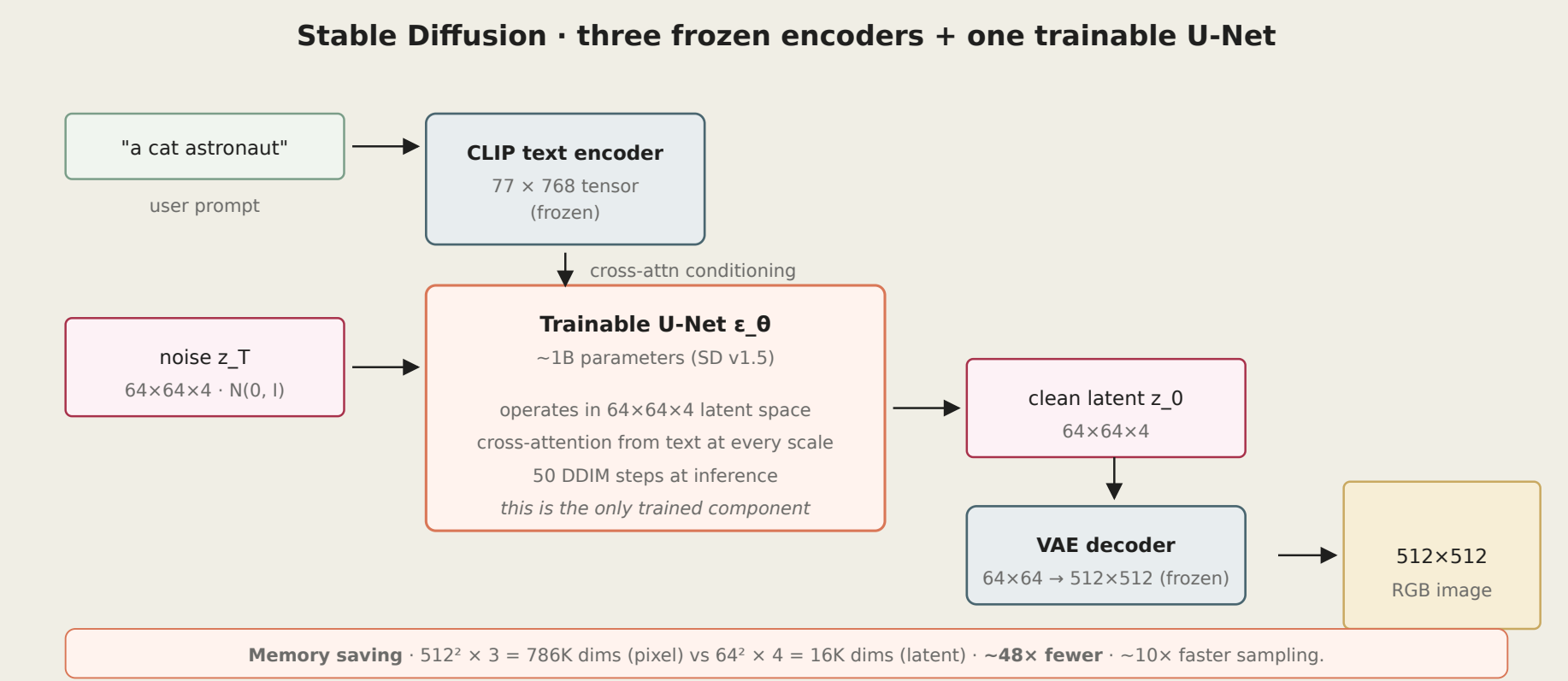
WHY DIFFUSE IN LATENT SPACE?

VAE compresses 512×512×3 (786k dims) → 4×64×64 (16k dims) · **48× smaller.**

Running 50 U-Net forwards at 64² resolution is feasible on consumer GPUs. Pixel-space diffusion would be 48× more expensive.

Rombach et al. 2022 — the one trick that made Stable Diffusion ship on laptops.

Stable Diffusion · annotated full stack



Latent diffusion · three components

1. VAE (frozen)

Encoder: image → 4-channel latent.

Decoder: latent → image.

Pretrained, not updated during diffusion training.

2. CLIP text encoder (frozen)

Tokenize → embed → 77 × 768 tensor.

Frozen — the diffusion model just consumes these.

3. U-Net diffuser (trainable)

Operates in the 64×64×4 latent space.

Cross-attention injects text embeddings at every scale.

~1B parameters for SD v1.5.

This is the only thing you train.

PART 4

Faster sampling

DDIM and DiT

Why 1000 steps · the quick math

DDPM's forward process adds tiny Gaussian noise at each of T steps. To keep the final distribution close to $\mathcal{N}(0, I)$ and each step's noise increment small (so the reverse can be Gaussian too), T has to be large — 1000 is the typical choice.

KEY IDEA

Small per-step noise → easier inverse problem for the network. But then the reverse loop has **1000 forward passes** — painful at inference.

DDIM (next slide) breaks this by reinterpreting the reverse process as *non-Markovian*, so you can skip steps without retraining.

DDPM is slow · 1000 steps

Vanilla DDPM sampling requires running the U-Net **1000 times per generation**. Each step is a forward pass of a ~1B param network.

We want to make it faster without retraining. Two approaches:

- **Fewer sampling steps** (DDIM, DPM-Solver)
- **Better architecture** (DiT, replacing U-Net with Transformer)

DDIM · the foggy-hill analogy

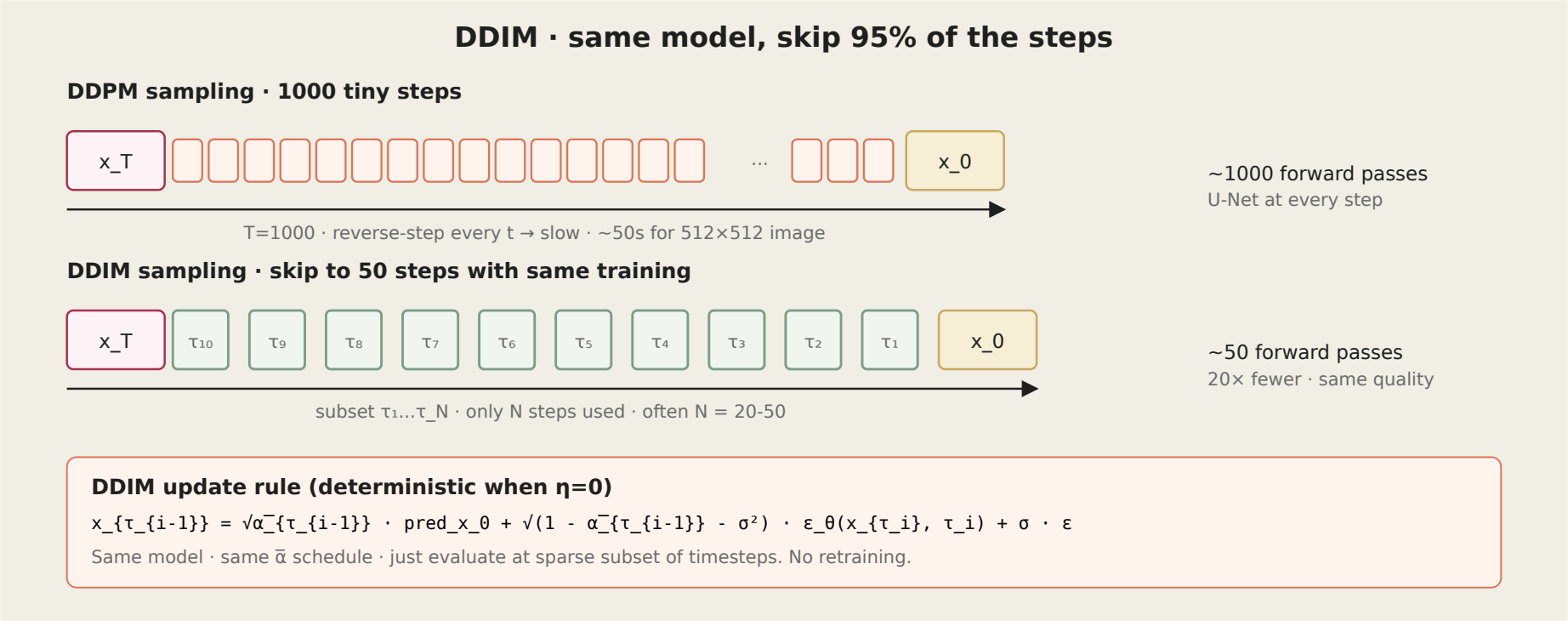
KEY IDEA

Original DDPM is like walking down a rocky hill in **fog**. You take 1000 tiny careful steps because you can only see one step ahead.

DDIM realizes · with the right model, **you can see the whole path**. Instead of 1000 wobbly steps, take 50 confident strides directly toward the final image.

The same trained model gets used · DDIM just chooses which subset of timesteps to evaluate. No retraining. 20× speedup at no cost in sample quality.

DDIM · skip 95% of steps



DDIM · deterministic sampling in 20–50 steps

REFERENCE

Song, Meng, Ermon 2020 · *"Denoising Diffusion Implicit Models"*

DDIM reformulates the reverse process to be **deterministic** given the initial noise. Crucially, the trained DDPM model can be sampled with DDIM — no retraining.

Effect · 50 DDIM steps ≈ 1000 DDPM steps in quality. **20× speedup** essentially for free.

In 2026, DDIM (and its successor DPM-Solver++) is the default sampler in every diffusion library.

DiT · the global town-hall analogy

INTUITION

Convolution (U-Net): you only talk to your neighbours. Messages travel via "telephone" through many layers to reach a far-away pixel.

Attention (Transformer): every pixel is in a global video call. The cat-ear pixel can directly ask the cat-tail pixel "are you also part of the cat?" Better long-range understanding.

DiT · how it works

Peebles & Xie 2023 · replace the U-Net with a Transformer.

1. **Patchify** the (small, latent) noisy image into a grid of patches (like ViT).
2. **Treat patches as tokens** · flatten each → vector. Image becomes a *sequence*.
3. **Transformer blocks** · self-attention lets every patch look at every other.
4. **Re-assemble** → noise prediction.

Why · Transformers scale incredibly well. More data + more compute = better, indefinitely. Inheriting LLM-ecosystem optimizations (FlashAttention, tensor parallelism). Backbone of **Sora**, **Stable Diffusion 3**, and most 2024+ frontier image/video models.

PART 5

The generative landscape · 2026

What ships in 2026

SYSTEM	ARCHITECTURE	NOTES
Stable Diffusion 3	DiT + rectified flow	open-weight, commercial
DALL-E 3	diffusion (details undisclosed)	integrated with ChatGPT
Midjourney v6+	custom diffusion	paid, high fidelity
Sora	DiT on spacetime patches	OpenAI video model
Veo 2	latent diffusion, video	Google
Flux	open-weight SDXL successor	commercial, fast

All diffusion-based. All descendants of the 2020 DDPM paper.

Diffusion for non-image modalities

- **Text** · text-diffusion (still exploratory; LLMs beat it for quality).
- **Audio** · AudioGen, Riffusion, MusicLM.
- **3D** · DreamFusion, Gaussian Splatting integrates with diffusion priors.
- **Molecules** · RFdiffusion designs proteins (Baker lab, won 2024 Nobel adjacent).
- **Robotics** · Diffusion Policy (Chi 2023) — action generation via diffusion.

Diffusion is one of the two big generative paradigms now (the other: autoregressive LLMs).

2026 · the sampler menu

DERIVATION

SAMPLER	STEPS	QUALITY	NOTES
DDPM (original)	1000	baseline	slowest, stochastic
DDIM (deterministic)	50-100	=	same model; drop-in
DPM-Solver++	20-30	=	ODE solver · default in HF diffusers
Flow matching ODE	8-20	=	straighter trajectories
Consistency models	**1-4**	slight drop	distilled; near-real-time

INTUITION

5-year trajectory · 1000 steps (2020) → 50 steps (DDIM) → 4 steps (consistency) → 1 step (Rectified Flow v3 distilled). Each generation reduced inference cost by ~10×.

DiT · Transformer backbone

Peebles & Xie 2022 · replace U-Net entirely with a Transformer.

DERIVATION

- **Patchify** the noisy latent (like ViT).
- **Transformer blocks** with attention + FFN.
- **Time + class conditioning** via adaLN (adaptive LayerNorm).
- Scales cleanly · more params → better (U-Net plateaus).

IN PRACTICE

Stable Diffusion 3 and Sora both use DiT-based architectures. DiT inherits Transformer's scaling laws · more data + compute = better samples, indefinitely.

Diffusion · by modality in 2026

DERIVATION

MODALITY	MODEL	PIXELS / SEC TO GENERATE
Images 1024^2	SD3 · Flux	~0.5 images/s (20 steps)
Video 720p · 16s	Sora · VEO	minutes per clip
Audio music	AudioGen · MusicLM	real-time (8 DDIM steps)
3D meshes	Diffusion-SDF · ShapE	~30s per mesh
Molecules	RFdiffusion	10s per protein structure
Robot policies	Diffusion Policy	50 Hz control loop

INTUITION

The diffusion paradigm scaled to every signal that can be noised. 2026 is the golden age · expect more modalities (brain signals, weather, materials) by 2028.

Consistency models · winding road vs teleporter

INTUITION

Normal DDIM/DDPM · walk down a winding mountain path from foggy peak (noise) to clear valley (image). 50+ steps along the path.

Consistency model = a teleporter. Stand at any point on the path, press a button, instantly arrive at the valley. The teleporter is *consistent* — no matter where on the path you stand, it always takes you to the same final x_0.

Consistency models · the property

Train $f_\theta(x_t, t) \to x_0$ to satisfy:

$$f_\theta(x_{t_1}, t_1) = f_\theta(x_{t_2}, t_2) \approx x_0$$

for any two points on the same trajectory.

Inference:

1. Sample $x_T \sim \mathcal{N}(0, I)$.
2. Run network **once**: $\hat{x}_0 = f_\theta(x_T, T)$.
3. Done.

   SDXL-Turbo (2023) · 1-step generation at near-50-step DDIM quality. LCM adapters · drop-in for Stable Diffusion. **GAN-speed with diffusion-quality, finally.**

Flow matching · the next paradigm?

Rectified flow (Liu 2022) and **flow matching** (Lipman 2022) generalize diffusion:

- Train a network to predict a velocity field from noise to data.
- Sample with ODE solver in **4-10 steps** (vs diffusion's 50+).
- Cleaner math; often better quality at fewer steps.

  Stable Diffusion 3, Flux, and many 2024+ models use flow-matching instead of pure diffusion. The boundary is blurring — both are "continuous-time generative models."

Summary · Lecture 22 — summary

- **Text conditioning** · CLIP text encoder + cross-attention in every U-Net block.
- **CFG** · $\epsilon_{\text{CFG}} = \epsilon(x, \emptyset) + w(\epsilon(x, c) - \epsilon(x, \emptyset))$. Default $w = 7$.
- **Latent diffusion** · VAE compresses 48×; U-Net diffuses in latent. Made SD feasible.
- **DDIM** · deterministic sampling, 20×+ fewer steps, no retraining.
- **DiT** · Transformer backbone replacing U-Net; powers Sora, SD3.
- **Flow matching** · successor to pure diffusion; fewer sampling steps, cleaner math.

Read before Lecture 23

Chip Huyen blog posts on efficient inference; Dao 2022 (FlashAttention).

Next lecture

Efficient Inference — KV-cache, quantization, distillation, FlashAttention, speculative decoding.

NOTEBOOK

Notebook 22 · `22-stable-diffusion.ipynb` — use HF `diffusers`; implement CFG loop manually from a pretrained model; sweep guidance scale; compare samplers.