

# Diffusion Models: Theory and Applications

## Lecture 6: Conditional Generation - From Random to Controllable


Shubham Chatterjee

Missouri University of Science and Technology, Department of Computer Science

June 11, 2025

We learned to turn noise into data...

But what if we want **CONTROL** over what we generate? 

- ✓ **DDPM sampling:** Stochastic reverse process with 1000 steps
- ✓ **DDIM sampling:** Deterministic acceleration with 20-50 steps
- ✓ **Trade-offs:** Quality vs. speed vs. diversity
-  **Today:** Add steering wheels to our diffusion models!

# Today's Mission: Controllable Generation

 How do we tell diffusion models **WHAT** to generate?

## The Challenge:

- Random generation is not enough
- Need to guide the sampling process
- Want specific classes, styles, or attributes
- Must maintain high quality

## Three Main Approaches:

- **Class-conditional:** Simple, direct conditioning
- **Classifier guidance:** External steering during sampling
- **Classifier-free guidance:** The modern breakthrough

**From chaos to creation - with intention!**

# The Fundamental Problem: Unconditional Limitations ?

## Why unconditional generation isn't enough...


### The randomness dilemma

- You get random samples from the data distribution
- No control over what specific content is generated
- Need to generate many samples to find what you want
- Wastes computational resources and time

# The Fundamental Problem: Unconditional Limitations ?

## What we really want

- **Class control:** “Generate a dog” vs. “Generate a cat”
- **Style control:** “In the style of Van Gogh”
- **Text control:** “A sunset over mountains”
- **Spatial control:** “Put the object here”

**Solution:** Replace  $p(\mathbf{x})$  with  $p(\mathbf{x}|\mathbf{y})$  where  $\mathbf{y}$  is our condition! 

Remember how we parameterized the reverse process?

The learned reverse transition

$$p_{\theta}(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_{\theta}(\mathbf{x}_t, t), \tilde{\sigma}_t^2 \mathbf{I})$$

Mean parameterization (from noise prediction)

$$\mu_{\theta}(\mathbf{x}_t, t) = \frac{1}{\sqrt{\alpha_t}} \left( \mathbf{x}_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \epsilon_{\theta}(\mathbf{x}_t, t) \right)$$


Fixed variance

$$\tilde{\sigma}_t^2 = \frac{1 - \bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t} \beta_t$$

**Key insight:** We learn the mean, variance comes from the noise schedule! 

# The Neural Network Behind Diffusion: U-Net

Before diving into conditioning, let's understand our core architecture...

What exactly is  $\epsilon_{\theta}(\mathbf{x}_t, t)$ ? 

- **Input:** Noisy image  $\mathbf{x}_t$  + timestep  $t$
- **Output:** Predicted noise  $\hat{\epsilon}$
- **Architecture:** U-Net - the workhorse of diffusion models

Why U-Net? 

- Originally designed for image segmentation
- Perfect for image-to-image tasks (noise  $\rightarrow$  same size noise)
- Preserves spatial information while processing
- Natural fit for denoising operations

**Understanding U-Net is key to understanding conditioning!** 

# U-Net: The Backbone of Diffusion Models

Understanding the architecture that makes conditional generation possible

## Why U-Net for diffusion?

- **Originally designed:** Biomedical image segmentation (2015)
- **Perfect fit:** Image-to-image transformation with spatial preservation
- **Key insight:** Denoising is just another image-to-image task!
- **Conditioning ready:** Architecture naturally supports additional inputs

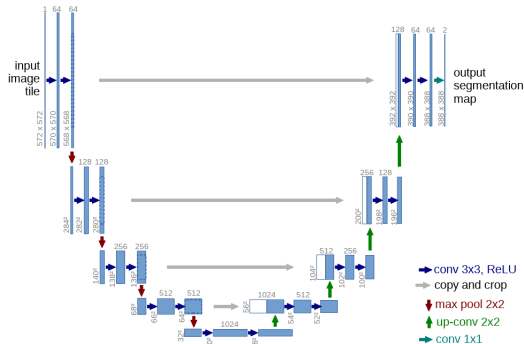
## The segmentation heritage

- U-Net solved: *“Given a noisy microscopy image, output a clean segmentation”*
- Diffusion adapted: *“Given a noisy image + timestep, output predicted noise”*
- **Same spatial reasoning, different task!**

**From 30 biomedical images to billion-parameter generative models!** 



# U-Net Architecture: The “U” Shape Design



**Fig. 1.** U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.

Image taken from original paper: <https://arxiv.org/abs/1505.04597v1>

# U-Net Architecture: The “U” Shape Design

## ↓ Contracting Path (Encoder):

- **Purpose:** Build semantic understanding
- **Operations:** Conv + Pool (downsample)
- **Effect:** ↓ spatial size, ↑ channels
- **Result:** Rich contextual features

## Spatial resolution:

- $512 \times 512 \rightarrow 256 \times 256 \rightarrow 128 \times 128 \rightarrow 64 \times 64 \rightarrow 32 \times 32$

## ↑ Expanding Path (Decoder):

- **Purpose:** Recover spatial precision
- **Operations:** Upsample + Conv
- **Effect:** ↑ spatial size, ↓ channels
- **Result:** Precise spatial output

## Channel progression:

- $64 \rightarrow 128 \rightarrow 256 \rightarrow 512 \rightarrow 1024$

The magic: Skip connections 

High-resolution features from encoder are concatenated with decoder features → **Best of both worlds!**

# Skip Connections: The Key Innovation

## The fundamental problem

- **Downsampling:** Necessary for semantic understanding but loses spatial details
- **Upsampling:** Can't perfectly recover lost information
- **Trade-off:** Context vs. localization - can't have both?

## Skip connections solve this!

- **Preserve details:** High-res features bypass the bottleneck
- **Combine information:** Concatenate rather than choose
- **Multi-scale features:** Each decoder level gets appropriate details
- **Gradient flow:** Better training with direct paths to early layers

## Why this matters for diffusion

- **Noise prediction:** Needs both global context AND local spatial precision
- **Fine details:** Skip connections preserve edge and texture information
- **Conditioning:** Multiple injection points for different scales of control

## How diffusion models modified the original U-Net design...

### Time embedding injection

- **Challenge:** Network needs to know the noise level
- **Solution:** Sinusoidal position encoding for timestep  $t$
- **Injection:** Add time embeddings to features after each ResNet block
- **Effect:** Network learns noise-level-specific denoising

## How diffusion models modified the original U-Net design...

### Attention mechanisms

- **Self-attention:** Added at multiple resolutions for global context
- **Cross-attention:** For conditioning on text, classes, etc.
- **Placement:** Typically at  $16 \times 16$  and  $8 \times 8$  spatial resolutions
- **Benefit:** Long-range dependencies and flexible conditioning

# U-Net: Perfect for Conditioning +

## Multiple injection points ↓

- **Embedding level:** Time + class embeddings combined
- **Feature level:** Cross-attention with text/image features
- **Skip connections:** Additional conditioning paths
- **Multi-scale:** Different conditions at different resolutions

## Hierarchical conditioning

- **Global:** Text/class conditioning affects overall generation
- **Regional:** Spatial conditioning controls local areas
- **Fine-scale:** Detail conditioning affects textures and edges
- **Flexible:** Can combine multiple conditioning types seamlessly

**U-Net's multi-scale nature makes it the perfect canvas for complex conditioning!** 

# From Unconditional to Conditional U-Net

## Unconditional U-Net

- **Input:** Noisy image  $\mathbf{x}_t$  + timestep  $t$
- **Output:** Predicted noise  $\epsilon_{\theta}(\mathbf{x}_t, t)$
- **Training:** MSE on unconditional noise prediction

## Conditional U-Net

- **Input:** Noisy image  $\mathbf{x}_t$  + timestep  $t$  + condition  $\mathbf{y}$
- **Output:** Predicted noise  $\epsilon_{\theta}(\mathbf{x}_t, \mathbf{y}, t)$
- **Training:** MSE on conditional noise prediction

## Key question: How to inject $\mathbf{y}$ ?

- Where in the network should conditioning information go?
- How to combine image features with condition features?
- How to ensure the network learns to use the conditioning?

## The simplest form of conditioning: class labels

### The basic idea

- Train on dataset with class labels:  $\{(\mathbf{x}^{(i)}, y^{(i)})\}$
- Learn  $p(\mathbf{x}|y)$  instead of  $p(\mathbf{x})$
- Modify U-Net to take class information as input
- Generate specific classes during sampling

### Mathematical formulation

Original objective:  $\mathcal{L} = \mathbb{E}_{t, \mathbf{x}_0, \epsilon} [\|\epsilon - \epsilon_\theta(\mathbf{x}_t, t)\|^2]$

Conditional objective:  $\mathcal{L} = \mathbb{E}_{t, \mathbf{x}_0, \mathbf{y}, \epsilon} [\|\epsilon - \epsilon_\theta(\mathbf{x}_t, \mathbf{y}, t)\|^2]$

**Same training procedure, just add the class label! +**



## How to inject class information into U-Net...

### Step 1: Class embedding

- Convert class label  $y$  to dense embedding:  $\mathbf{e}_y = \text{Embedding}(y)$
- Learnable embedding table (e.g., 1000 classes  $\rightarrow$  512-dim vectors)
- Same dimension as time embeddings for easy combination

## How to inject class information into U-Net...

### Step 2: Embedding fusion

- Combine with time embedding:  $\mathbf{e} = \mathbf{e}_t + \mathbf{e}_y$
- Alternative: concatenation  $\mathbf{e} = [\mathbf{e}_t; \mathbf{e}_y]$
- Pass through MLP for projection if needed

## How to inject class information into U-Net...

### Step 3: Feature injection ↓

- Inject combined embedding into each U-Net block
- Add to feature maps after normalization layers
- Scale and shift operations (similar to batch norm)

# Class-Conditional Training Algorithm

**Input:** Dataset  $\{(\mathbf{x}^{(i)}, y^{(i)})\}$ , model  $\epsilon_\theta$   
**while** not converged **do**  
    Sample batch  $\{(\mathbf{x}, \mathbf{y})\}$  from dataset  
    Sample timesteps  $t \sim \text{Uniform}(1, T)$  for each sample  
    Sample noise  $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$  for each sample  
    Compute noisy samples:  $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x} + \sqrt{1 - \bar{\alpha}_t}\epsilon$   
    Embed class:  $\mathbf{e}_y = \text{ClassEmbedding}(\mathbf{y})$   
    Embed timestep:  $\mathbf{e}_t = \text{TimeEmbedding}(t)$   
    Combine embeddings:  $\mathbf{e} = \mathbf{e}_t + \mathbf{e}_y$   
    Predict noise:  $\hat{\epsilon} = \epsilon_\theta(\mathbf{x}_t, \mathbf{e}, t)$   
    Compute loss:  $\mathcal{L} = \|\epsilon - \hat{\epsilon}\|^2$   
    Update parameters:  $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}$   
**end while**

Simple and effective for basic conditional generation! ✓

## Generating specific classes at inference time...

### DDIM sampling with class conditioning

**Input:** Trained model  $\epsilon_\theta$ , desired class  $y^*$ , timesteps  $\{\tau_1, \dots, \tau_S\}$

Sample  $\mathbf{x}_{\tau_S} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

Embed class:  $\mathbf{e}_y = \text{ClassEmbedding}(y^*)$

**for**  $i = S, S-1, \dots, 1$  **do**

$t = \tau_i, s = \tau_{i-1}$

    Embed timestep:  $\mathbf{e}_t = \text{TimeEmbedding}(t)$

    Combine:  $\mathbf{e} = \mathbf{e}_t + \mathbf{e}_y$

    Predict noise:  $\hat{\epsilon} = \epsilon_\theta(\mathbf{x}_t, \mathbf{e}, t)$

    Predict clean:  $\hat{\mathbf{x}}_0 = \frac{\mathbf{x}_t - \sqrt{1 - \bar{\alpha}_t} \hat{\epsilon}}{\sqrt{\bar{\alpha}_t}}$

    Update:  $\mathbf{x}_s = \sqrt{\bar{\alpha}_s} \hat{\mathbf{x}}_0 + \sqrt{1 - \bar{\alpha}_s} \hat{\epsilon}$

**end for**

**Return:**  $\mathbf{x}_0$

**Same DDIM sampling, just with class embedding! =**

# Class-Conditional: Pros and Cons

## ✓ Advantages:

- **Simple:** Easy to implement and understand
- **Fast:** No computational overhead during sampling
- **Reliable:** Works consistently across datasets
- **Memory efficient:** Small additional parameters

## ✗ Limitations:

- **Fixed:** Only works with predefined classes
- **Limited:** Can't combine or modify conditions
- **Retraining:** Need new model for new classes
- **Basic:** No fine-grained control within classes

Great for simple applications, but what about more complex control?

Enter classifier guidance... →

What if we could steer **ANY** pre-trained diffusion model?

### The brilliant insight

- Take an existing unconditional model (already trained!)
- Use a separate classifier to guide the sampling process
- No need to retrain the diffusion model
- Can add multiple types of guidance

### The mathematical foundation

From Bayes' rule:  $p(\mathbf{x}_t|\mathbf{y}) \propto p(\mathbf{y}|\mathbf{x}_t)p(\mathbf{x}_t)$

Taking gradients:  $\nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t|\mathbf{y}) = \nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t) + \nabla_{\mathbf{x}_t} \log p(\mathbf{y}|\mathbf{x}_t)$

**Conditional score = Unconditional score + Classifier gradient! +**

## How to modify noise predictions using classifier gradients...

### The score connection 🔗

Since  $\nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t) = -\frac{\epsilon_{\theta}(\mathbf{x}_t, t)}{\sqrt{1-\bar{\alpha}_t}}$ , we get:

$$\tilde{\epsilon}(\mathbf{x}_t, \mathbf{y}, t) = \epsilon_{\theta}(\mathbf{x}_t, t) - \omega \sqrt{1 - \bar{\alpha}_t} \nabla_{\mathbf{x}_t} \log p_{\phi}(\mathbf{y}|\mathbf{x}_t)$$

### What each term means 🎯

- $\epsilon_{\theta}(\mathbf{x}_t, t)$ : Original unconditional noise prediction
- $\nabla_{\mathbf{x}_t} \log p_{\phi}(\mathbf{y}|\mathbf{x}_t)$ : Classifier gradient (direction toward class  $\mathbf{y}$ )
- $\omega$ : Guidance scale (how strong the steering is)
- $\sqrt{1 - \bar{\alpha}_t}$ : Noise level adjustment

**Subtract classifier gradient to steer toward desired class! →**



# Classifier Guidance Sampling Algorithm

**Input:** Unconditional model  $\epsilon_\theta$ , classifier  $p_\phi$ , target class  $\mathbf{y}$ , guidance scale  $\omega$

Sample  $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

**for**  $t = T, T - 1, \dots, 1$  **do**

    // Step 1: Get unconditional noise prediction

$$\epsilon_u = \epsilon_\theta(\mathbf{x}_t, t)$$

    // Step 2: Compute classifier guidance

$$\mathbf{g} = \nabla_{\mathbf{x}_t} \log p_\phi(\mathbf{y}|\mathbf{x}_t)$$

▷ Requires backprop!

    // Step 3: Apply guidance to noise prediction

$$\tilde{\epsilon} = \epsilon_u - \omega \sqrt{1 - \bar{\alpha}_t} \mathbf{g}$$

    // Step 4: Standard DDIM update with guided noise

$$\hat{\mathbf{x}}_0 = \frac{\mathbf{x}_t - \sqrt{1 - \bar{\alpha}_t} \tilde{\epsilon}}{\sqrt{\bar{\alpha}_t}}$$

$$\mathbf{x}_{t-1} = \sqrt{\bar{\alpha}_{t-1}} \hat{\mathbf{x}}_0 + \sqrt{1 - \bar{\alpha}_{t-1}} \tilde{\epsilon}$$

**end for**

**Return:**  $\mathbf{x}_0$

# The Classifier Challenge

## The noisy data problem

- Standard classifiers work on clean images
- During diffusion sampling, images are heavily corrupted with noise
- A clean-image classifier gives nonsensical gradients on noisy data
- Need a classifier that works across ALL noise levels


## Solution: Noise-aware classifier training

- Train classifier on noisy images at all timesteps
- Input:  $(\mathbf{x}_t, t)$  where  $\mathbf{x}_t$  has noise level for timestep  $t$
- Output: Class probabilities despite the noise
- Same noise schedule as the diffusion model

**Training a robust classifier is non-trivial! **

# Classifier Guidance: The Guidance Scale Knob

The guidance scale  $\omega$  controls the quality-diversity trade-off...

**Low guidance ( $\omega \rightarrow 0$ ):** 

- High sample diversity
- Natural-looking images
- Weak conditioning adherence
- Like unconditional generation

**High guidance ( $\omega \gg 1$ ):** 

- Strong conditioning adherence
- Clear class characteristics
- Lower sample diversity
- Risk of artifacts

## Typical values

- $\omega = 1$ : Theoretically correct Bayesian inference
- $\omega \in [2, 10]$ : Common practical range
- Higher values: More conditioning strength but potential artifacts

**Finding the sweet spot requires experimentation!** 

## ✓ Advantages:

- **Modular:** Works with any pre-trained model
- **Flexible:** Multiple classifiers for different attributes
- **No retraining:** Use existing diffusion models
- **Strong control:** High-quality conditional generation

## ✗ Disadvantages:

- **Slow:** Extra classifier forward + backward pass
- **Complex:** Need robust noise-aware classifiers
- **Memory:** Store gradients for guidance
- **Brittle:** Sensitive to classifier quality

Powerful but complex - is there a simpler way?

Yes! Classifier-free guidance eliminates the classifier entirely... 

## Approach 3: Classifier-Free Guidance - The Breakthrough ⚡

What if we could get classifier guidance **WITHOUT** a classifier?

The revolutionary insight 🧠

From Bayes' rule:  $\nabla_{\mathbf{x}_t} \log p(\mathbf{y}|\mathbf{x}_t) = \nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t|\mathbf{y}) - \nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t)$

**The classifier gradient is just the difference between conditional and unconditional scores!**

The elegant solution ✎

- Train ONE model to do BOTH conditional and unconditional generation
- During training, randomly drop conditioning with probability  $p_{\text{uncond}}$
- During sampling, compute both predictions and take their difference
- No separate classifier needed!

**One model, unlimited control! ∞**

## The conditioning dropout trick

**Input:** Dataset  $\{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}$ , dropout probability  $p_{\text{uncond}} = 0.1$

**while** not converged **do**

    Sample batch  $\{(\mathbf{x}, \mathbf{y})\}$  from training dataset

    Sample timesteps  $t \sim \text{Uniform}(1, T)$  and noise  $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

    Compute noisy samples:  $\mathbf{x}_t = \sqrt{\bar{\alpha}_t} \mathbf{x} + \sqrt{1 - \bar{\alpha}_t} \epsilon$

**for** each sample  $(\mathbf{x}_t, \mathbf{y}, t)$  in batch **do**

**if**  $\text{random}() < p_{\text{uncond}}$  **then**

            Set  $\mathbf{y} = \emptyset$

**end if**

**end for**

    Predict noise:  $\hat{\epsilon} = \epsilon_{\theta}(\mathbf{x}_t, \mathbf{y}, t)$

    Compute loss:  $\mathcal{L} = \|\epsilon - \hat{\epsilon}\|^2$

    Update parameters

**end while**

- ▷ Randomly drop conditioning
- ▷ Use null token

**Model learns to handle both real conditions and null conditions! ✓**

## How to combine conditional and unconditional predictions...

The classifier-free guidance equation 🧮

$$\tilde{\epsilon}(\mathbf{x}_t, \mathbf{y}, t) = (1 + \omega)\epsilon_{\theta}(\mathbf{x}_t, \mathbf{y}, t) - \omega\epsilon_{\theta}(\mathbf{x}_t, \emptyset, t)$$

Intuitive understanding 🎯

Rewrite as:  $\tilde{\epsilon} = \epsilon_{\text{uncond}} + (1 + \omega)[\epsilon_{\text{cond}} - \epsilon_{\text{uncond}}]$

- Start with unconditional prediction
- Compute the “conditioning direction”
- Amplify this direction by  $(1 + \omega)$
- Move further in the conditioning direction!

**Extrapolation beyond normal conditioning for stronger control! ⬆**

# CFG Sampling Algorithm

**Input:** Model  $\epsilon_\theta$ , condition  $\mathbf{y}$ , guidance scale  $\omega$ , timesteps  $\{\tau_1, \dots, \tau_S\}$

Sample  $\mathbf{x}_{\tau_S} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

**for**  $i = S, S-1, \dots, 1$  **do**

$t = \tau_i, s = \tau_{i-1}$

    // Step 1: Get both predictions

$\epsilon_c = \epsilon_\theta(\mathbf{x}_t, \mathbf{y}, t)$

$\epsilon_u = \epsilon_\theta(\mathbf{x}_t, \emptyset, t)$

    // Step 2: Apply classifier-free guidance

$\hat{\epsilon} = (1 + \omega)\epsilon_c - \omega\epsilon_u$

    // Step 3: Standard DDIM update

$\hat{\mathbf{x}}_0 = \frac{\mathbf{x}_t - \sqrt{1 - \bar{\alpha}_t} \hat{\epsilon}}{\sqrt{\bar{\alpha}_t}}$

$\mathbf{x}_s = \sqrt{\bar{\alpha}_s} \hat{\mathbf{x}}_0 + \sqrt{1 - \bar{\alpha}_s} \hat{\epsilon}$

**end for**

**Return:**  $\mathbf{x}_0$

▷ Conditional  
▷ Unconditional

Two forward passes per step, but no classifier needed! ✓



# CFG vs. Classifier Guidance: The Comparison

## How do these two approaches stack up?

### Classifier Guidance

- + Works with any pre-trained model
- + Modular approach
- Requires noise-aware classifier
- Slower (classifier + gradients)
- Complex implementation

### Classifier-Free Guidance

- + No separate classifier needed
- + Simpler implementation
- + Better text conditioning
- Requires retraining
- 2x forward passes

### The verdict

**CFG has become the dominant approach** - powers Stable Diffusion, DALL-E, and most modern text-to-image models!

**CFG enables the text-to-image revolution!** 

# The Three Approaches: When to Use What?

Choosing the right conditioning approach for your application...

## Class-conditional models

**Use when:** Simple categorical control, speed critical, limited compute **Examples:** Medical imaging, game assets, scientific visualization

## Classifier guidance

**Use when:** Retrofitting existing models, research applications, multiple objectives  
**Examples:** Adding control to pre-trained models, experimental setups

## Classifier-free guidance

**Use when:** Complex conditioning, production systems, text-to-image **Examples:** Stable Diffusion, DALL-E, commercial creative tools

**Modern default: CFG for most new applications!** 

# Example: Stable Diffusion








## Prompt

cartoon character of a person with a hoodie , in style of cytus and deemo, ork, gold chains, realistic anime cat, dripping black goo, lineage revolution style, thug life, cute anthropomorphic bunny, balrog, arknights, aliased, very buff, black and red and yellow paint, painting illustration collage style, character composition in vector with white background.

# Summary: Conditional Generation Mastered

## What we've learned today ✓

-  **Class-conditional:** Simple embedding-based conditioning
-  **Classifier guidance:** External steering with gradients
-  **Classifier-free guidance:** The modern standard
-  **U-Net architecture:** The backbone enabling all conditioning
-  **Text conditioning:** Cross-attention for language control



## Key practical insights

- CFG with cross-attention dominates modern applications
- Proper conditioning dropout is crucial for CFG training
- Guidance scale controls the quality-diversity trade-off
- U-Net's hierarchical structure enables multi-scale conditioning

**We can now build controllable, practical diffusion systems! **

# Next Session Preview: Latent Diffusion & Efficiency

**We'll explore how to make diffusion models practical and scalable:**

- How does Stable Diffusion work in latent space?
- What are the key efficiency techniques for production deployment? 
- How do we handle ultra-high resolution generation?
- What are the latest advances in fast sampling? 

**From beautiful theory to real-world systems  
that serve millions of users! **

**Ready to scale diffusion to the real world? **