Lecture 16:

Generative Models - Diffusion

Ho-min Park

homin.park@ghent.ac.kr

powersimmani@gmail.com

Lecture Contents

Part 7: Applications and Extensions

Part 1/7:

Introduction & Motivation

1. Evolution of Generative Models
2. Intuitive Understanding
3. Comparison with GANs

2013

VAEs

Probabilistic encoding with KL divergence

→

2014

GANs

Adversarial training approach

→

2015-2020

Diffusion

Gradual noise addition & removal

→

2018

Flow-based

Invertible transformations

→

2020+

Score-based

Score matching connection

🎯 Recent Success (2022-2023)

DALL-E 2 | Stable Diffusion | Midjourney

Key Advantages: High quality • Stable training • Mode coverage

Like a sculptor gradually refining marble from rough block to statue

Forward Process	Reverse Process
Original Data x_0 Clear, structured data	**Pure Noise x_t** Start from random noise
+ Gaussian Noise Systematically add noise	**Neural Network** Learn denoising function
Intermediate States Gradually corrupted data	**Iterative Denoising** Step-by-step noise removal
Pure Noise x_t Complete randomization	**Generated Data x_0** Clean, realistic output

Markov Chain

Key Insights

✓ Easier to learn small denoising steps

✓ Each step depends only on previous

✓ More controllable than direct generation

Diffusion Models vs GANs

Comprehensive Feature Comparison

Criterion	Diffusion Models	GANs
⚖️ Training Stability	More stable, no adversarial collapse ✓	Can be unstable, risk of mode collapse
🎯 Mode Coverage	Captures all modes effectively ✓	May miss some modes
✨ Sample Quality	Comparable, often more detailed ✓	High quality, sometimes less detailed
⚡ Generation Speed	Slower (100-1000 steps)	Faster (1 step generation)
🔧 Training Complexity	Simpler loss, no discriminator ✓	Complex, requires discriminator tuning

☐ Advantage ✓ Winner in this category

Part 2/7:

Forward Process

4. Forward Process Definition
5. Noise Schedule
6. Forward Process Properties
7. Reparameterization Trick
8. Mathematical Foundation

Forward Process Definition

Part 2/7: Forward Process

🎯 Goal: Gradually transform data into pure Gaussian noise

🔗 Markov Chain: $q(x_t \mid x_{t-1})$

$t = 1, 2, \ldots, T$

$\longrightarrow \longrightarrow \longrightarrow$

x_0
Original Data

x_t
$\sim N(0, I)$

📐 Gaussian Transition (Distribution Form)

$$q(x_t \mid x_{t-1}) = N(x_t; \sqrt{\alpha_t} \cdot x_{t-1}, (1-\alpha_t) I) \quad \text{where } \alpha_t = 1 - \beta_t$$

🎲 Sampling Form (Reparameterization)

$$x_t = \sqrt{\alpha_t} \cdot x_{t-1} + \sqrt{(1-\alpha_t)} \cdot \varepsilon, \text{ where } \varepsilon \sim N(0, I)$$

⚙️ Noise Schedule (β_t)

Controls amount of noise at step t

📈 Fixed Schedule

β_t increases from 0.0001 to 0.02

⏱️ Time Steps (T)

Usually T = 1000 steps

✨ End Result

$x_t \sim N(0, I)$ is pure Gaussian noise

🧮 Concrete Example: 4D Vector Forward Process

Using larger β values (0.1→0.2→0.3) for visualization. Real diffusion uses smaller values over 1000 steps.

x_0 (Original)

$\|x\|$ = 2.45

Original 4D Vector

1.00	2.00	-1.00	0.50

📌 This is our **starting data point**

📌 Could represent: image pixel values, latent code, etc.

x_1 (t=1)

$\|x\|$ = 2.41

β_1=0.10, α_1=0.90

1.11	1.80	-0.70	0.41

$\sqrt{\alpha_1}$ = **0.949**, $\sqrt{(1-\alpha_1)}$ = **0.316**

$\varepsilon_1 \sim N(0,I)$ = [0.5, -0.3, 0.8, -0.2]

$x_1 = 0.949 \cdot x_0 + 0.316 \cdot \varepsilon_1$

↓ Add more noise ↓

x_2 (t=2)

$\|x\|$ = 2.19

β_2=0.20, α_2=0.80

0.79	1.43	-0.81	0.82

$\sqrt{\alpha_2}$ = **0.894**, $\sqrt{(1-\alpha_2)}$ = **0.447**

$\varepsilon_2 \sim N(0,I)$ = [-0.4, -0.2, -0.4, 1.0]

$x_2 = 0.894 \cdot x_1 + 0.447 \cdot \varepsilon_2$

x_3 (t=3)

$\|x\|$ = 1.87

β_3=0.30, α_3=0.70

0.41	1.45	-0.42	0.55

$\sqrt{\alpha_3}$ = **0.837**, $\sqrt{(1-\alpha_3)}$ = **0.548**

$\varepsilon_3 \sim N(0,I)$ = [-0.5, 0.3, 0.5, -0.2]

$x_3 = 0.837 \cdot x_2 + 0.548 \cdot \varepsilon_3$

💡 Key Observations from the Example

- **Signal decay:** $\sqrt{\alpha_t}$ shrinks the previous value → original information gradually fades
- **Noise accumulation:** $\sqrt{(1-\alpha_t)}$ scales random noise → randomness increases each step
- **Norm trend:** Vector magnitude ($\|x\|$) changes as data transforms toward N(0, I)
- **After T=1000 steps:** With real β schedule (0.0001→0.02), x_T ≈ pure Gaussian noise

Cumulative Effect

Part 2/7: Forward Process

✨ Closed Form Solution

Jump directly from x_0 to any x_t without iterating through all steps

✗ Iterative (Slow)

Need all intermediate steps

✓ Direct Sampling (Fast)

One-step computation

Mathematical Formulation

Alpha definitions:

$$\alpha_t = 1 - \beta_t$$

Cumulative product:

$$\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$$

Direct Sampling Formula:

$$x_t = \sqrt{\bar{\alpha}_t} \cdot x_0 + \sqrt{(1-\bar{\alpha}_t)} \cdot \varepsilon$$

Efficient Training

Signal-to-Noise Control

Reparameterization

Mathematical Properties

Part 2/7: Forward Process

Mean Preservation

$$\mathbb{E}[x_t] = \sqrt{\bar{\alpha}_t} \cdot x_0$$

Expected value scaled by $\sqrt{\bar{\alpha}_t}$

Variance Growth

$$\mathrm{Var}[x_t] = (1 - \bar{\alpha}_t) \cdot \mathbf{I}$$

Variance increases as noise accumulates

Gaussian Distribution

$$q(x_t|x_0) = \mathcal{N}(x_t;\ \sqrt{\bar{\alpha}_t}\, x_0,\ (1 - \bar{\alpha}_t)\mathbf{I})$$

Complete distributional characterization

Posterior Distribution

$$q(x_{t-1}|x_t, x_0) = \mathcal{N}(x_{t-1};\ \tilde{\mu}_t,\ \tilde{\beta}_t\mathbf{I})$$

Also Gaussian with closed-form solution

Tractable Likelihood

$$\log p_\theta(x_0) \geq \mathcal{L}_{\mathrm{ELBO}}$$

Can compute exact log-likelihood for model evaluation and comparison

Reversibility

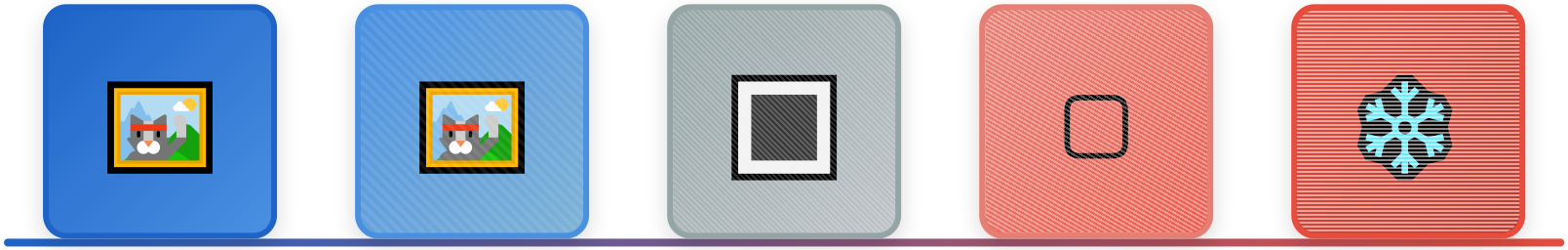
$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1};\ \mu_\theta,\ \Sigma_\theta)$$

Theoretical foundation enables the reverse generative process

All properties maintain Gaussian structure, enabling tractable inference and generation

Forward Process Visualization

Part 2/7: Forward Process



t = 0

Original clear image

t = 250

Slight noise, details visible

t = 500

Moderate noise, structure recognizable

t = 750

Heavy noise, vague outline

t = 1000

Pure noise, no information

📉 Information Loss

Gradual and controlled degradation over timesteps

✅ Visual Inspection

Validates noise schedule effectiveness

Why is this Process Necessary?

Part 2/7: Forward Process

🤔 Understanding the Fundamental Motivation

📊 Tractable Posterior

Known distribution $q(x_{t-1}|x_t,x_0)$ enables efficient training

📚 Curriculum Learning

Start easy (small noise), gradually increase difficulty

⚡ Stable Gradients

Small steps ensure smooth gradient flow during training

🎯 Mode Coverage

Noise prevents collapse to single mode

🧮 Theoretical Foundation

Enables variational bound derivation for optimization

🔧 Flexible Design

Can adjust β_t schedule for different data types

🔄 Reversibility

Ensures reverse generative process is well-defined

The forward process creates a tractable learning framework that makes the reverse process trainable

Part 3/7:

Reverse Process

9. Reverse Process Goal
10. Neural Network Parameterization
11. Deep Dive into Score Function
12. Denoising Objective Function
13. Training Algorithm

Reverse Process Goal

Part 3/7: Reverse Process

🎯 **Objective: Learn to reverse the forward diffusion process**

Start Point

$x_T \sim N(0, I)$

Pure Noise

$p_\theta(x_{t-1} \mid x_t)$

Iterative Denoising

→

Neural Network θ

End Point

$x_0 \sim p_{data}(x)$

Data Sample

Markov Chain

Parameterized by neural network

Conditional Distribution

Model Gaussian transitions

Denoising Direction

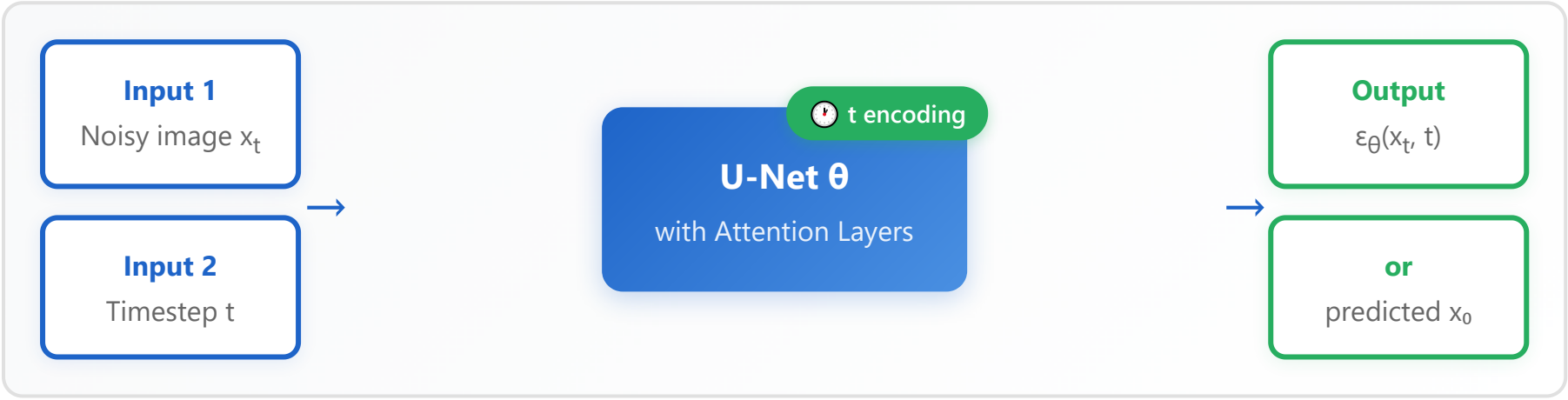
Remove added noise

Generation Process

Iteratively denoise from noise to create new high-quality samples

Neural Network Parameterization

Part 3/7: Reverse Process



Mean Prediction

$$\mu_\theta(x_t, t) = 1/\sqrt{\alpha_t} \cdot (x_t - (1-\alpha_t)/\sqrt{(1-\bar{\alpha}_t)} \cdot \varepsilon_\theta(x_t, t))$$

Variance

σ_t^2 (Fixed or Learned)

Reverse Transition Distribution

$$p_\theta(x_{t-1}|x_t) = N(x_{t-1}; \mu_\theta(x_t, t), \sigma_t^2 I)$$

Deep Dive into Score Function

Part 3/7: Reverse Process

📊 Score Definition

$$\nabla_x \log p(x)$$

Indicates direction of density increase

Connection

Denoising ≈ Score matching

Tweedie's Formula

Score relates to optimal denoising estimation

Score Estimation

$$\varepsilon_\theta(x_t,t) \approx -\sqrt{1-\bar{\alpha}_t} \cdot \nabla_{x_t} \log p(x_t)$$

Langevin Dynamics

Reverse process as Langevin sampling from the score function

Continuous Limit

Score-based SDEs when T → ∞

Unified View

Diffusion and score-based models are equivalent

Denoising Objective Function

Part 3/7: Reverse Process

Simplified Loss

✓ Preferred

$$L_{simple} = E_{t,\, x_0,\, \varepsilon}\left[\, || \,\varepsilon - \varepsilon_\theta(x_t,\, t)\, ||^2 \,\right]$$

Works better in practice

Variational Lower Bound

$$L_{vlb} = \Sigma_t\, E_q\left[\, D_{KL}(\, q(x_{t-1}|x_t,\, x_0) \,||\, p_\theta(x_{t-1}|x_t)\,)\,\right]$$

Theoretical foundation (ELBO)

🎯 Practical Training Process

1 Sample Timestep

$t \sim Uniform\{1, \ldots, T\}$

→

2 Sample Noise

$\varepsilon \sim N(0, I)$

→

3 Create Noisy Sample

$x_t = \sqrt{\bar{\alpha}_t} \cdot x_0 + \sqrt{(1-\bar{\alpha}_t)} \cdot \varepsilon$

→

4 Gradient Descent

Update θ to minimize loss

⚡ Optimization Objective

$$minimize \;||\, \varepsilon - \varepsilon_\theta(x_t,\, t)\, ||^2$$

📐 4D Vector Calculation Example

1 Initial Data & Parameters

$x_0 =$ [0.8, −0.5, 0.3, 1.2]

$t =$ 50 (with T=100)

$\bar{\alpha}_t =$ 0.64 ($\sqrt{\bar{\alpha}_t}$ = 0.8)

2 Noise Sampling

$\varepsilon \sim$ N(0, I)

$\varepsilon =$ [0.2, −0.7, 1.1, −0.4]

$\sqrt{(1-\bar{\alpha}_t)} = \sqrt{0.36} = 0.6$

3 Create Noisy Sample

$$x_t = \sqrt{\bar{\alpha}_t} \cdot x_0 + \sqrt{(1-\bar{\alpha}_t)} \cdot \varepsilon$$
$$= 0.8 \times [0.8, -0.5, 0.3, 1.2] + 0.6 \times [0.2, -0.7, 1.1, -0.4]$$
$$= [0.64, -0.40, 0.24, 0.96] + [0.12, -0.42, 0.66, -0.24]$$

$x_t =$ [0.76, −0.82, 0.90, 0.72]

4 Model Prediction & Loss

$\varepsilon_\theta(x_t, t) =$ [0.3, −0.5, 0.9, −0.3]

$\varepsilon - \varepsilon_\theta = [0.2-0.3, -0.7-(-0.5), 1.1-0.9, -0.4-(-0.3)]$

Difference = [−0.1, −0.2, 0.2, −0.1]

MSE Loss Calculation

$$L = \| \varepsilon - \varepsilon_\theta \|^2 = (-0.1)^2 + (-0.2)^2 + (0.2)^2 + (-0.1)^2$$
$$= 0.01 + 0.04 + 0.04 + 0.01$$

L = 0.10

Key Point: The model ε_θ predicts the original noise ε by only observing x_t and t. As the loss decreases, noise prediction becomes more accurate, enabling the reverse process to recover clean images.

Training Algorithm

Part 3/7: Reverse Process

1

Sample Data

2

Sample Timestep

3

Sample Noise

4

Add Noise

5

Predict Noise

Neural Network

6

Compute Loss

7

Backprop

Update parameters

8

Repeat Until Convergence

Loop through steps 1-7

💡 Core Training Loop: Sample → Add Noise → Predict → Optimize → Repeat

Concrete Calculation Example

2D Case with Actual Numbers

Step-by-Step Calculation

Step 1: Sample Data Point

Original data point from training set

Step 2: Sample Timestep

(out of)

Random timestep, halfway through diffusion

Step 3: Sample Noise

Random Gaussian noise vector

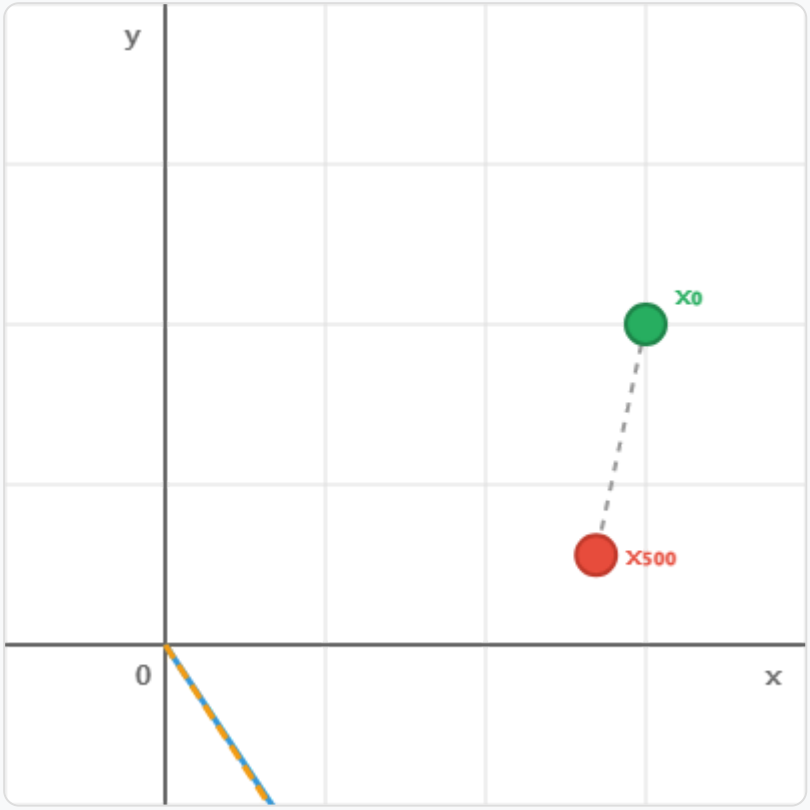
Step 4: Add Noise (Forward Process)

(example)

Step 5: Predict Noise

Neural network prediction

Visual Representation



Legend

- Original data
- Noisy data
- True noise
- Predicted noise

Step 6: Compute Loss

Step 7: Update Parameters

Gradient descent to minimize loss

Goal: Make closer to by updating

Part 4/7:

Sampling

14. DDPM Sampling
15. DDIM - Fast Sampling
16. Conditional Generation
17. Classifier-Free Guidance

DDPM Sampling

Part 4/7: Sampling

1 Initialize from Pure Noise

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

Iterative Denoising

$t = T, T-1, \ldots, 1$

① Predict Noise

$\hat{\boldsymbol{\epsilon}} = \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$

② Compute Mean

$\boldsymbol{\mu}_\theta(\mathbf{x}_t, t)$

③ Sample Noise

$\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$

④ Update State

$\mathbf{x}_{t-1} = \boldsymbol{\mu}_\theta(\mathbf{x}_t, t) + \sigma_t \cdot \mathbf{z}$

3 Final Output

$\mathbf{x}_0$ is the generated sample

⏱️ Time Complexity: $\mathcal{O}(T)$ forward passes • Typically $T = 1000$

DDIM - Fast Sampling

Part 4/7: Sampling

⚠ Problem

DDPM requires many steps (1000) for good quality

✅ Solution

DDIM uses deterministic, non-Markovian process

DDPM

- All timesteps (1000)
- Stochastic process
- Random noise added
- Slower generation

DDIM

⚡ 20x Speedup

- ✓ Subset only (e.g., 50 steps)
- ✓ Deterministic ($\sigma_t = 0$)
- ✓ Skip: $\tau = [1,20,40,...,1000]$
- ✓ Comparable quality

📐 DDIM Update Formula

$$x_{t-1} = \sqrt{\bar{\alpha}_{t-1}} \cdot \hat{x_0} + \sqrt{(1-\bar{\alpha}_{t-1})} \cdot \hat{\varepsilon_t}$$

x_{t-1}: previous step sample | $\bar{\alpha}_{t-1}$: cumulative noise schedule | $\hat{x}_0$: predicted original image | $\hat{\varepsilon_t}$: predicted noise

⚖️ Trade-off: Can interpolate between DDPM and DDIM for speed-quality balance

Conditional Generation

Part 4/7: Sampling

🎯 **Goal: Generate samples conditioned on class, text, or image**

🔍 Classifier Guidance Method

Base Score (original) + **Classifier Gradient** = **Modified Score**

Modified Score Formula

Guidance Scale

controls conditioning strength

✅ Applications

- Class-conditional ImageNet generation

⚠️ Limitations

- Requires trained classifier
- Less flexible
- Need separate classifier for each type

💡 **Key Insight: Use classifier gradients to guide generation toward desired conditions**

📐 Vector Calculation Example

Input Values

Base Score : `[0.5, -0.3, 0.2]`

Classifier Gradient : `[0.1, 0.4, -0.2]`

Guidance Scale : `2.0`

Step-by-Step Calculation

Step 1: Scale the Gradient

Step 2: Add to Base Score

Final Modified Score

Interpretation

The Classifier Gradient represents the direction toward the desired class (e.g., "cat"). By setting **w=2.0**, we push more strongly in this direction, making the generated image closer to the target class. Higher values of w lead to stronger conditioning, but setting it too high may degrade image quality.

Classifier-Free Guidance

Part 4/7: Sampling

Innovation: No separate classifier needed

Training Strategy

Randomly drop condition c with probability p (e.g., 10%)

Unconditional

$\varepsilon_\theta(x_t, t, \emptyset)$

+

Weight

w

×

Difference

$(\varepsilon_\theta(x_t, t, c) - \varepsilon_\theta(x_t, t, \emptyset))$

Guided Prediction Formula

$$\tilde{\varepsilon} = \varepsilon_\theta(x_t, t, \emptyset) + w \cdot (\varepsilon_\theta(x_t, t, c) - \varepsilon_\theta(x_t, t, \emptyset))$$

Guidance Scale

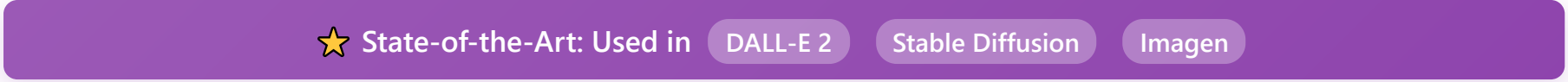
w > 1 increases strength

Trade-off

Higher w = better align, less diverse

Flexibility

Single model, multiple conditions


State-of-the-Art: Used in
DALL-E 2
Stable Diffusion
Imagen

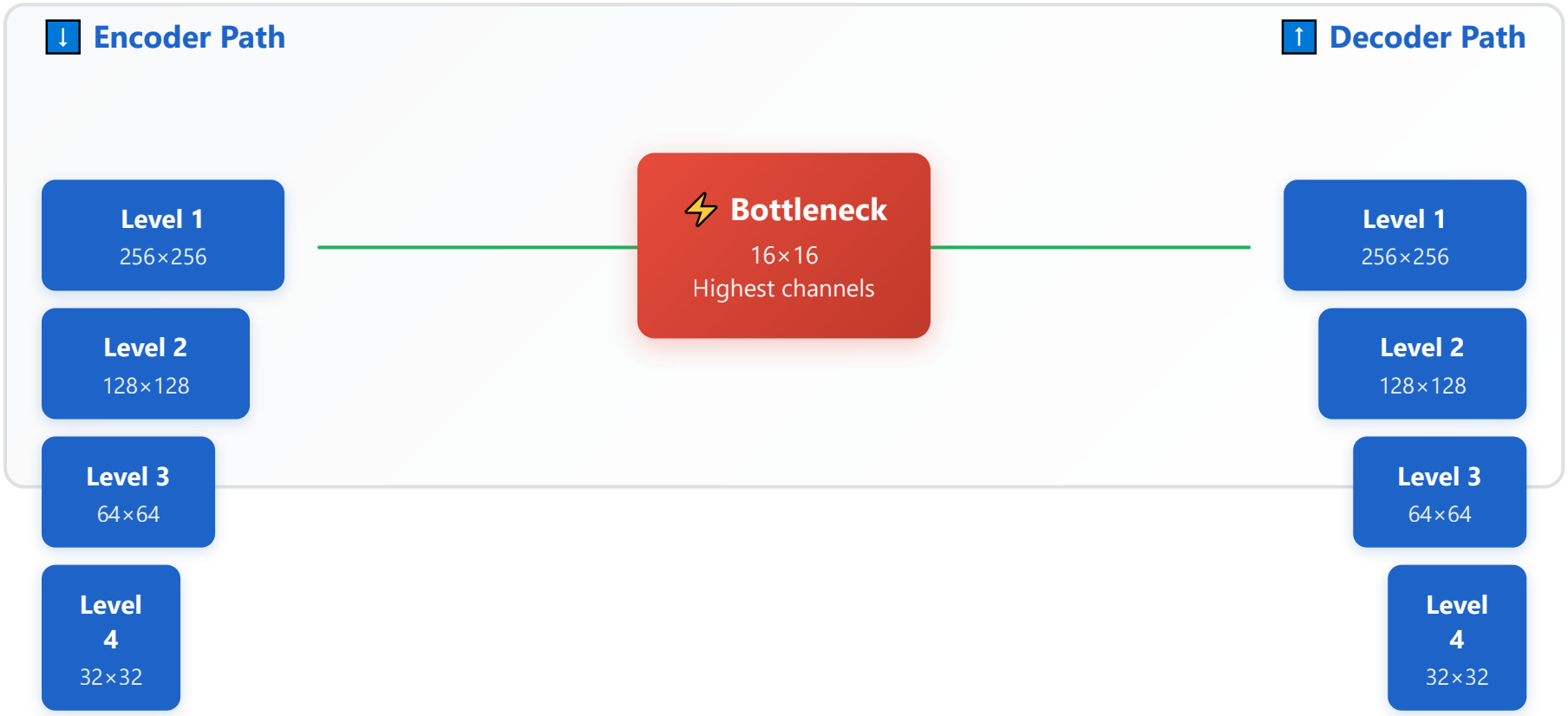
Part 5/7:

Architecture

18. U-Net Structure
19. Attention Mechanism
20. Condition Injection Methods

U-Net Structure

Part 5/7: Architecture



ResNet Blocks

Normalization

Time Embedding

Concatenation


Two conv + residual
Group norm
Adaptive norm
Encoder features

Attention Mechanism

Part 5/7: Architecture

Self-Attention

Q | K | V

Models long-range dependencies in image. Applied at 16×16, 32×32 resolutions.

Cross-Attention

Text Q → Image K,V

For text conditioning. Text→Image attention enables controllable generation.

Key Properties

Multi-Head
Multiple patterns

Spatial Attention
Different locations

Complexity
$O(n^2)$

Mechanism
Standard transformer

Location
Low resolution

Dependencies
Long-range

✨ Impact: Crucial for coherent, high-quality generation

Condition Injection Methods

Part 5/7: Architecture

Time Embedding

Sinusoidal positional encoding + MLP

Class Embedding

Learned embedding for each class

Text Conditioning

CLIP or T5 text encoder → cross-attention

Adaptive Group Norm

Scale and shift from condition embedding

FiLM Layers

Feature-wise Linear Modulation

Concatenation

Directly concat condition to input channels

Cross-Attention

Text tokens attend to image features

Hybrid Approaches

Combine multiple conditioning methods for optimal performance

Categorization by Type

Temporal

Time Embedding

Categorical

Class Embedding

Textual

Text + Cross-Attention

Modern models often combine multiple injection methods for best results

Part 6/7:

Advanced Techniques

21. Latent Diffusion (Stable Diffusion)
22. Noise Schedule Improvements
23. Other Improvement Techniques

Latent Diffusion (Stable Diffusion)

Part 6/7: Advanced Techniques

⚠️ Motivation: Pixel-space diffusion is computationally expensive

Efficiency
64x speedup

Compression
8×8 reduction

Quality
Perceptual loss

Memory
Consumer GPUs

🎨 Stable Diffusion: Open-source latent diffusion model — High-resolution generation

📊 Practical Example: 512×512 Image Generation

Pixel-space Diffusion

Input size: `512 × 512 × 3 = 786,432`

VS

Latent Diffusion (8× compression)

Compressed size: `64 × 64 × 4 = 16,384`

Memory per step: ~3 MB

50 steps total: ~150 MB ✗

Memory per step: ~65 KB

50 steps total: ~3.2 MB ✓

Memory Reduction: 48× | Speed Improvement: 64× | Quality: Preserved via perceptual loss

Interactive Demo: **Diffusion Explainer**

Noise Schedule Improvements

Part 6/7: Advanced Techniques

📈 Evolution of Noise Schedules

📏 Original

Linear Schedule

Original DDPM approach

```
β_t: 1e-4 → 0.02
```

〰️ Improved

Cosine Schedule

Smoother transition, better for high-res

🧠 Advanced

Learned Schedule

Network learns optimal β_t during training

🔬 Key Concepts & Innovations

📊 **Signal-to-Noise Ratio**

SNR(t) = ᾱ_t / (1-ᾱ_t)

🎯 **V-parameterization**

Predict velocity v_t instead of noise

⚡ **EDM Framework**

Exponential noise for better scaling

♾️ **Continuous Time**

Formulate as continuous SDE

✨ Impact: Better sample quality and training stability

Other Improvement Techniques

Part 6/7: Advanced Techniques

Cascaded Diffusion

Low-res generation → super-resolution

Upsampler

Specialized diffusion model for upscaling

Progressive Distillation

Student mimics T-step teacher in T/2 steps

Consistency Models

One-step generation via consistency training

Flow Matching

Alternative to diffusion with optimal transport

Rectified Flow

Straightens probability paths for efficiency

Adversarial Diffusion

Add adversarial loss for sharper samples

Truncation Tricks

Early stopping or partial diffusion for control

Categorization by Purpose

Quality

Speed

Alternative

Control

3 techniques

2 techniques

2 techniques

1 technique

Part 7/7:

Applications and Extensions

Applications and Conclusion

Part 7/7: Applications and Extensions

Text-to-Image Popular

DALL-E 2, Stable Diffusion, Midjourney, Imagen

Image Editing

Inpainting, outpainting, style transfer

Video Generation

Temporal diffusion models

3D Generation

NeRF + diffusion (DreamFusion)

Audio Synthesis

Diffusion for music and speech generation

Molecular Design

Drug discovery and protein design

Medical Imaging

Denoising, reconstruction, synthesis

Future Directions

- Real-time Generation
- Better Control
- Improved Efficiency

Thank you

Ho-min Park

homin.park@ghent.ac.kr

powersimmani@gmail.com