

Generative AI: Diffusion Model

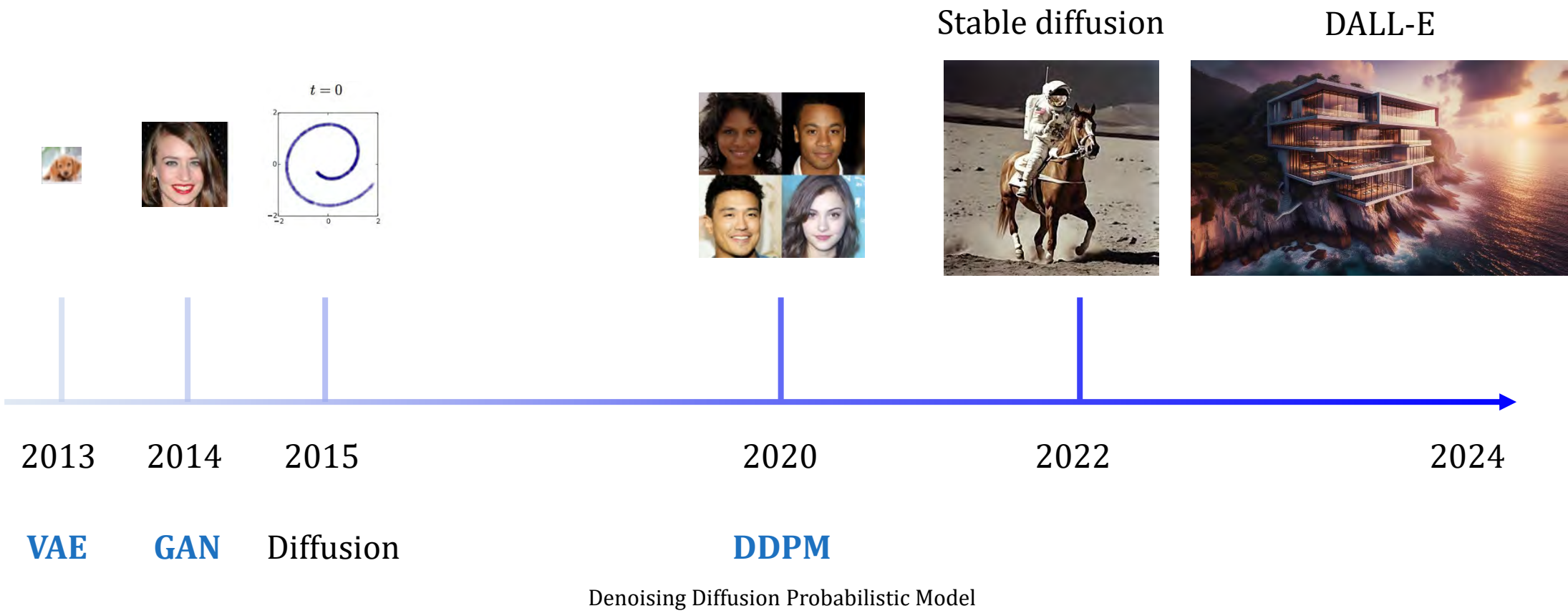
Prof. Sooyoung Lee

School of Mechanical Engineering
Chung-Ang University

Agenda

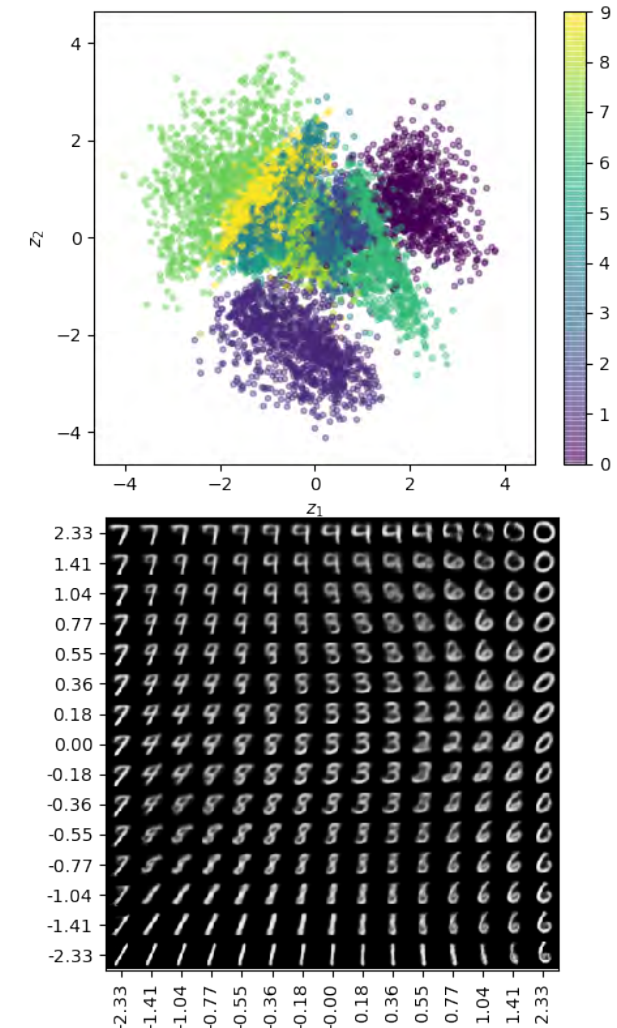
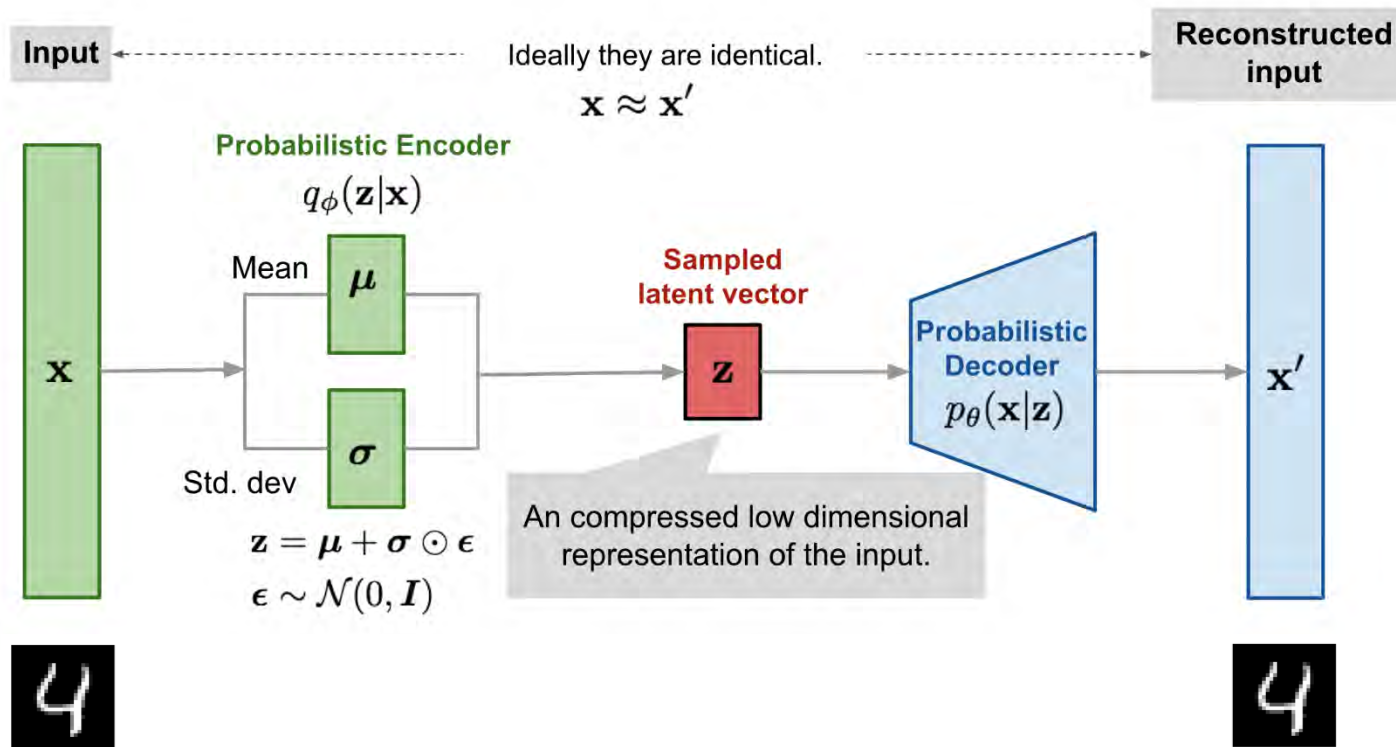
- Diffusion Model
- Engineering Applications of Diffusion Model

Generative Models Timeline



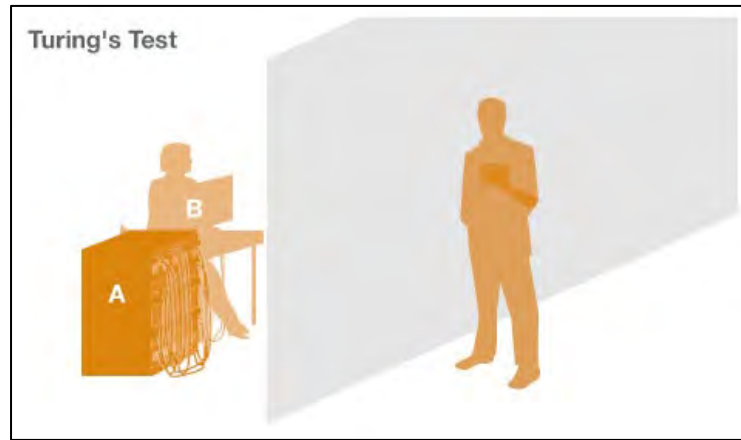
Generative Models: (1) VAE (Variational Autoencoder)

- Encoder-decoder structure with a prior distribution, where latent space enables both efficient representation learning and generative sampling

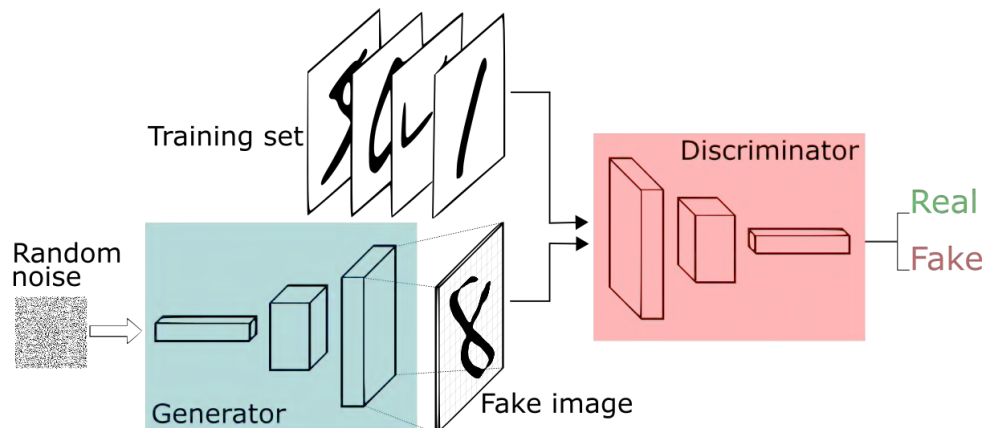
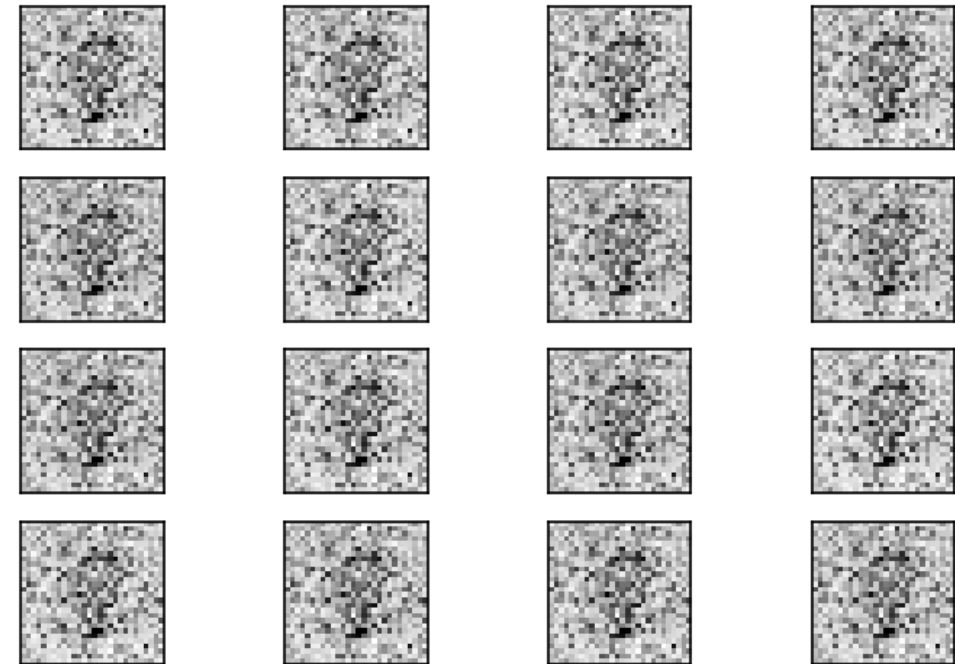


Generative Models: (2) GAN (Generative Adversarial Network)

- A generator and a discriminator are trained simultaneously in a minmax game, where the generator aims to produce realistic data to fool the discriminator

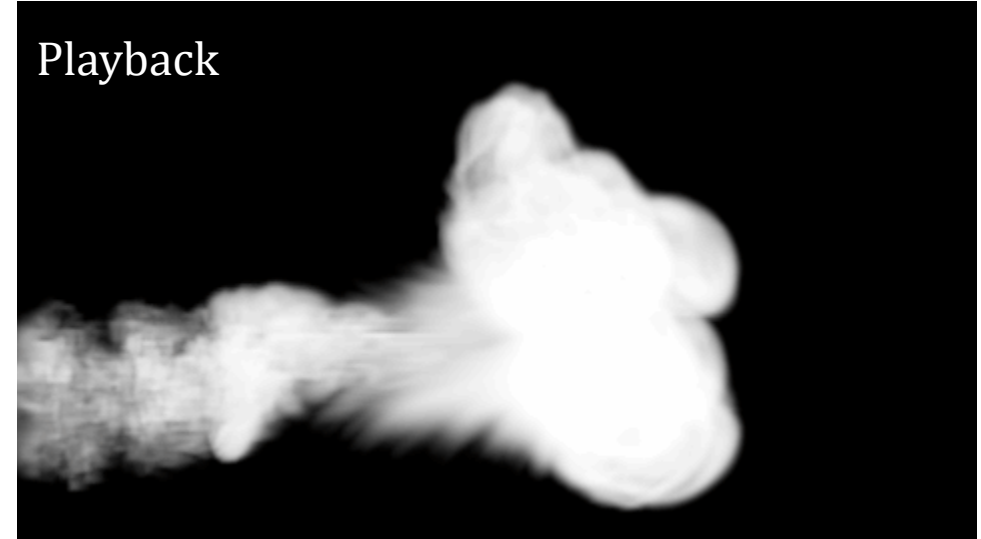
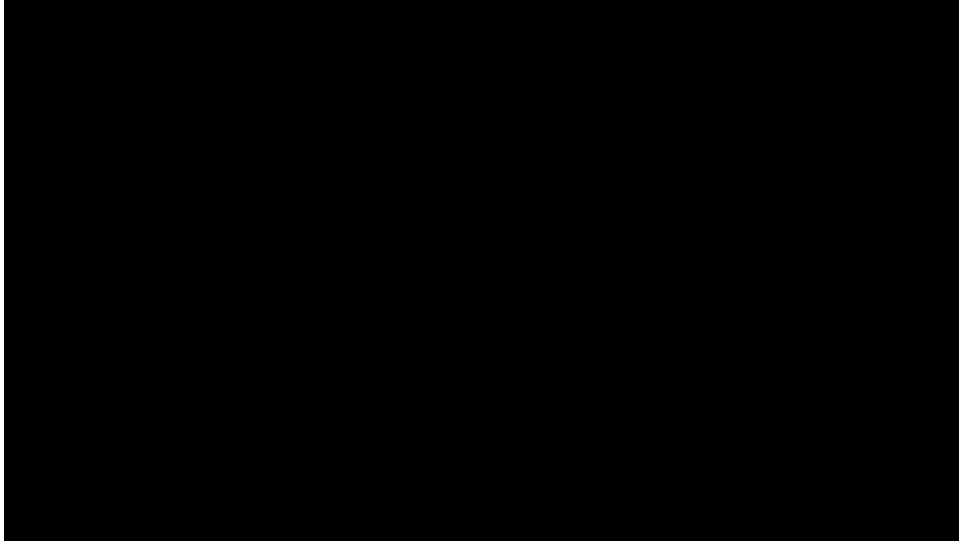


After 1 epoch(s)

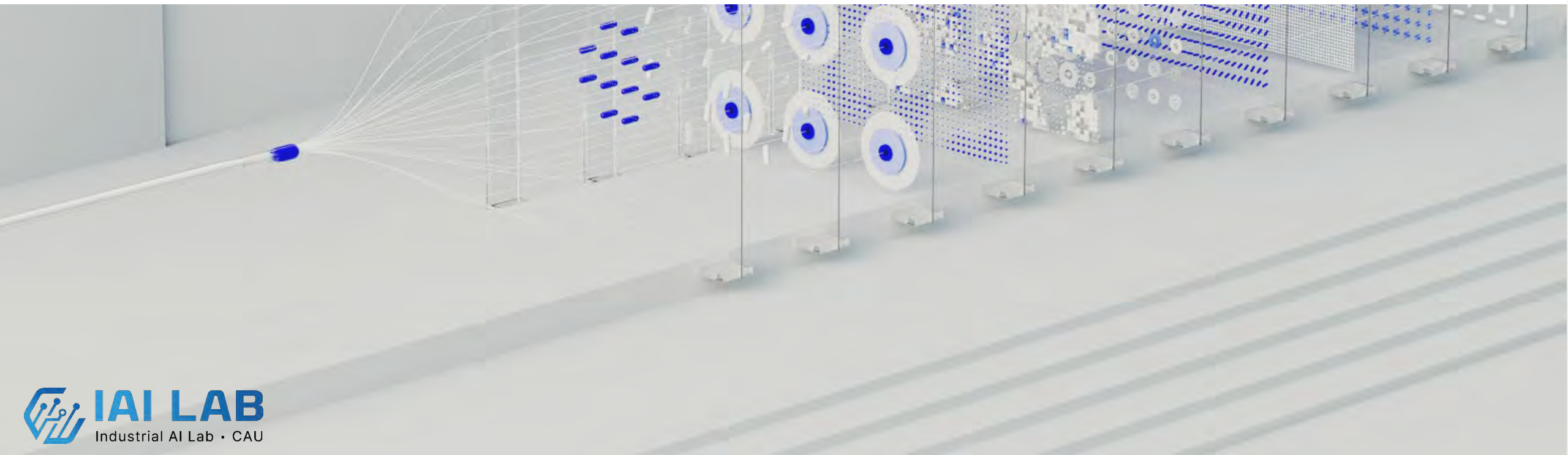


Generative Models: (3) Diffusion Model

- Physical intuition of diffusion models



Diffusion Model



Diffusion Models Beat GANs

Diffusion Models Beat GANs on Image Synthesis

Prafulla Dhariwal* OpenAI
prafulla@openai.com

Alex Nichol* OpenAI
alex@openai.com

Abstract

We show that diffusion models can achieve image sample quality superior to the current state-of-the-art generative models. We achieve this on unconditional image synthesis by finding a better architecture through a series of ablations. For conditional image synthesis, we further improve sample quality with classifier guidance: a simple, compute-efficient method for trading off diversity for fidelity using gradients from a classifier. We achieve an FID of 2.97 on ImageNet 128×128, 4.59 on ImageNet 256×256, and 7.72 on ImageNet 512×512, and we match BigGAN-deep even with as few as 25 forward passes per sample, all while maintaining better coverage of the distribution. Finally, we find that classifier guidance combines well with upsampling diffusion models, further improving FID to 3.94 on ImageNet 256×256 and 3.85 on ImageNet 512×512.

1 Introduction

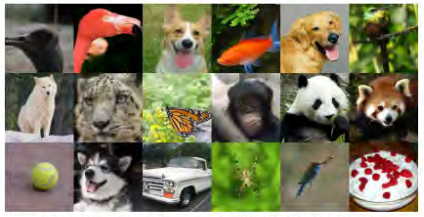


Figure 1: Selected samples from our best ImageNet 512×512 model (FID 3.85)

Over the past few years, generative models have gained the ability to generate human-like natural language [9], high-quality synthetic images [8, 15, 62] and highly diverse human speech and music [64, 12]. These models can be used in a variety of ways, such as generating images from text prompts [78, 56] or learning useful feature representations [18, 10]. While these models are already capable

*Equal contribution
35th Conference on Neural Information Processing Systems (NeurIPS 2021).

Diffusion Models Beat GANs on Topology Optimization

François Mazé¹, Faez Ahmed¹
¹Massachusetts Institute of Technology
francois.maze@mit.edu, {fmaze, faez}@mit.edu

Abstract

Structural topology optimization, which aims to find the optimal physical structure that maximizes mechanical performance, is vital in engineering design applications in aerospace, mechanical, and civil engineering. Recently, generative adversarial networks (GANs) have emerged as a popular alternative to traditional iterative topology optimization methods. However, GANs can be challenging to train, have limited generalizability, and often neglect important performance objectives such as mechanical compliance and manufacturability. To address these issues, we propose a new architecture called TopoDiff that uses conditional diffusion models to perform performance-aware and manufacturability-aware topology optimization. Our method introduces a surrogate model-based guidance strategy that actively favors structures with low compliance and directly generates images of optimized structures. Although popular, most of these models optimize a loss function that does not align with the primary goals of topology optimization — getting high-performance and feasible structures. They often train for loss functions related to image reconstruction to achieve visual similarity and ignore modeling the physical performance of the generated structures. Most of them produce disconnected, floating material that seriously affects the manufacturability of the generated design. They also suffer from limited generalizability, especially for out-of-distribution boundary conditions.

1 Introduction

Structural topology optimization (TO) of solid structures involves generating the optimal shape of a material by minimizing an objective function, for instance, mechanical compliance, within a given domain and under a given set of constraints (volume fraction, boundary conditions, and loads). TO is therefore becoming an essential design tool and is now included in most professional design software, such as Autodesk's Fusion 360 and SolidWorks. It is the driving force behind Autodesk's generative design toolset, where designers input design goals into the software, along with parameters such as performance or spatial requirements, materials, manufacturing methods, and cost constraints. The software quickly generates design alternatives. Most methods to solve TO rely on gradient-based approaches, the most common method being the Solid Isotropic Material with Penalization method (Bendsoe and Kikuchi 1988; Roerup, Zhou, Copyright © 2023, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

and Birker 1992). Despite their wide adoption, these traditional methods have two major pitfalls: their iterative nature makes them computationally expensive and they may generate non-optimal designs, for example, when penalization and filtering augmentations are used to avoid grayscale pixels in SIMP (Sigmund and Maute 2013). Several deep learning methods have been developed in recent years to improve and speed up the TO process (Yu et al. 2018; Sharpe and Seepersad 2019; Nie et al. 2021; Belzadi and Ilies 2021) by learning from large datasets of optimized structures. The latest and most promising results were obtained with deep generative models (DGMs) and notably with conditional generative adversarial networks (cGANs) trained for image synthesis, which take as input the boundary conditions and directly generate images of optimized structures. Although popular, most of these models optimize a loss function that does not align with the primary goals of topology optimization — getting high-performance and feasible structures. They often train for loss functions related to image reconstruction to achieve visual similarity and ignore modeling the physical performance of the generated structures. Most of them produce disconnected, floating material that seriously affects the manufacturability of the generated design. They also suffer from limited generalizability, especially for out-of-distribution boundary conditions. We hypothesize that the absence of explicit methods to generate designs with low compliance and good manufacturability causes these issues. We further hypothesize that the reliance of the optimization objective on the sole cGAN prompts the model to only mimic pixel-wise the ground truth produced by traditional TO methods. As a result, two images with comparable pixel-wise similarity may still have significantly different performance values. The absence of explicit external guidance is even more problematic since the ground truth data is not guaranteed to be optimal, as explained above. This paper introduces TopoDiff, a conditional diffusion-model-based method for TO. Dhariwal and Nichol (2021) have shown that diffusion models can outperform GANs for image generation, are easier to train, and are thus more readily adaptable to other tasks. We show that by introducing performance and constraints to diffusion models, they also outperform GANs on topology optimization problems. In addition, the sequential nature of diffusion models makes

Diffused Heads: Diffusion Models Beat GANs on Talking-Face Generation

Michał Stypułkowski¹ **Konstantinos Vougioukas²** **Sen He**
michal.stypulkowski@cs.uni.wroc.pl k.vougioukas@imperial.ac.uk senhe752@gmail.com

Maciej Zięba^{1,4} **Stavros Petridis²** **Maja Pantić²**
maciej.zieba@pwr.edu.pl sp104@imperial.ac.uk m.pantic@imperial.ac.uk

¹University of Wrocław ²Imperial College London
³Wrocław University of Science and Technology ⁴Tooploox

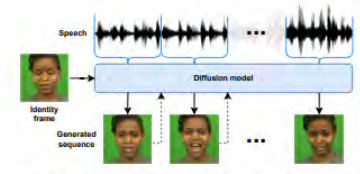


Figure 1: Overview of the proposed approach. Given a single identity frame and an audio clip containing speech, the model uses a diffusion model to sample consecutive frames in an autoregressive manner, preserving the identity, and modeling lip and head movement to match the audio input. Contrary to other methods, no additional guidance is required.

Abstract

Talking face generation has historically struggled to produce head movements and natural facial expressions without guidance from additional reference videos. Recent developments in diffusion-based generative models allow for more realistic and stable data synthesis and their performance on image and video generation has surpassed that of other generative models. In this work, we present an autoregressive diffusion model that requires only one identity image and audio sequence to generate a video of a realistic talking head. Our solution is capable of hallucinating head movements, facial expressions, such as blinks, and preserving a given background. We evaluate our model on two different datasets, achieving state-of-the-art results in expressiveness and smoothness on both of them.¹

¹Project page: <https://stypulkowski.github.io/diffusedheads/>.

1. Introduction

Animation of faces from speech can have a broad scope of applications from an alternative to video compression during virtual calls with poor connectivity, to artistic animation for entertainment industry applications, e.g., movies, video games, and VR experience. Up to date, existing methods struggle to create naturally-looking faces that maintain genuine expressions and movements, while still requiring additional supervision during the generation process. Deep generative models are constantly gaining popularity and achieving impressive results in image and video generation tasks and have become the de facto standard for most facial animation systems, which are a simple and effective way of producing character animations, have been revolutionized by the introduction of recent generative models such as Generative Adversarial Networks (GANs) [14]. GANs are known for being able to produce high-quality frames while simultaneously giving a large degree of control over

Diffusion models beat GANs
on image synthesis

Dhariwal & Nichol, NeurIPS, 2021.

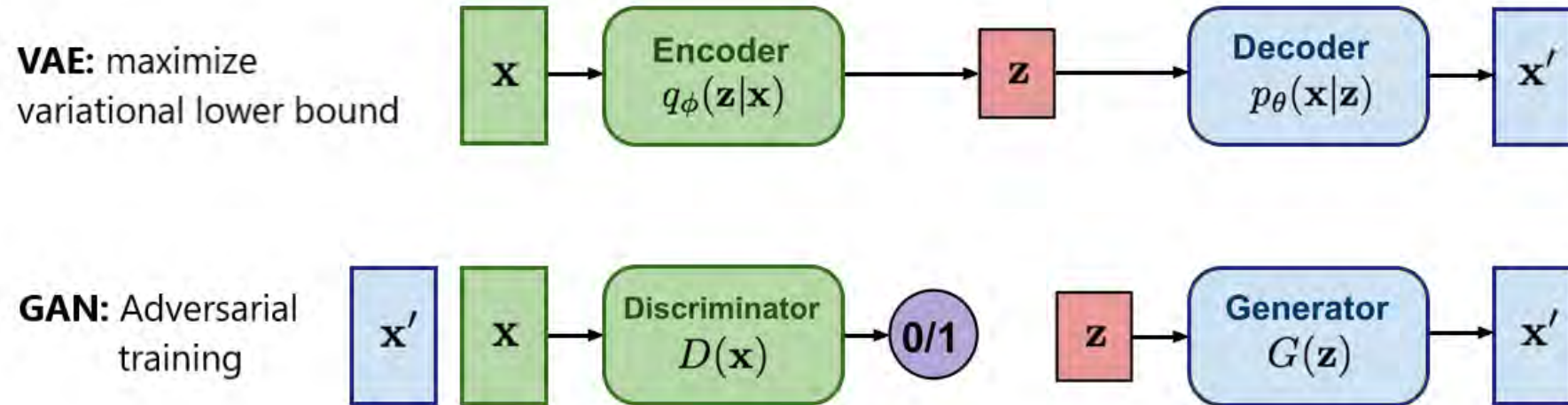
Diffusion models beat GANs
on topology optimization

Maze & Ahmed, AAAI, 2023.

Diffusion models beat GANs
on talking-face generation

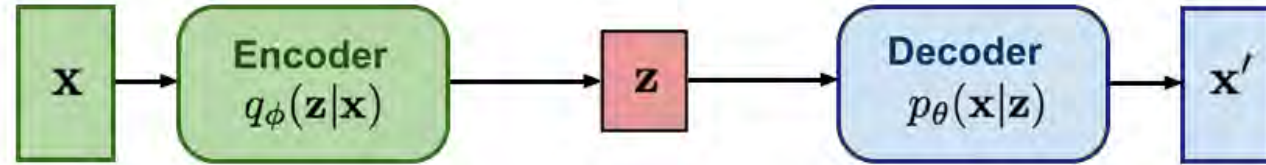
Stypułkowski et al., WACV, 2024.

Revisit: VAE/GAN Generation Process

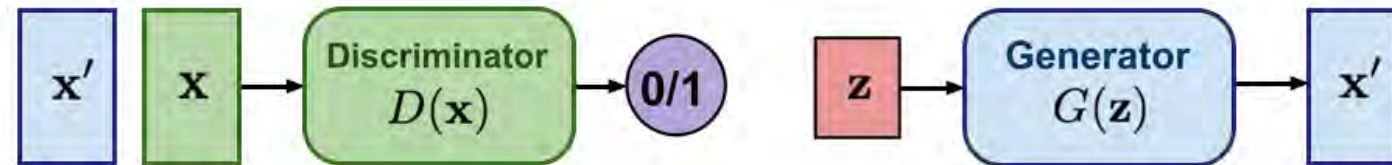


Generation Process of Diffusion Models

VAE: maximize
variational lower bound



GAN: Adversarial
training

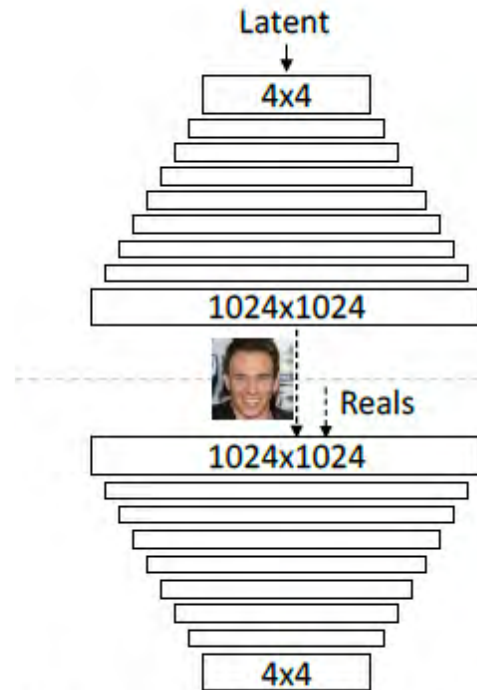


Diffusion models:
Gradually add Gaussian
noise and then reverse



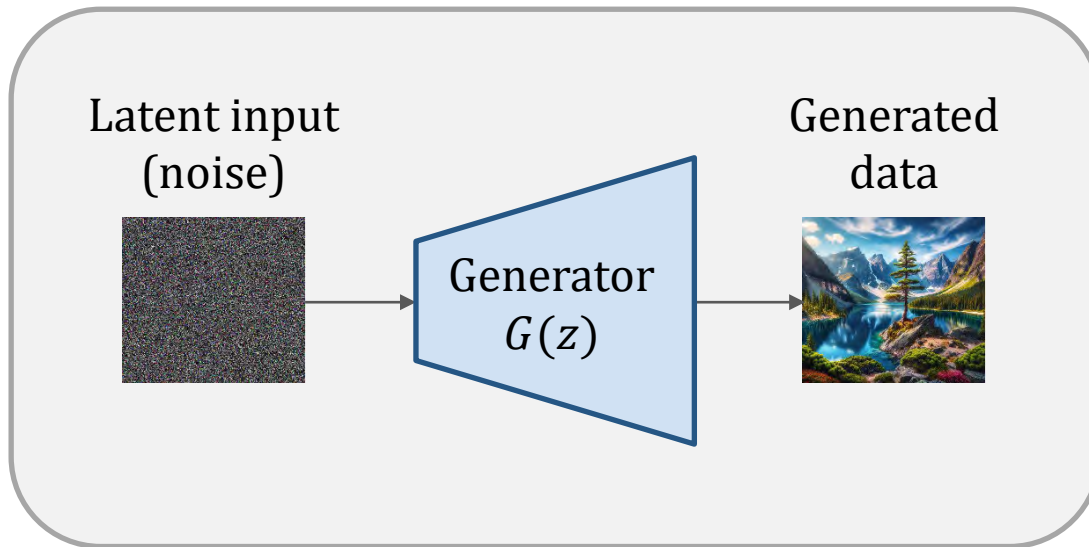
Intuition from Progressive Growing of GAN (PGGAN)

- To generate high-quality images and improve training stability through progressive growing of GANs
- A latent vector is transformed through layers of the generator

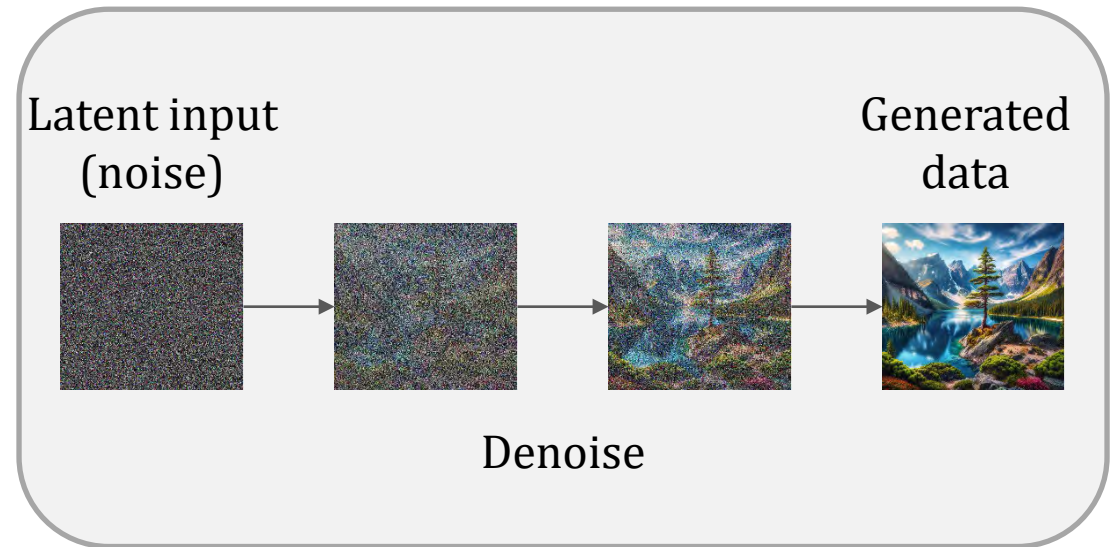


Principles of Data Generation in Diffusion Models

- Generate new data gradually from latent input
 - Improved stability [1], [2]
 - Better sampling [1]



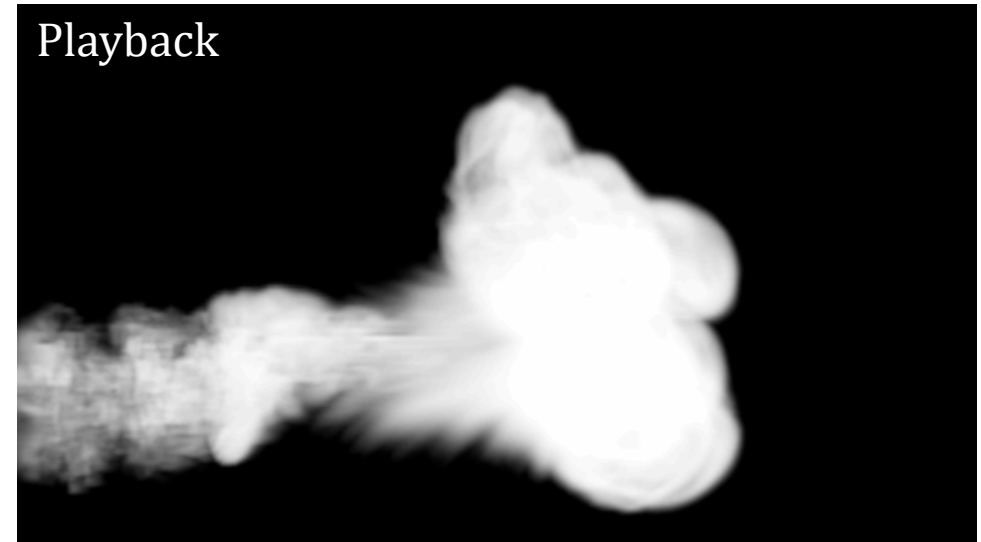
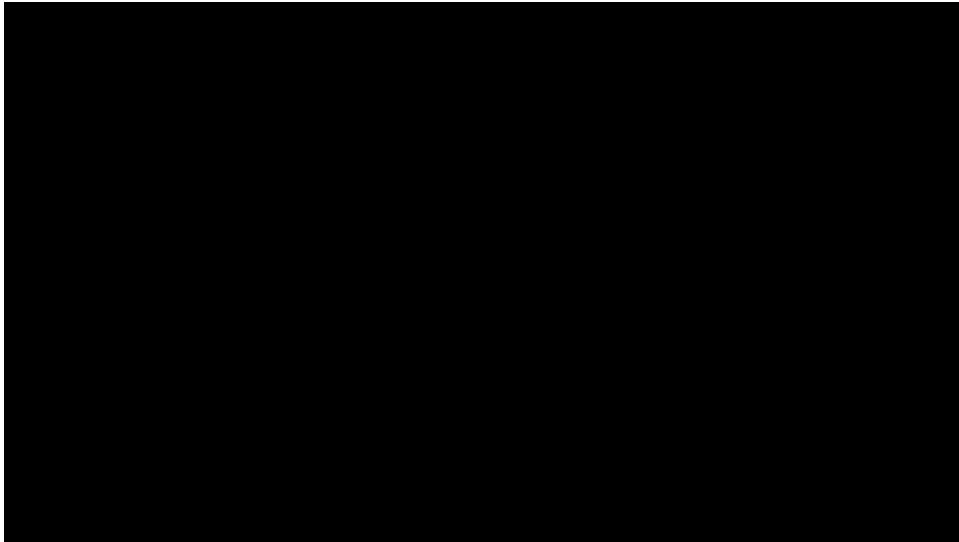
Data generation with GAN/VAE



Data generation with diffusion models

Physical Intuition of Diffusion Models

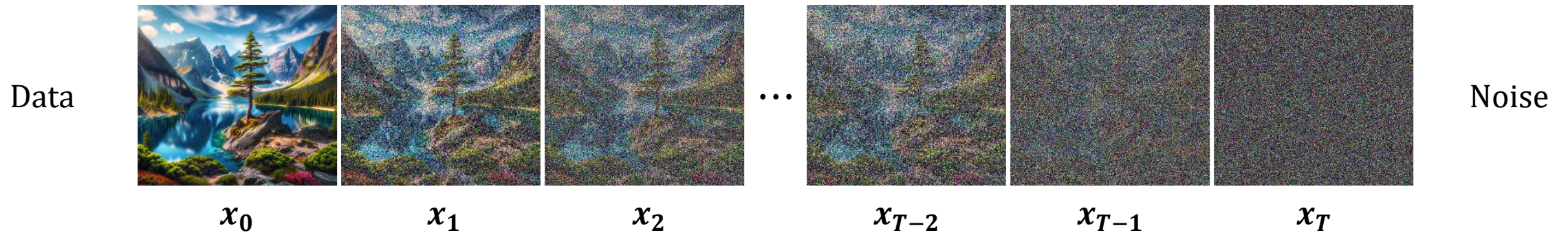
- Formulated as Markov chains
 - Gaussian transition for a sufficiently small timestep



1. Markovian property
2. Gaussian transition

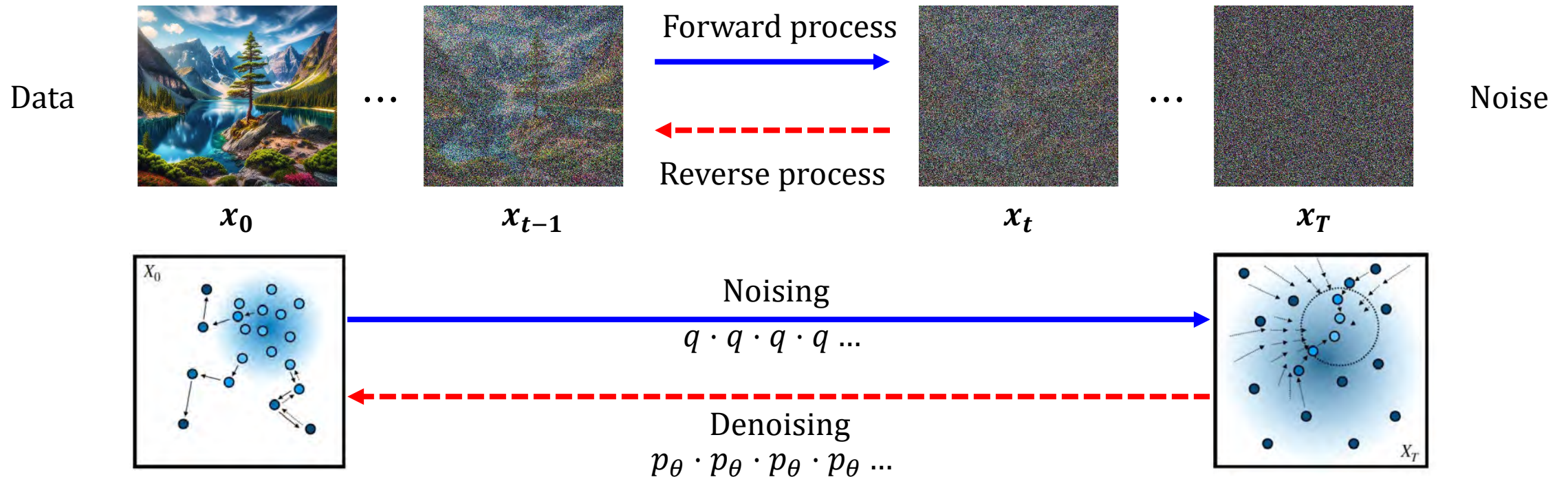
Diffusion Models Overview

- Learn to generate by denoising
- Consists of two processes
 - Forward process: gradually adds noise to input to destroy information
 - Reverse process: iteratively removes noise to generate sample



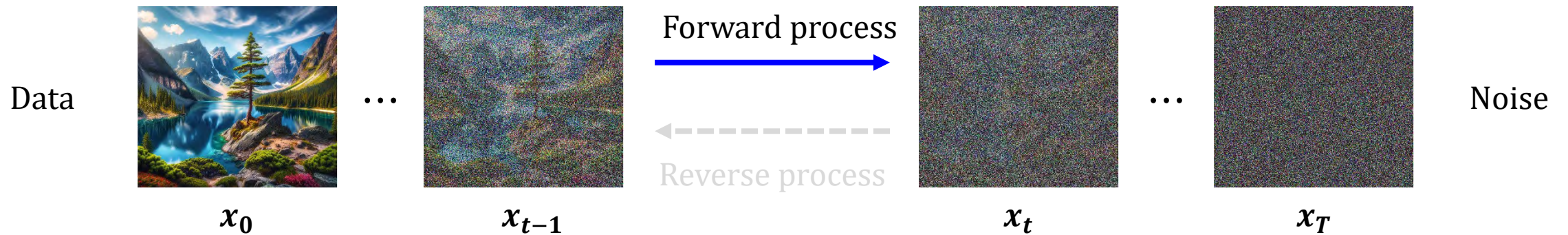
Diffusion Models Overview

- Learn to generate by denoising
- Consists of two processes
 - Forward process: gradually adds noise to input to destroy information
 - Reverse process: iteratively removes noise to generate sample



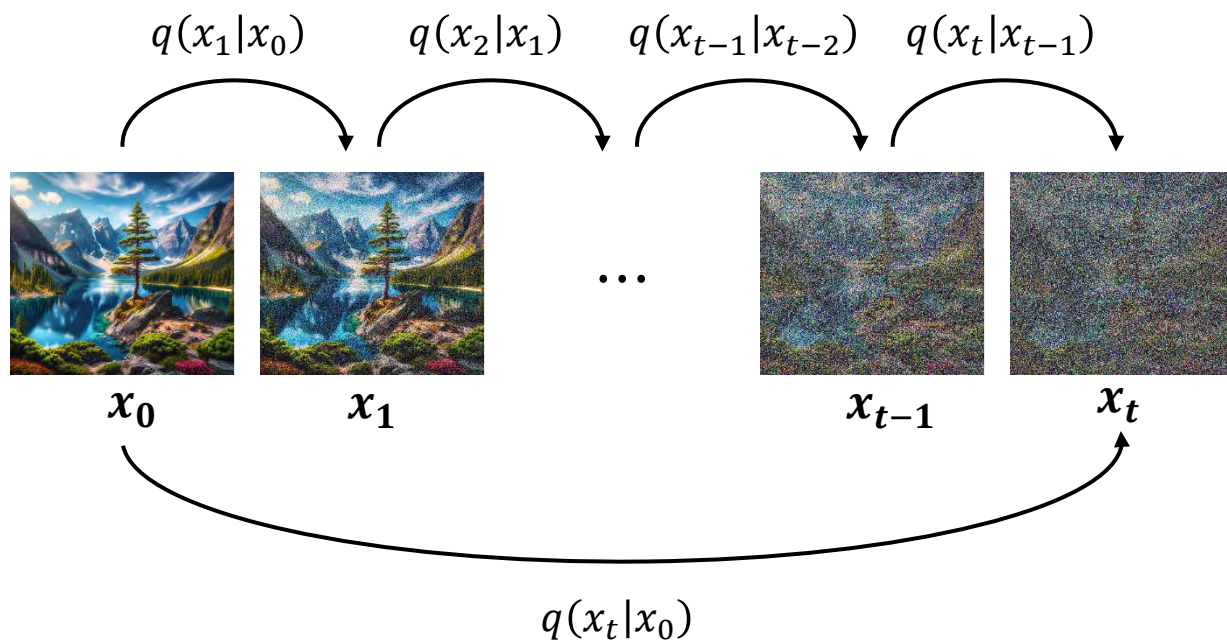
Diffusion Models Overview

- Learn to generate by denoising
- Consists of two processes
 - Forward process: gradually adds noise to input to destroy information
 - Reverse process: iteratively removes noise to generate sample



Forward Diffusion Process

- Tractable closed-form sampling for any t
 - Sample x_t using reparameterization trick



$$q(x_t|x_0) = q(x_t|x_{t-1})q(x_{t-1}|x_{t-2}) \dots q(x_1|x_0)$$

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

Reparameterization trick

$\mathcal{N}(\mu, \sigma^2) \rightarrow \mu + \sigma * \epsilon$, where $\epsilon \sim \mathcal{N}(0,1)$

$$x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1 - \bar{\alpha}_t}\epsilon$$

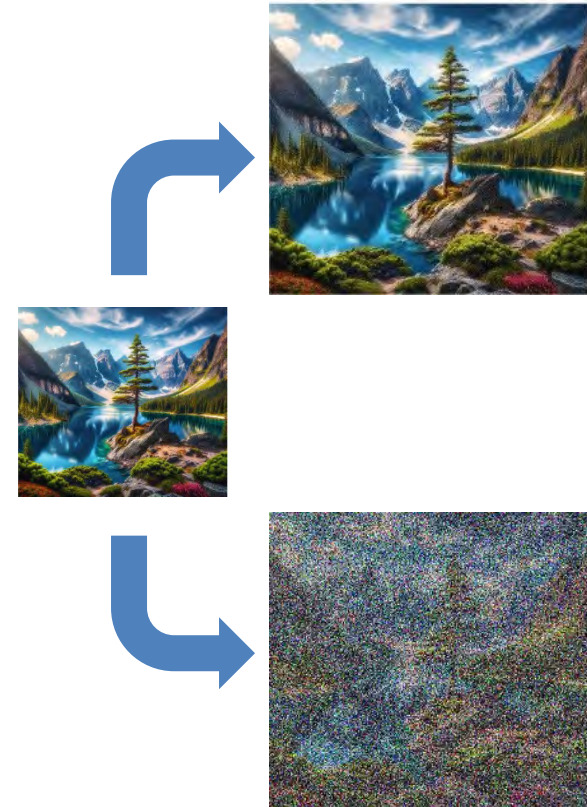
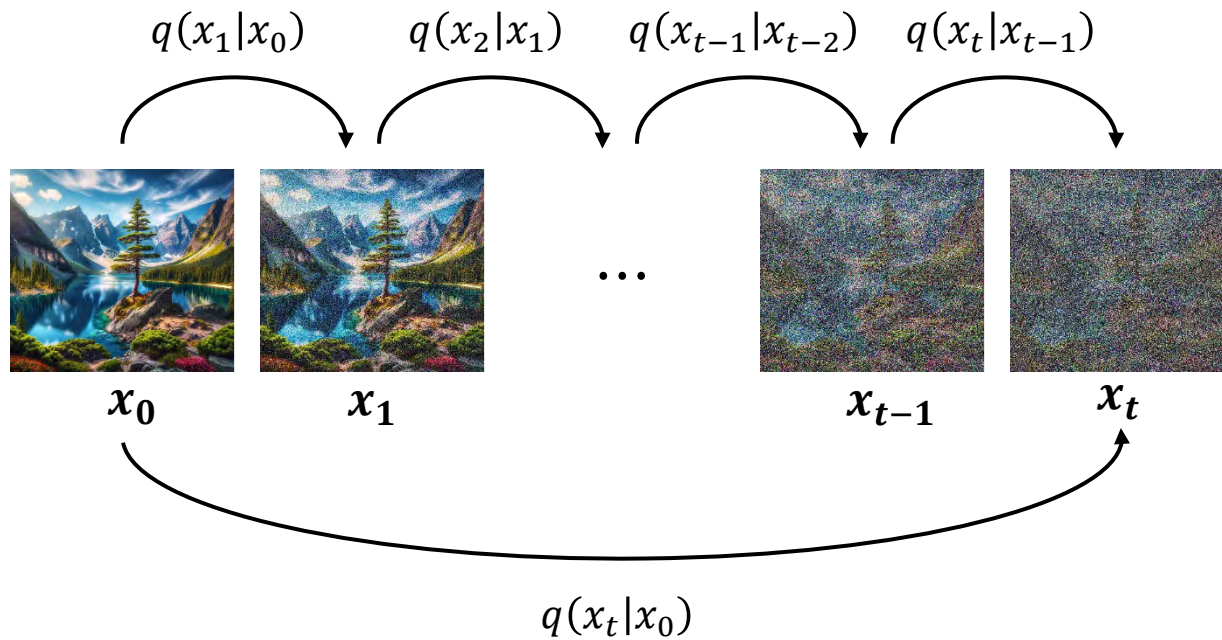
$$\alpha_t : 1 - \beta_t$$

$$\bar{\alpha}_t : \prod_{i=0}^t \alpha_i$$

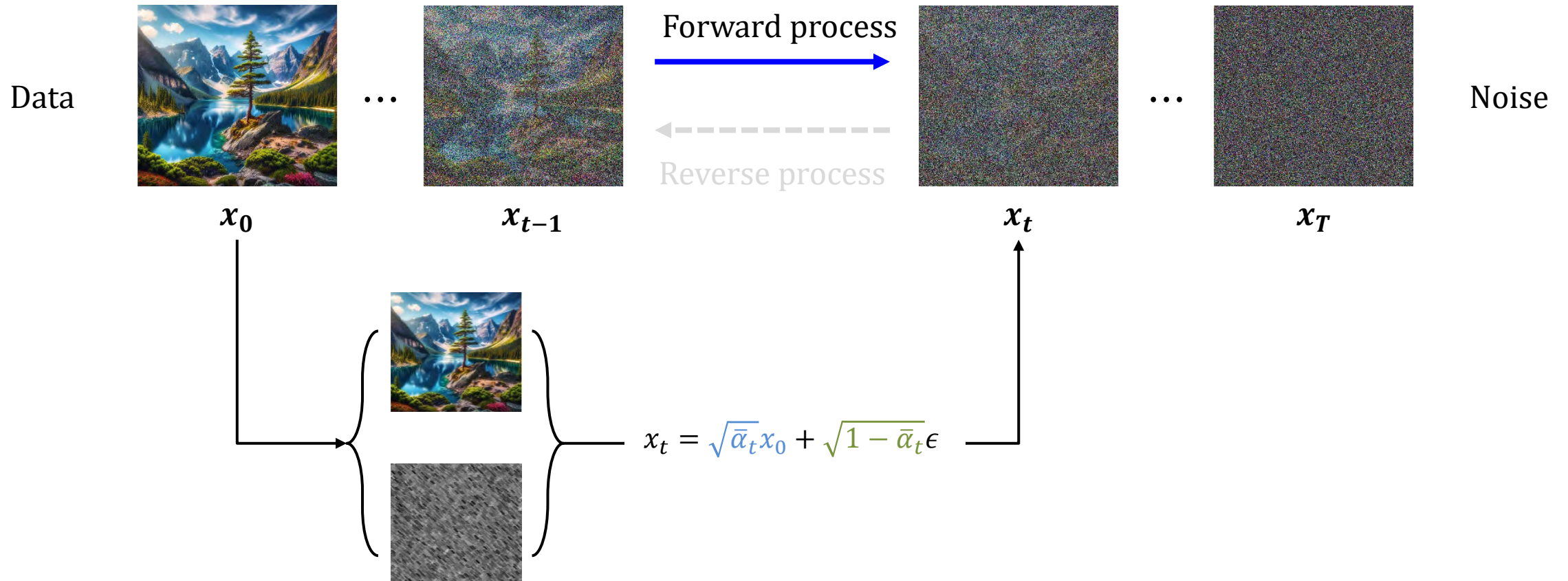
$$\epsilon : \text{Noise } (\epsilon \sim \mathcal{N}(0,1))$$

Forward Diffusion Process

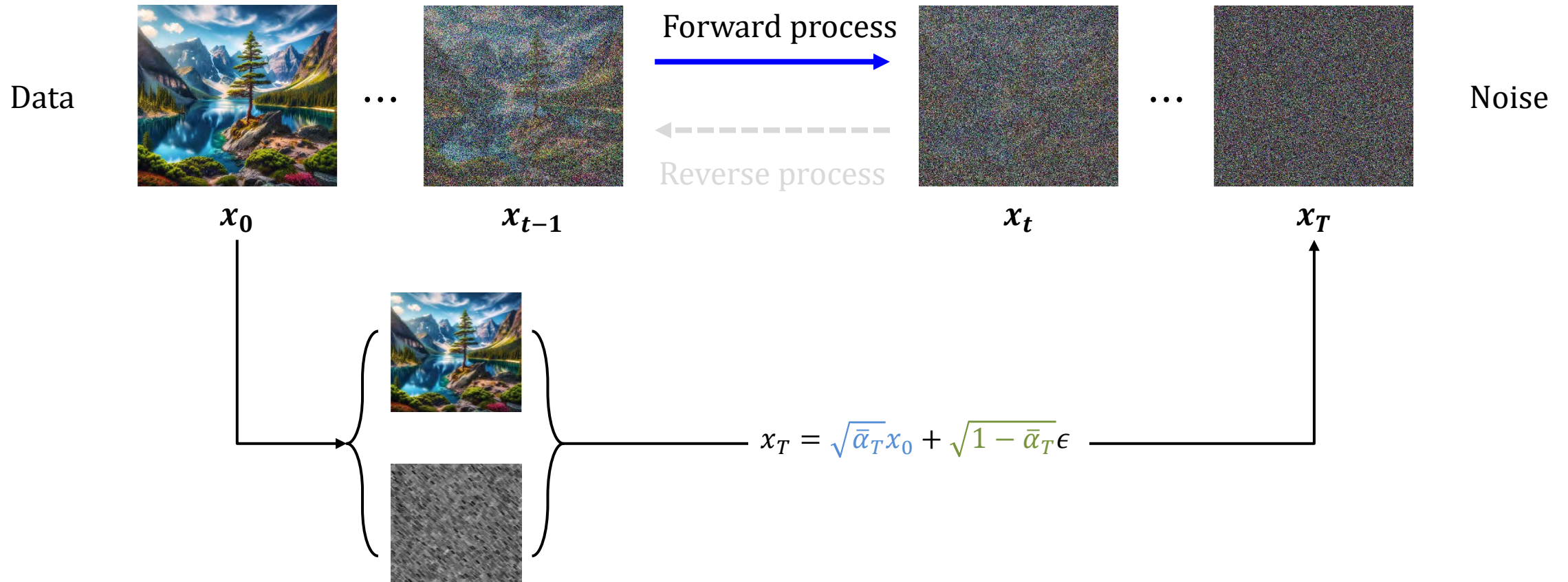
- Tractable closed-form sampling for any t
 - Sample x_t using reparameterization trick



Forward Diffusion Process

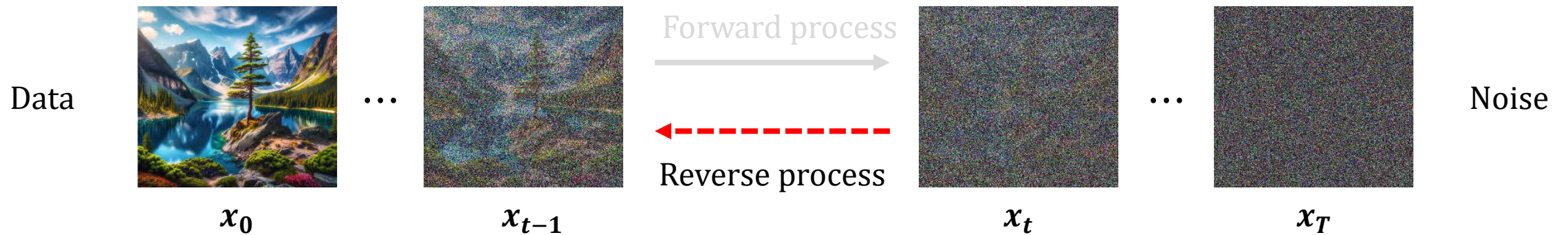


Forward Diffusion Process

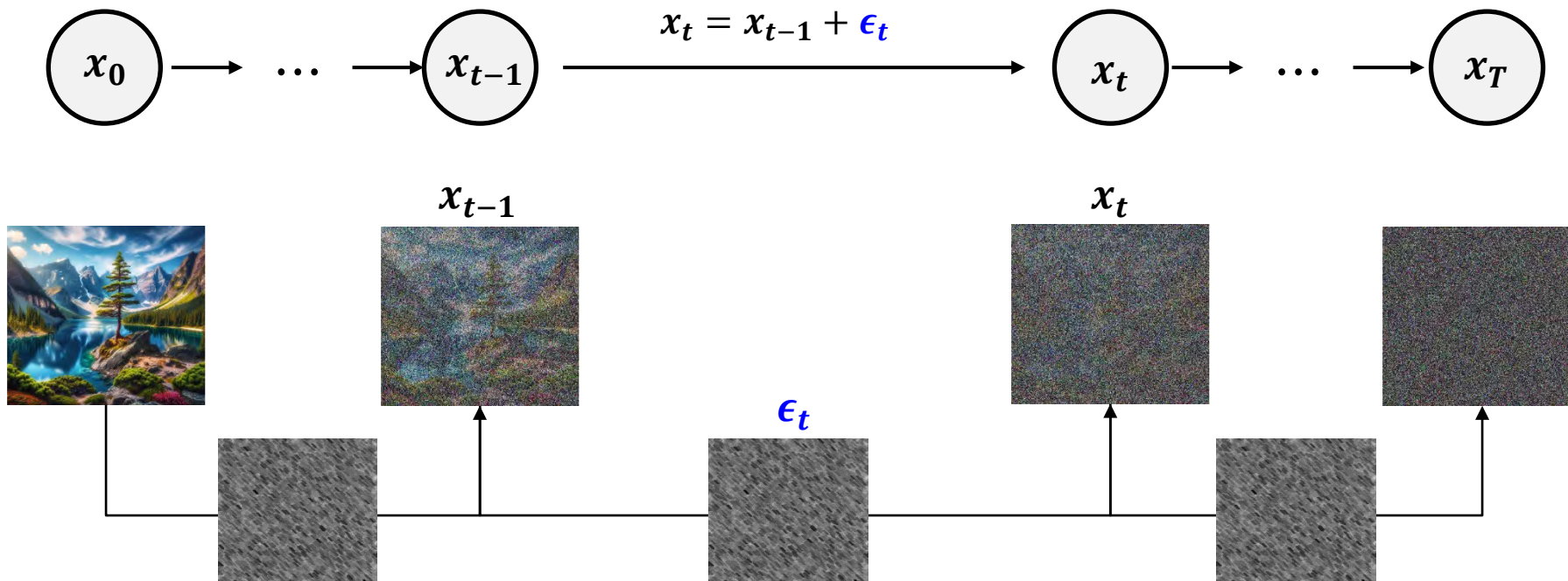


Diffusion Models Overview

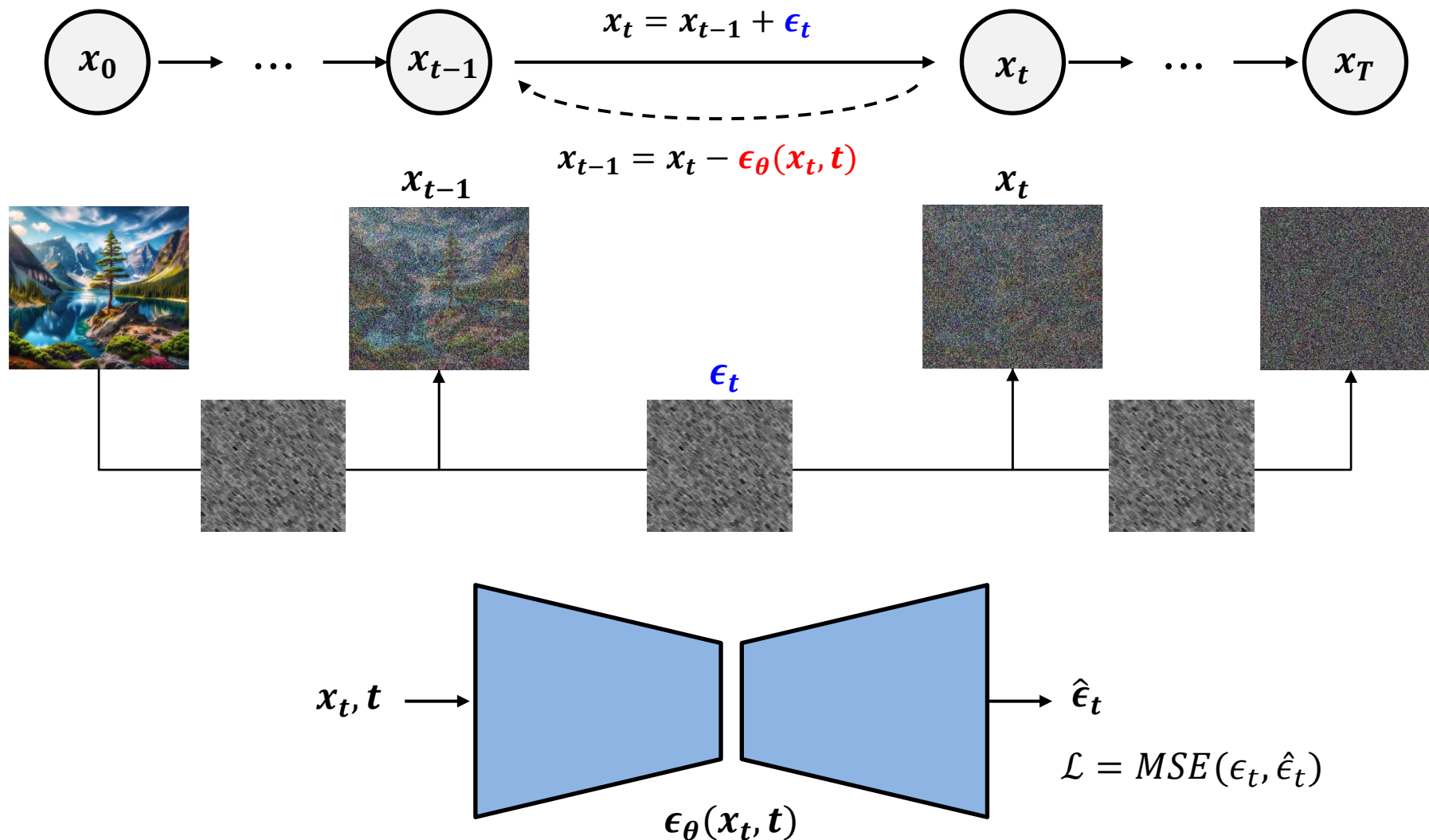
- Learn to generate by denoising
- Consists of two processes
 - Forward process: gradually adds noise to input to destroy information
 - Reverse process: iteratively removes noise to generate sample



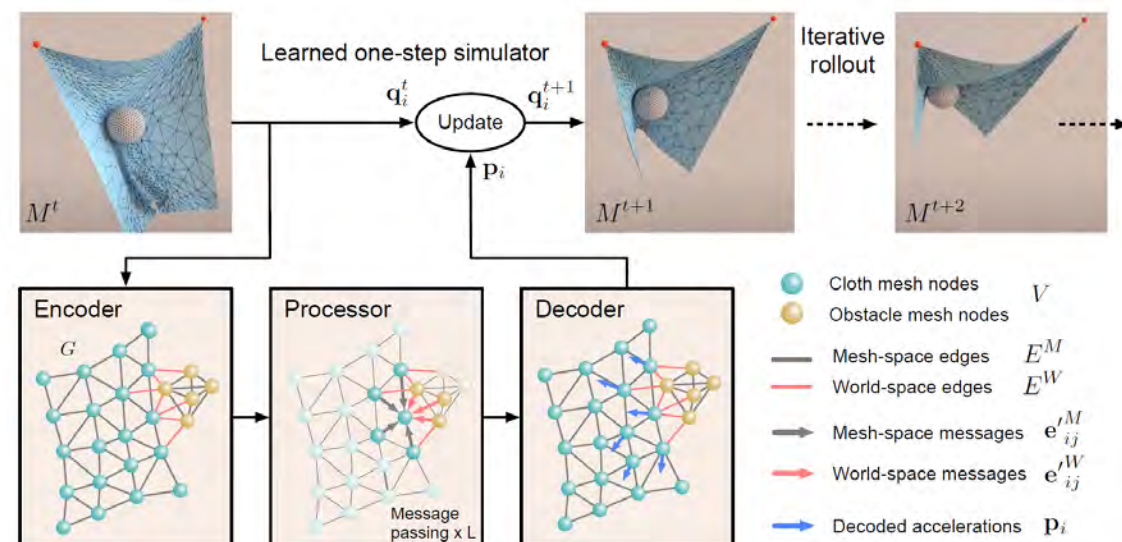
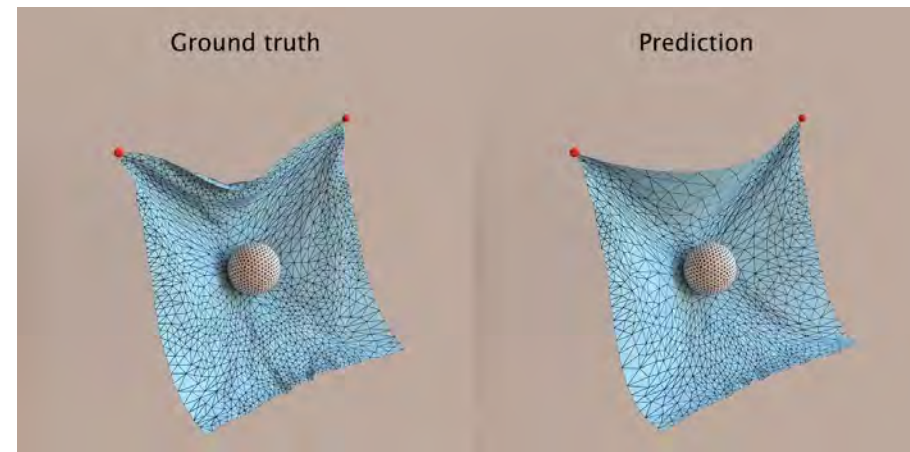
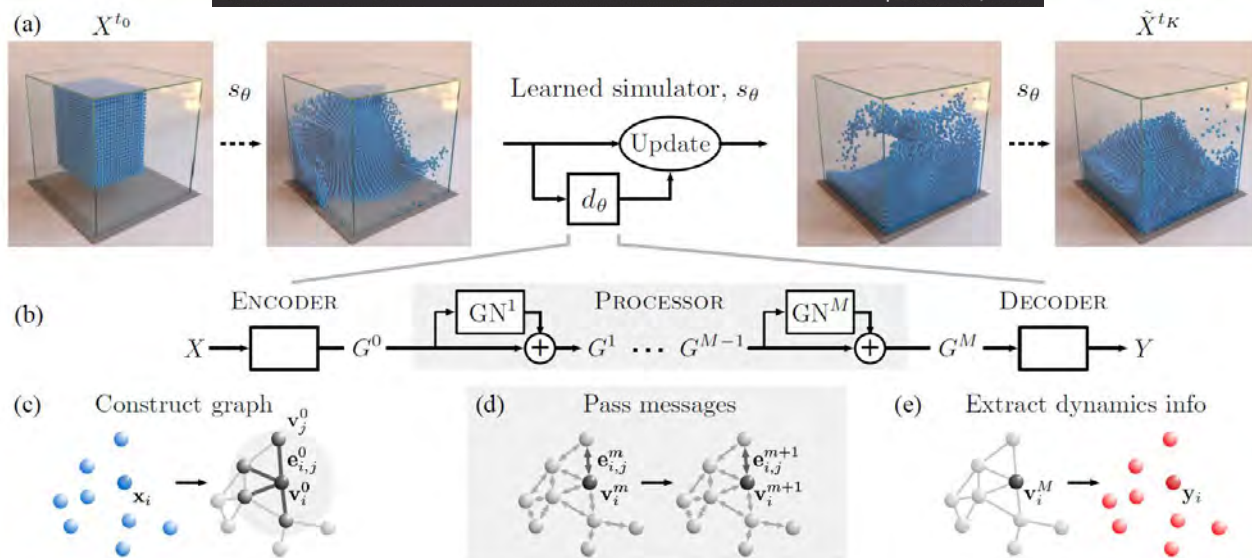
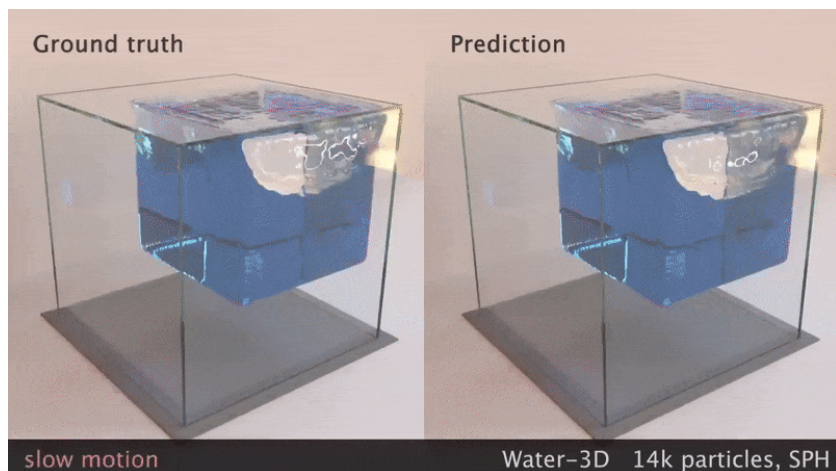
Reverse Diffusion Process (Overall)



Reverse Diffusion Process (Overall)

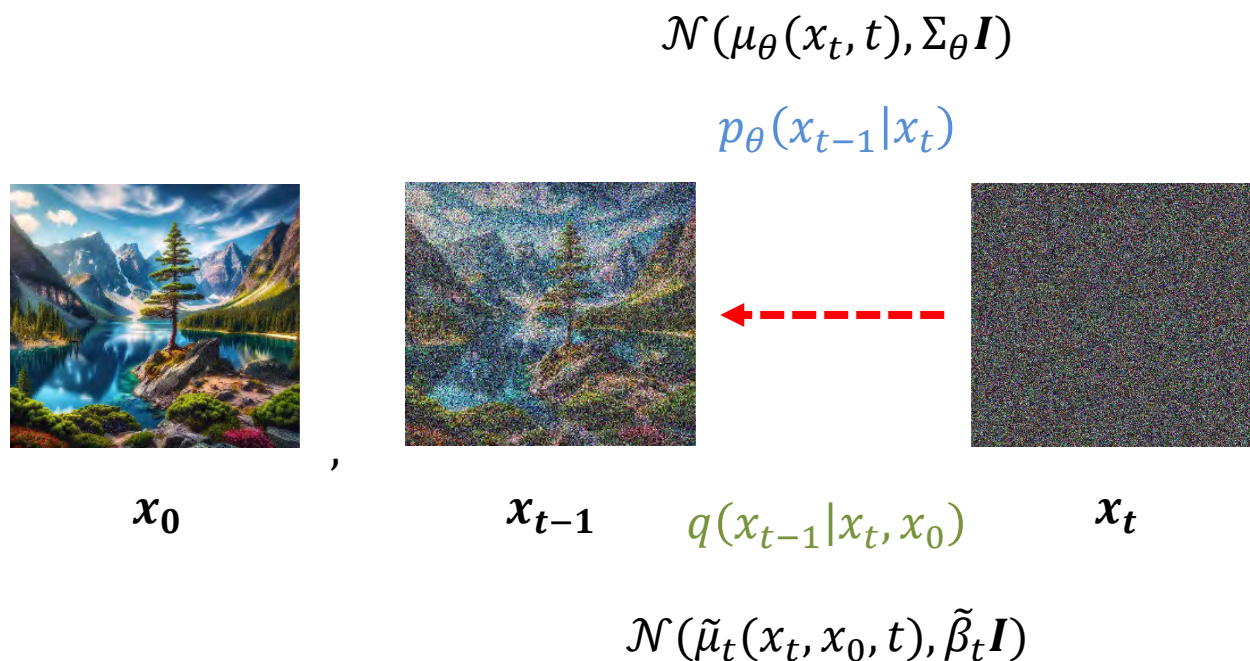


Learning Discrepancies



Reverse Diffusion Process (Detailed)

- Objective is to learn reverse transition $q(x_{t-1}|x_t)$ with neural network $p_\theta(x_{t-1}|x_t)$
 - Condition $q(x_{t-1}|x_t)$ on x_0 , i.e., use $q(x_{t-1}|x_t, x_0)$ during training



$$\mathcal{L} = D_{KL}(q(x_{t-1}|x_t, x_0) || p_\theta(x_{t-1}|x_t))$$

$$\tilde{\mu}_t(x_t, x_0, t) \quad \mu_\theta(x_t, t)$$

$$\tilde{\beta}_t I \quad \Sigma_\theta I$$

Ho et al. NeurIPS 2020

$$\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}} \left(x_0 - \frac{1 - \alpha_t}{\sqrt{1 - \alpha_t}} \epsilon_\theta(x_t, t) \right)$$

Thus $x_{t-1} = \mathcal{N}(x_t, \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{1 - \alpha_t}{\sqrt{1 - \alpha_t}} \epsilon_\theta(x_t, t) \right), \Sigma_\theta(x_t, t))$

The loss term L_t is parameterized to minimize the difference from $\tilde{\mu}$:

$$L_t = \mathbb{E}_{x_0, \epsilon} \left[\frac{1}{2 \|\Sigma_\theta(x_t, t)\|_F^2} \|\tilde{\mu}_t(x_t, x_0) - \mu_\theta(x_t, t)\|_2^2 \right]$$

$$= \mathbb{E}_{x_0, \epsilon} \left[\frac{1}{2 \|\Sigma_\theta\|_F^2} \left\| \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{1 - \alpha_t}{\sqrt{1 - \alpha_t}} \epsilon_t \right) - \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{1 - \alpha_t}{\sqrt{1 - \alpha_t}} \epsilon_\theta(x_t, t) \right) \right\|_2^2 \right]$$

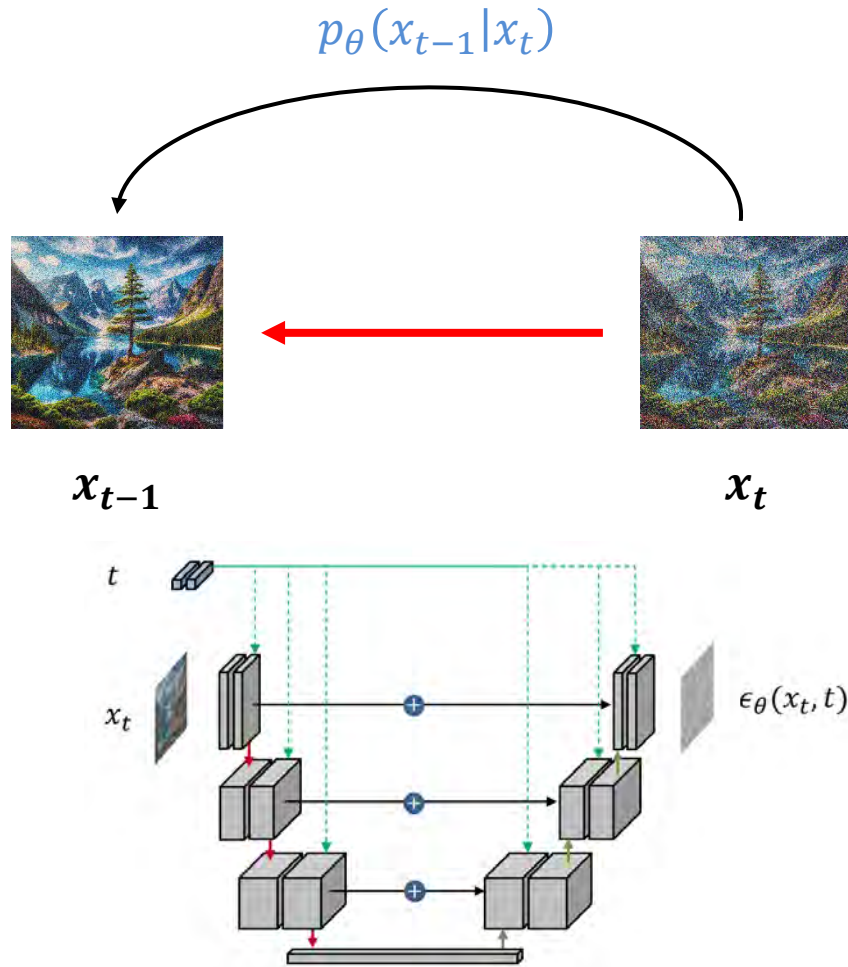
$$= \mathbb{E}_{x_0, \epsilon} \left[\frac{(1 - \alpha_t)^2}{2 \alpha_t (1 - \alpha_t) \|\Sigma_\theta\|_F^2} \|\epsilon_t - \epsilon_\theta(x_t, t)\|_2^2 \right]$$

$$= \mathbb{E}_{x_0, \epsilon} \left[\frac{(1 - \alpha_t)^2}{2 \alpha_t (1 - \alpha_t) \|\Sigma_\theta\|_F^2} \|\epsilon_t - \epsilon_\theta(\sqrt{\alpha_t} x_0 + \sqrt{1 - \alpha_t} \epsilon_t, t)\|_2^2 \right]$$

$$\mathcal{L} = \text{MSE}(\epsilon_t - \epsilon_\theta(x_t, t))$$

Predict noise to be removed

Training of Diffusion Model



$$\mathcal{L} = \text{MSE}(\epsilon_t - \epsilon_{\theta}(x_t, t))$$

$$x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1 - \bar{\alpha}_t}\epsilon$$

$$= \text{MSE}(\epsilon_t - \epsilon_{\theta}(\sqrt{\bar{\alpha}_t}x_0 + \sqrt{1 - \bar{\alpha}_t}\epsilon_t, t))$$

Algorithm 1 Training

- 1: **repeat**
- 2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
- 3: $t \sim \text{Uniform}(\{1, \dots, T\})$
- 4: $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
- 5: Take gradient descent step on

$$\nabla_{\theta} \|\epsilon - \epsilon_{\theta}(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t}\epsilon, t)\|^2$$
- 6: **until** converged

Data Generation (Sampling)

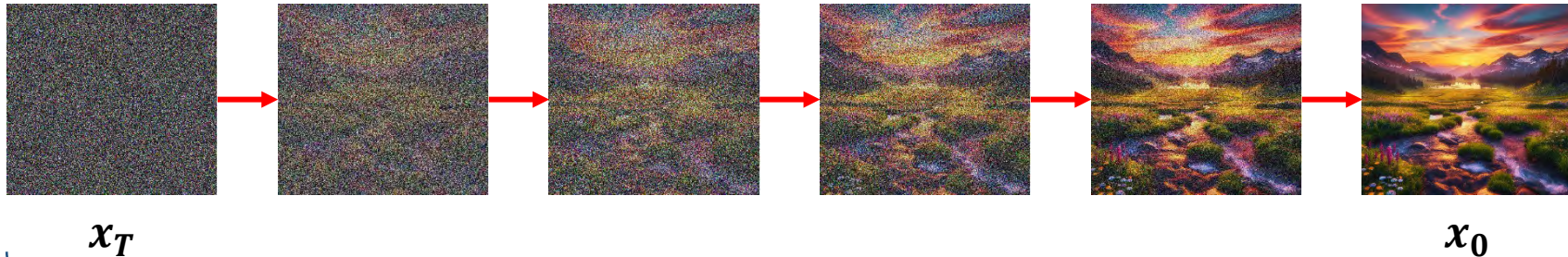
- Noise is iteratively removed from image
 - Repeat for T steps

$$\mathcal{N}(x_{t-1}; \mu_{\theta}(x_t, t), \Sigma_{\theta} \mathbf{I})$$

Reparameterization trick

$$x_{t-1} = \frac{1}{\sqrt{\bar{\alpha}_t}} \left(x_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \epsilon_{\theta}(x_t, t) \right) + \sigma_t \mathbf{z}$$

Noise
(Latent input)

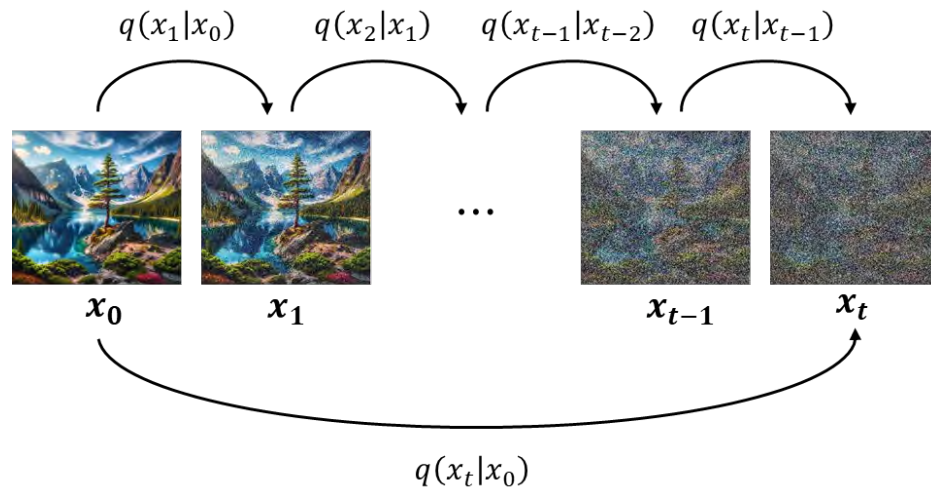


Data

Lab: Diffusion Model

Forward Process Implementation

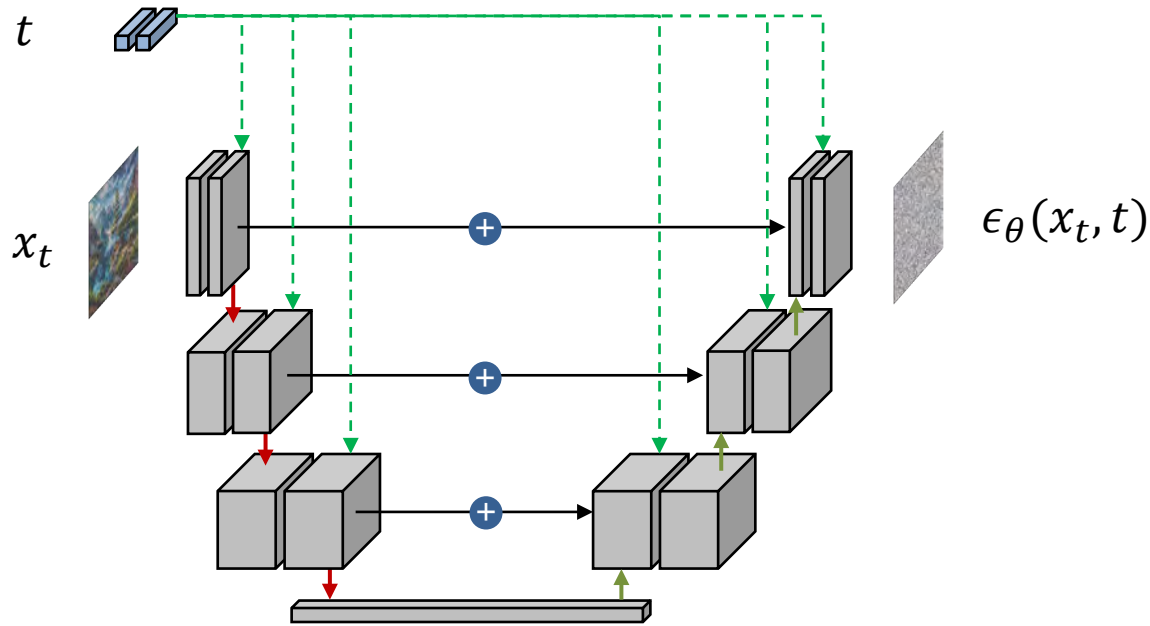
$$x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1 - \bar{\alpha}_t}\epsilon$$



```
def forward_diffusion(x0, t):  
    beta_schedule = torch.linspace(0, 1.0, T+1)  
    alphas = 1. - beta_schedule  
    alpha_bars = torch.cumprod(alphas, dim=0)  
  
    epsilon = torch.randn_like(x0)  
  
    alpha_bar_t = alpha_bars[t]  
    alpha_bar_t = alpha_bar_t.view(-1, 1, 1, 1)  
    noisy_image = torch.sqrt(alpha_bar_t) * x0 + torch.sqrt(1 - alpha_bar_t) * epsilon  
  
    return noisy_image, epsilon
```


Neural Network Architecture

- U-Net with time embedding
 - Neural network $\epsilon_{\theta}(x_t, t)$ takes in two inputs! (x_t and t)



```
class UNet(nn.Module):
    def __init__(self):
        super(UNet, self).__init__()
        self.time_embedding_16 = TimeEmbedding(16)
        self.time_embedding_32 = TimeEmbedding(32)
        self.time_embedding_64 = TimeEmbedding(64)
        self.max_pool = nn.MaxPool2d(2)
        self.flatten = nn.Flatten()
        self.dense1 = nn.Linear(1, 192)
        self.batch_norm1 = nn.BatchNorm1d(192)
        self.dense2 = nn.Linear(192, 7*7*64)
        self.batch_norm2 = nn.BatchNorm1d(7*7*64)
        self.conv_transpose1 = nn.ConvTranspose2d(64, 64, kernel_size=4, stride=2, padding=1)
        self.conv_transpose2 = nn.ConvTranspose2d(64, 32, kernel_size=4, stride=2, padding=1)
        self.conv_final = nn.Conv2d(32, 1, kernel_size=1, padding=0)

    def forward(self, x_img, x_ts):
        x_ts_out = F.relu(self.batch_norm1(self.dense1(x_ts)))
        x28_down = self.time_embedding_16(x_img, x_ts_out)
        block1_down = self.max_pool(x28_down)

        x14_down = self.time_embedding_32(block1_down, x_ts_out)
        block2_down = self.max_pool(x14_down)

        x7_down = self.time_embedding_64(block2_down, x_ts_out)

        mlp_input = self.flatten(x7_down)
        mlp_concat = torch.cat([mlp_input, x_ts_out], dim=1)
        mlp_out = F.relu(self.batch_norm2(self.dense2(mlp_concat)))
        mlp_out = mlp_out.view(-1, 64, 7, 7)

        x7_up_input = torch.cat([mlp_out, x7_down], dim=1)
        x7_up = self.time_embedding_64(x7_up_input, x_ts_out)
        block1_up = self.conv_transpose1(x7_up)

        x14_up_input = torch.cat([block1_up, x14_down], dim=1)
        x14_up = self.time_embedding_32(x14_up_input, x_ts_out)
        block2_up = self.conv_transpose2(x14_up)

        x28_up_input = torch.cat([block2_up, x28_down], dim=1)
        x28_up = self.time_embedding_16(x28_up_input, x_ts_out)

        x_out = self.conv_final(x28_up)
        return x_out
```

Training

$$\mathcal{L} = \text{MSE}(\epsilon_t - \epsilon_{\theta}(x_t, t))$$

Algorithm 1 Training

- 1: **repeat**
 - 2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
 - 3: $t \sim \text{Uniform}(\{1, \dots, T\})$
 - 4: $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
 - 5: Take gradient descent step on
 $\nabla_{\theta} \|\epsilon - \epsilon_{\theta}(\sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon, t)\|^2$
 - 6: **until** converged
-

```
for e in range(n_epoch):
    bar = trange(50)
    total = 100
    for i in bar:
        for j in range(total):
            x_img_batch = next(iter(train_loader))[0].to(device)
            ts_batch = torch.randint(0, T, (len(x_img_batch), 1), device=device, dtype=torch.float32)
            x_t, epsilon = forward_diffusion(x_img_batch, ts_batch)

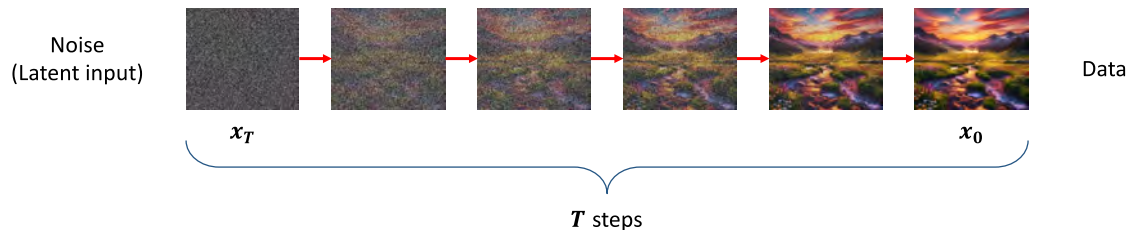
            optimizer.zero_grad()
            output = model(x_t, ts_batch)
            loss = criterion(output, epsilon)

            loss.backward()
            optimizer.step()

        pg = (j / total) * 100
        if j % 5 == 0:
            bar.set_description(f'loss: {loss.item():.5f}, p: {pg:.2f}%')
```

Reverse Process Implementation (After training)

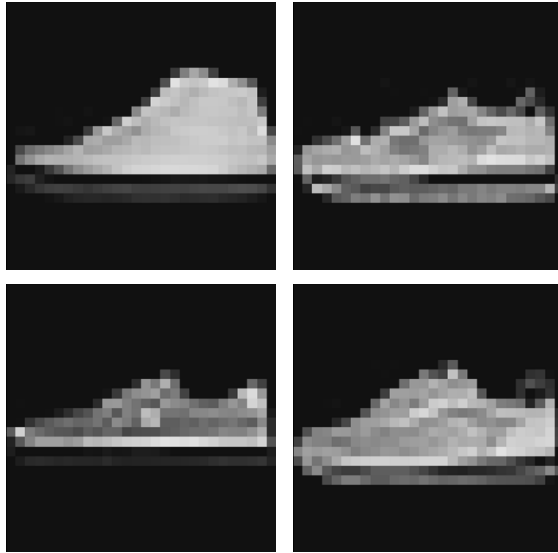
$$x_{t-1} = \frac{1}{\sqrt{\bar{\alpha}_t}} \left(x_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \epsilon_{\theta}(x_t, t) \right) + \beta_t z$$



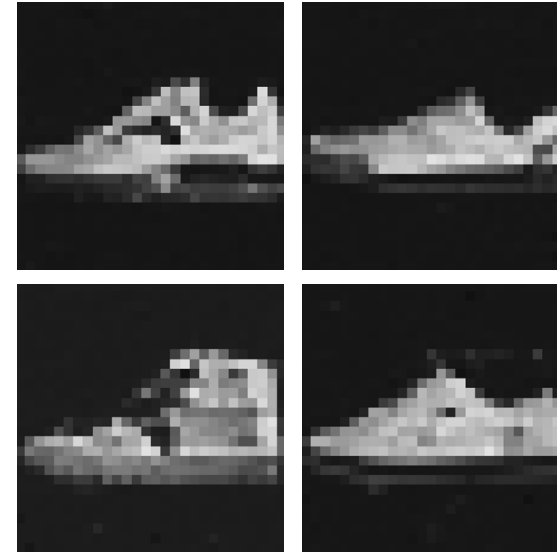
```
def reverse_diffusion(n_sample):  
    x_t = torch.randn(n_sample, 1, 28, 28, device=device)  
  
    alphas = 1. - beta_schedule  
    alpha_bars = torch.cumprod(alphas, dim=0).to(device)  
  
    for t in range(T-1):  
        t = T - t - 1  
  
        with torch.no_grad():  
            e_t = model(x_t, torch.full((n_sample, 1), t, device=device, dtype=torch.float32))  
            x_t_1 = (1 / torch.sqrt(alphas[t])) * (x_t - ((1 - alphas[t]) / torch.sqrt(1 - alpha_bars[t]) * e_t))  
  
            sigma_t = torch.sqrt(beta_schedule[t])  
            z = torch.randn(n_sample, 1, 28, 28, device=device)  
  
            if t > 1:  
                x_t = x_t_1 + sigma_t * z  
            else:  
                x_t = x_t_1  
  
    x0 = x_t  
    return x0
```

Generated Images

- Trained on Fashion MNIST

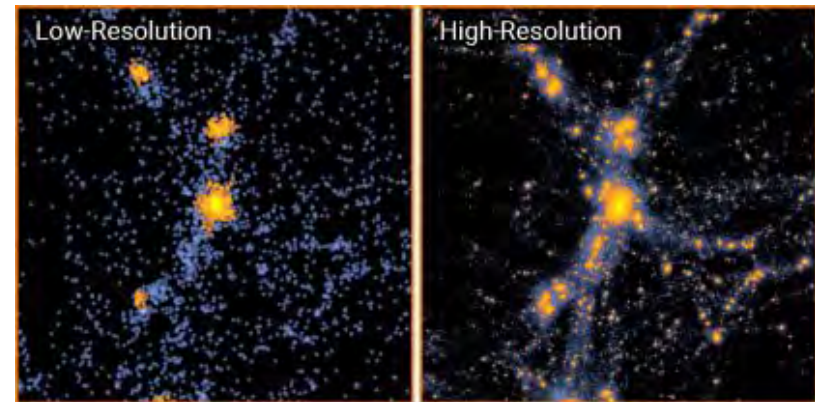
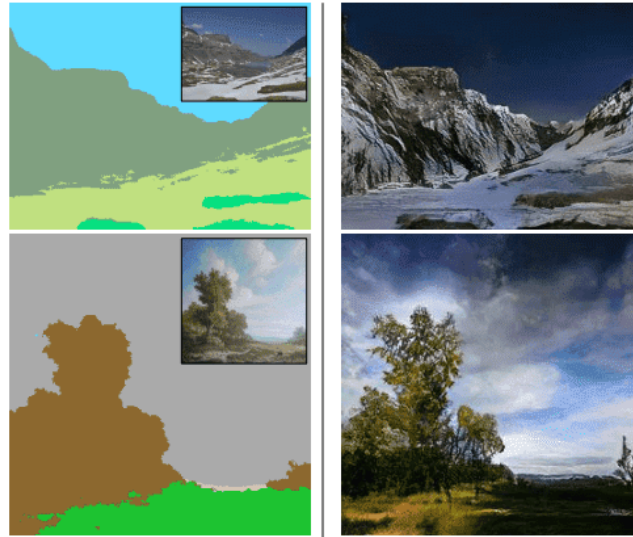


Training data



Generated

Generated Images

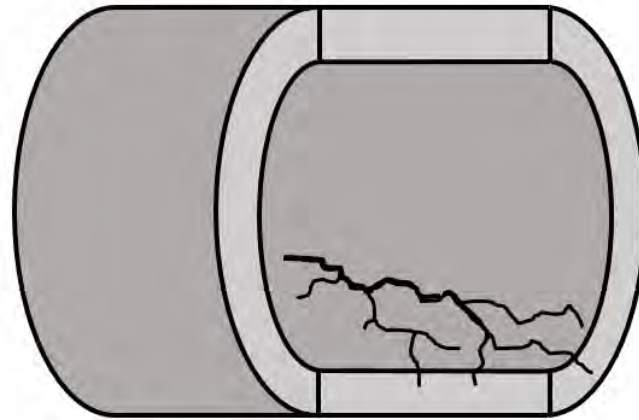
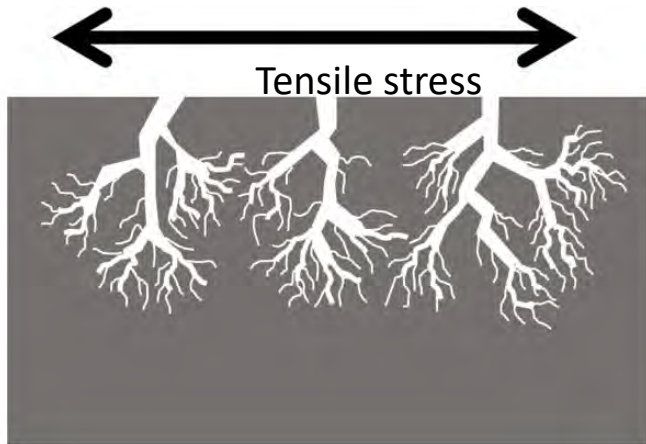


...

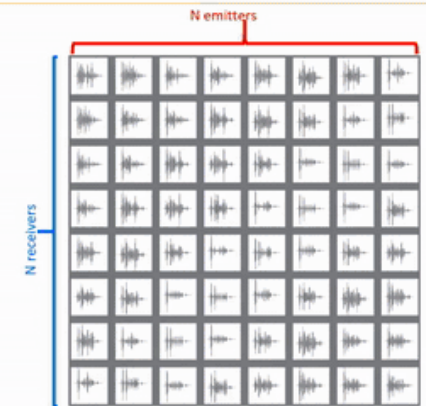
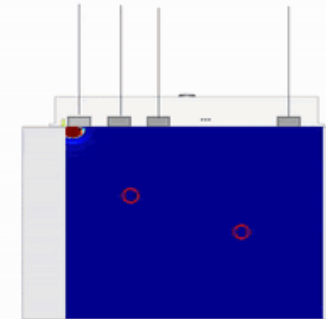
DiffectNet: Diffusion-enabled Conditional Target Generation of Internal Defects

Research Motivation

- Understanding internal cracks is critical in its relation with safety and reliability.
- *Total focusing method (TFM)* using phased array ultrasonic testing is widely adopted for defect detection tasks.
- While data-driven methods have shown promising results for addressing resolution limitation of TFM, it is impractical to obtain large volumes of defect cases, posing *data scarcity* problem.

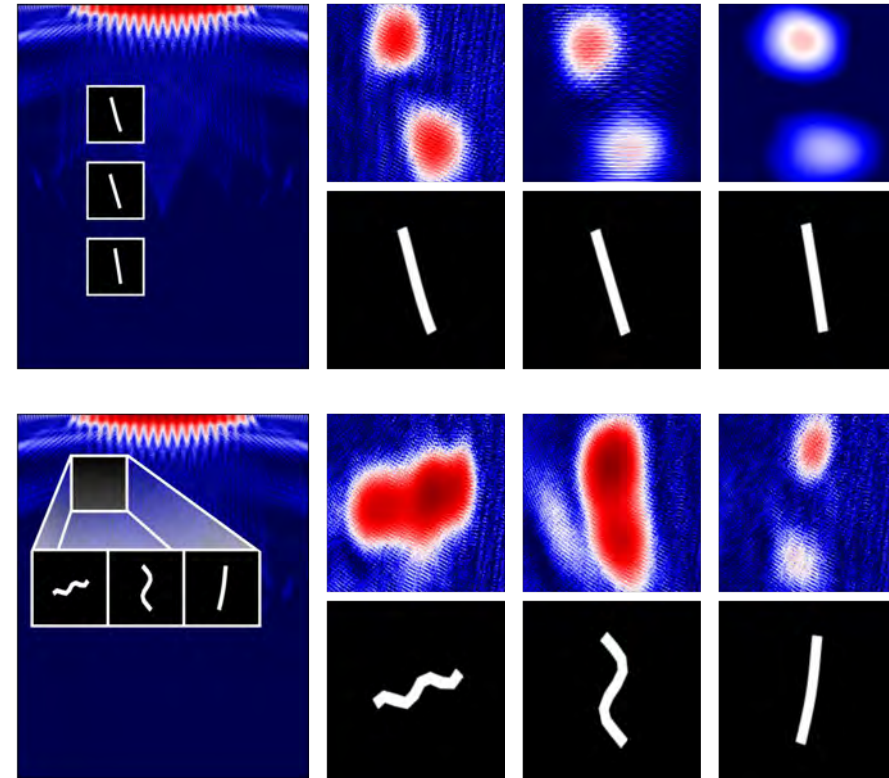
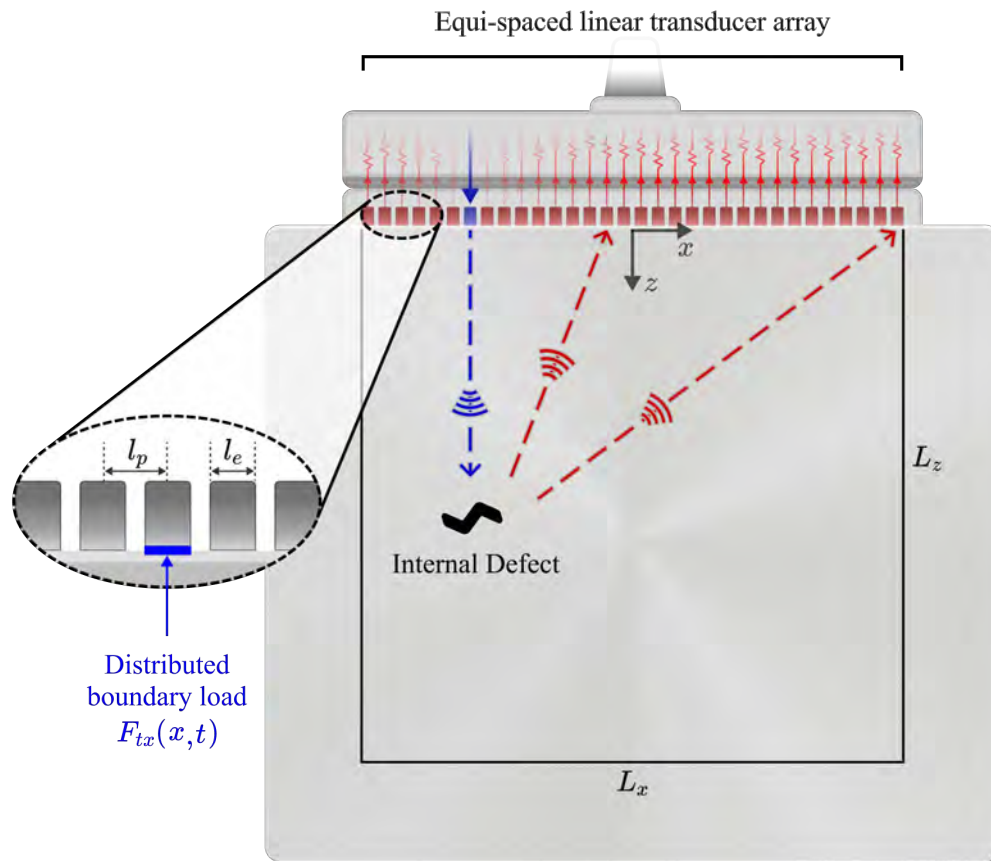


Full Matrix Capture (FMC)



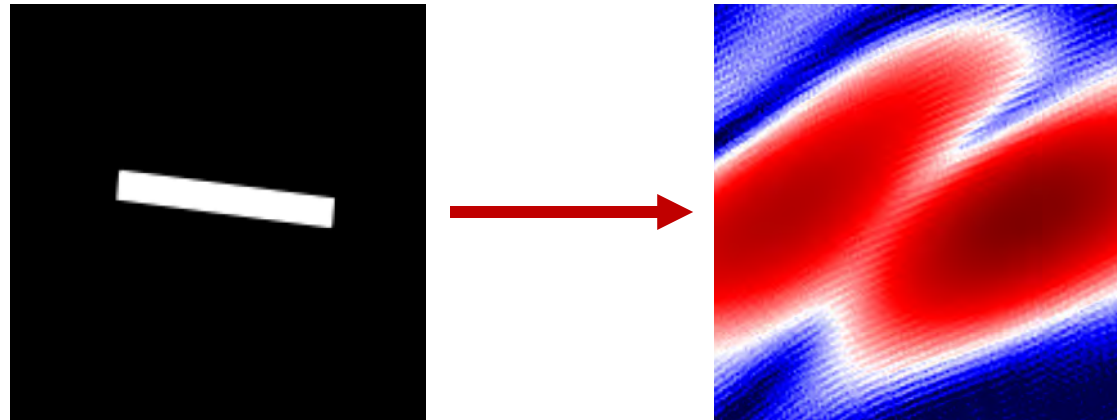
Ultrasonic Imaging for Non-destructive Testing

- TFMs exhibit various characteristics depending on spatial patterns and locations of defects.
- It is required to incorporate defect-related dependencies into the model's generation process.

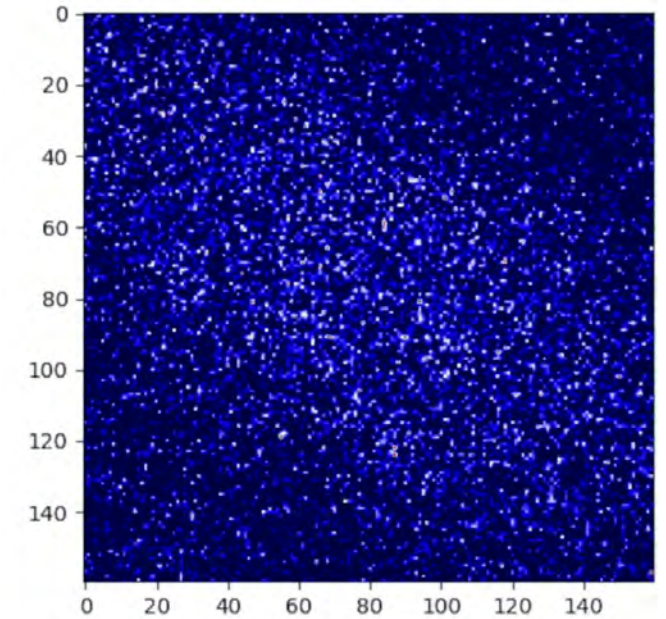
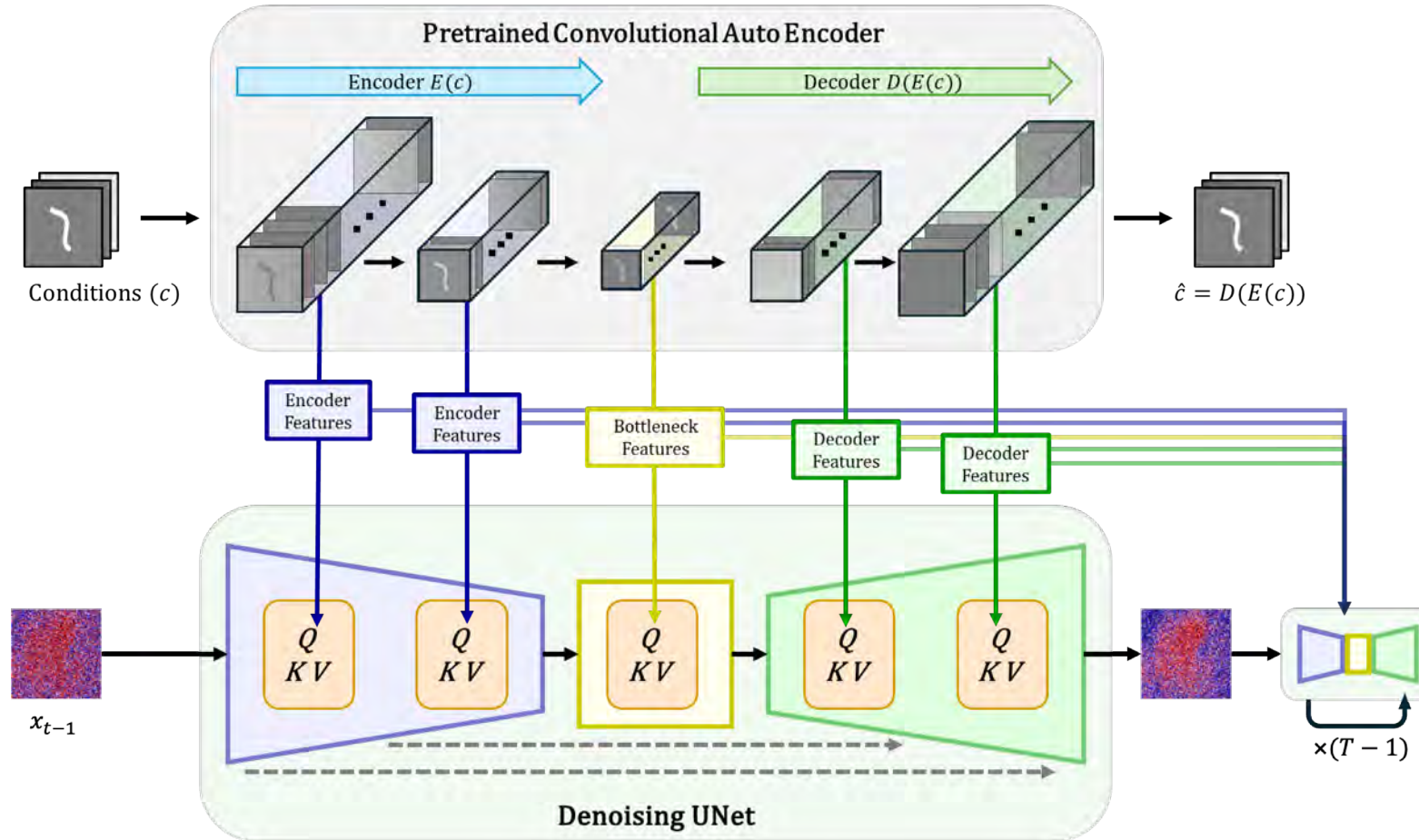


Neural TFM Generation

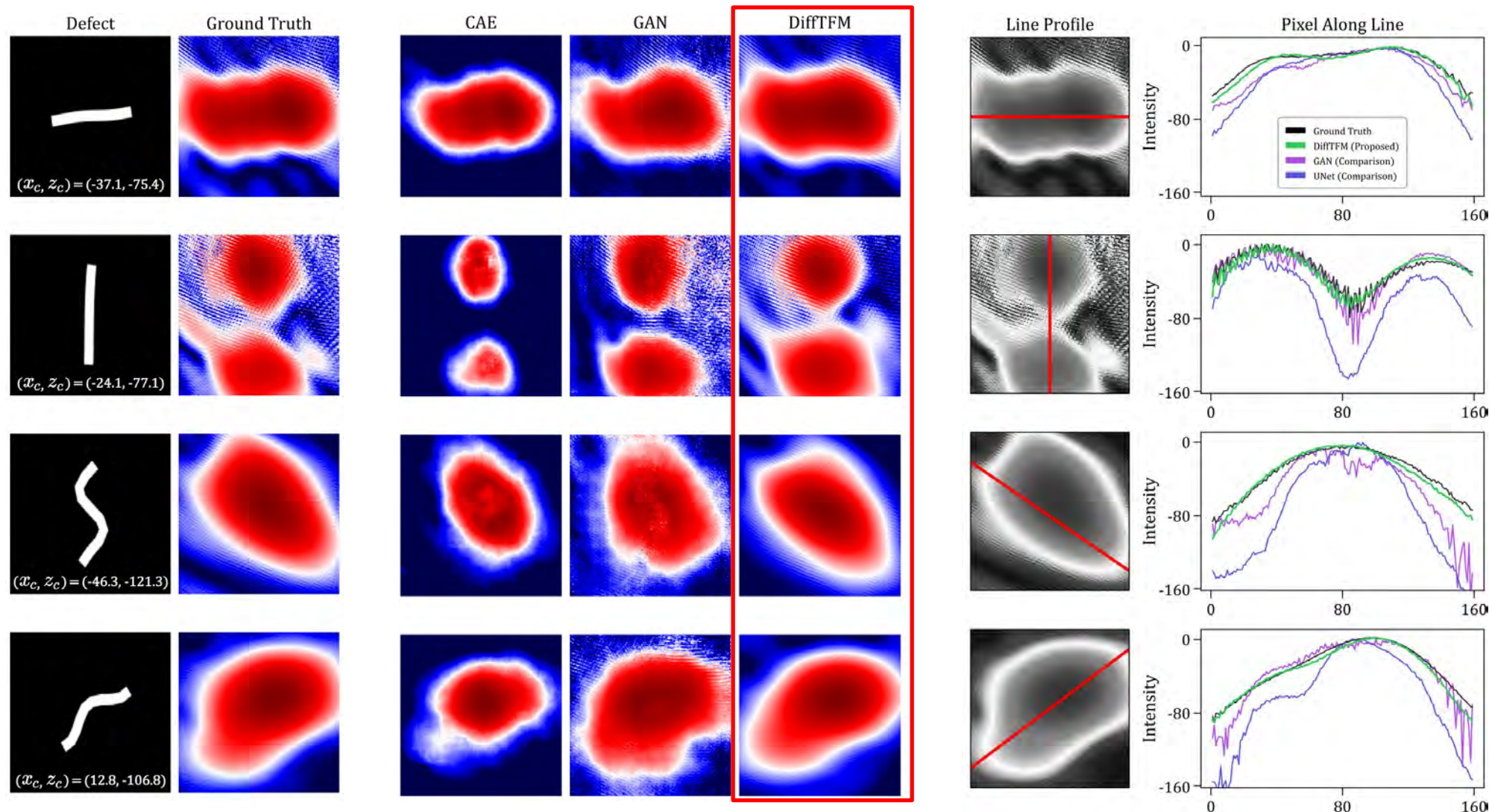
- TFM exhibits various characteristics depending on spatial patterns and locations of defects.
- It is required to incorporate defect-related dependencies into the model's generation process.



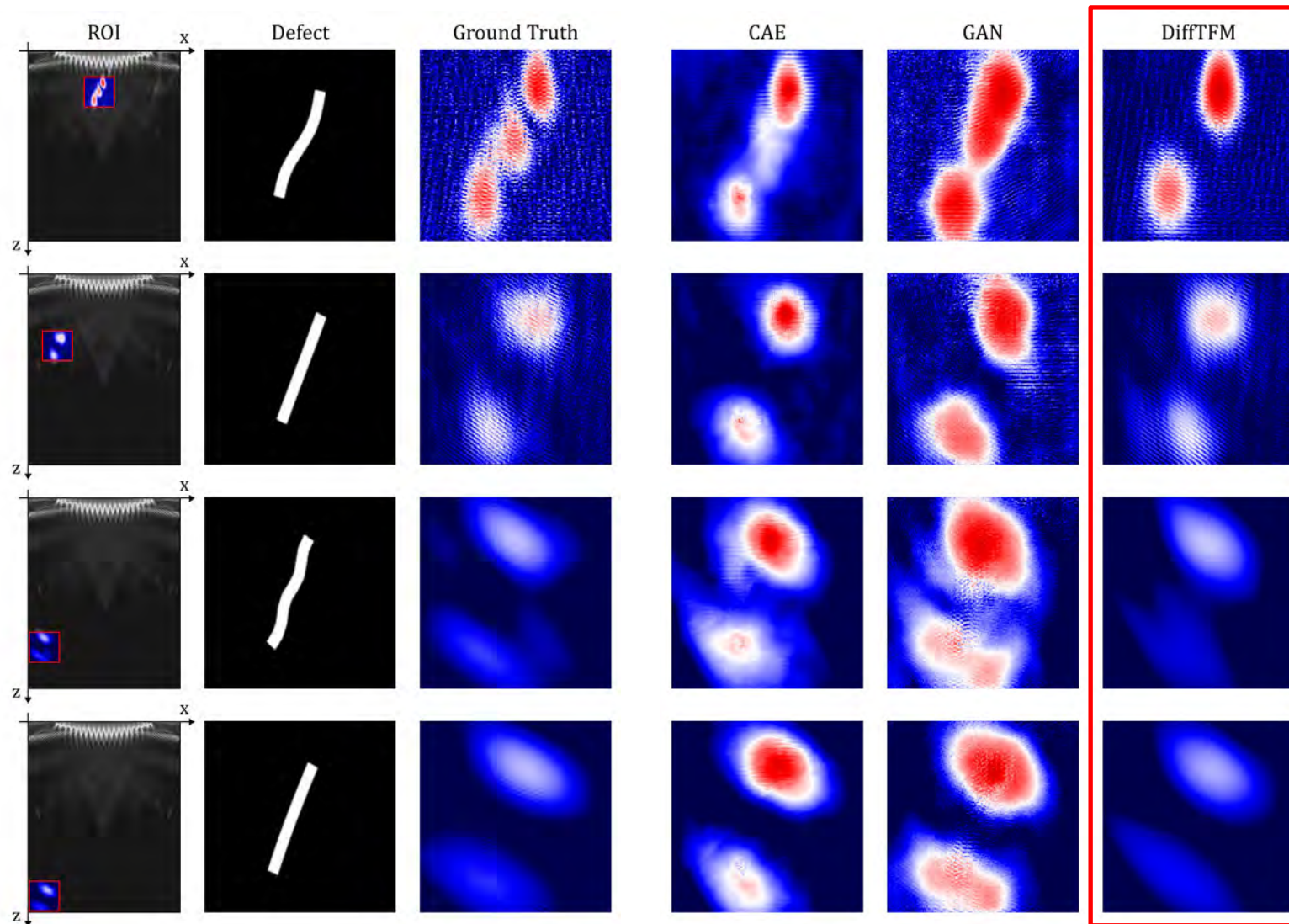
DiffTFM: Defect-aware Diffusion Model via Semi-supervision



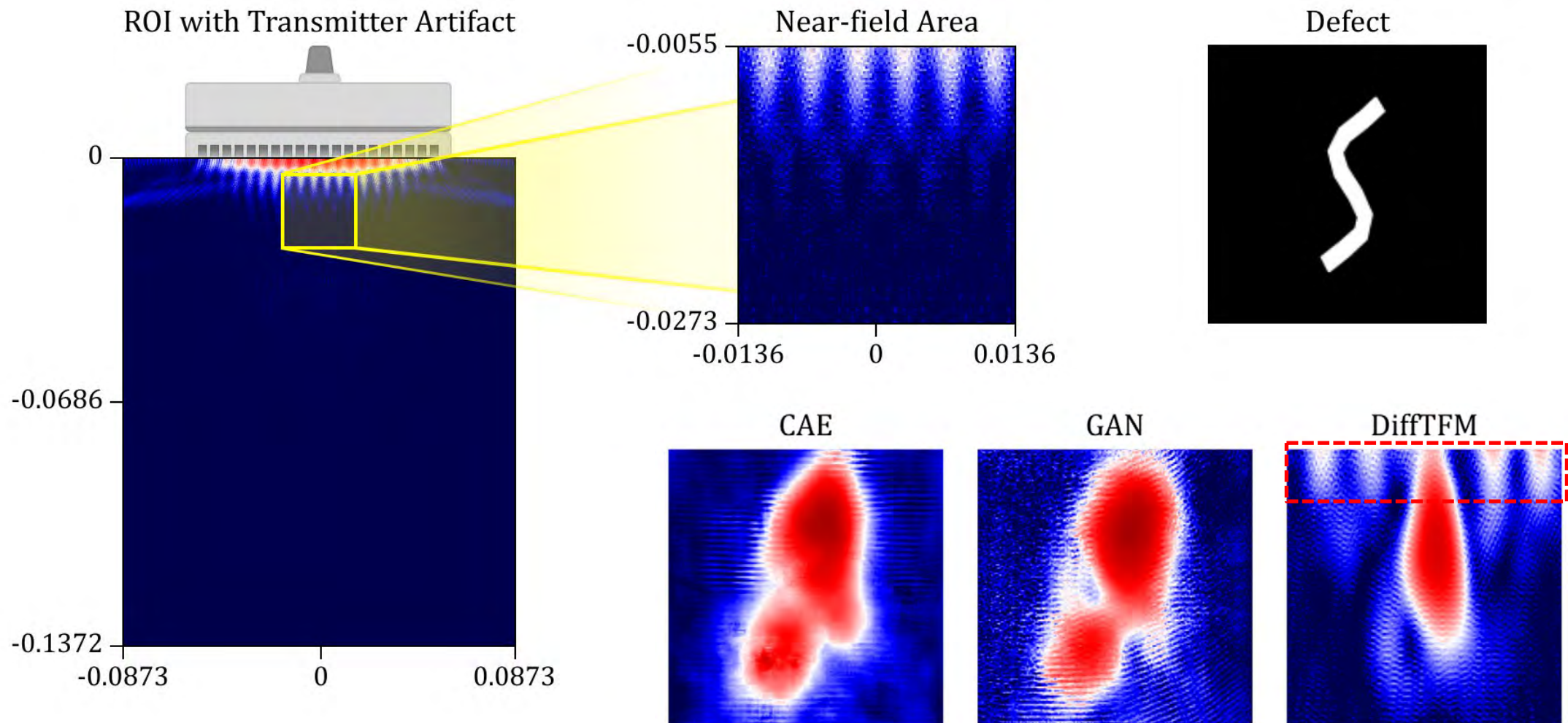
Qualitative Analysis 1: Pattern-wise Consistency



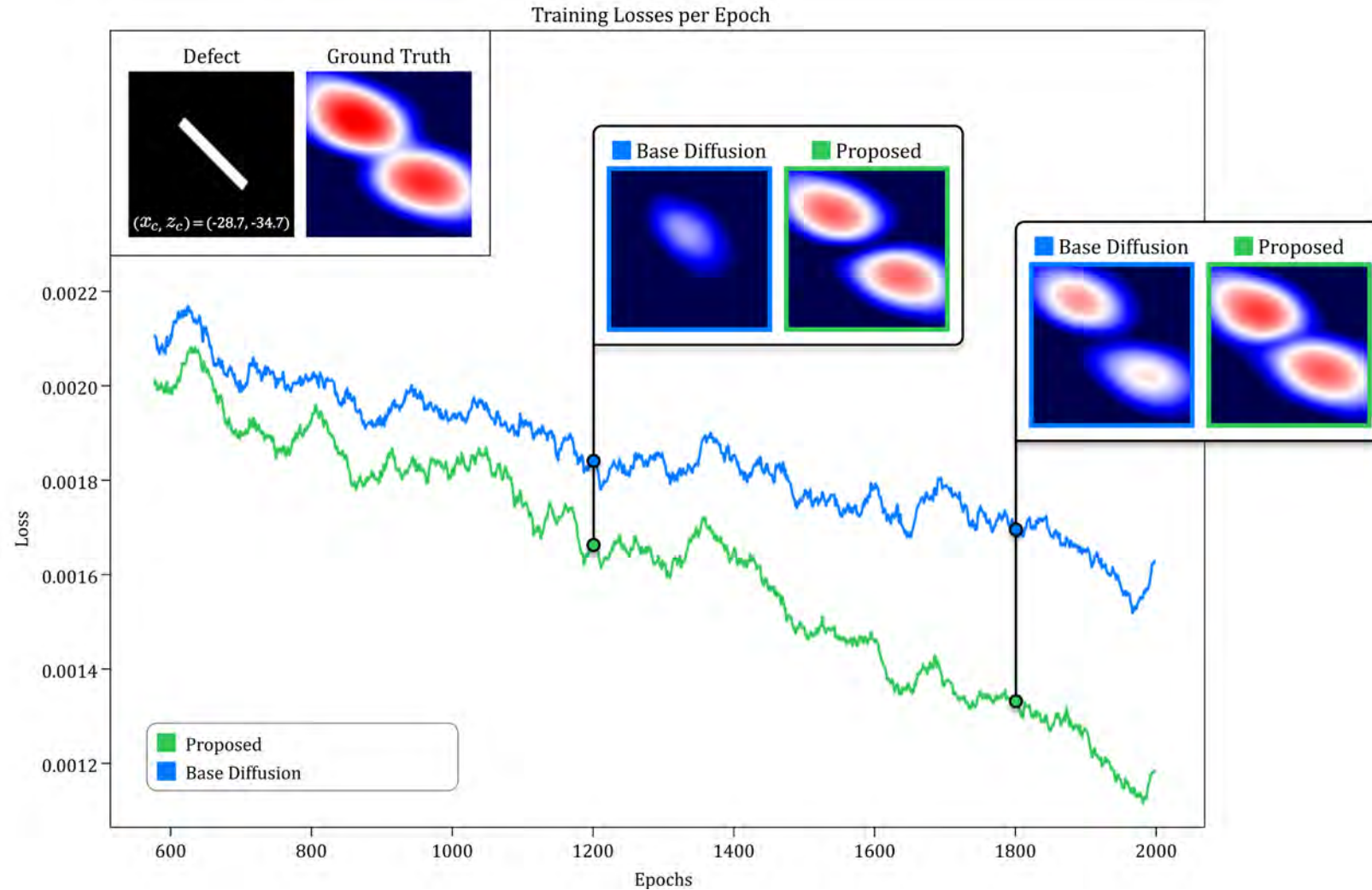
Qualitative Analysis 2: Location-wise Fidelity



Qualitative Analysis 3: Extrapolation

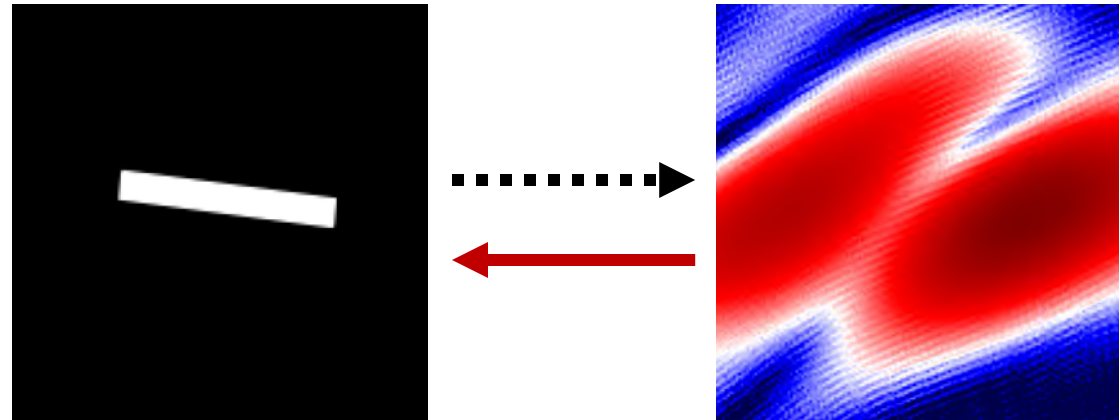


Qualitative Analysis 4: Convergence Capability

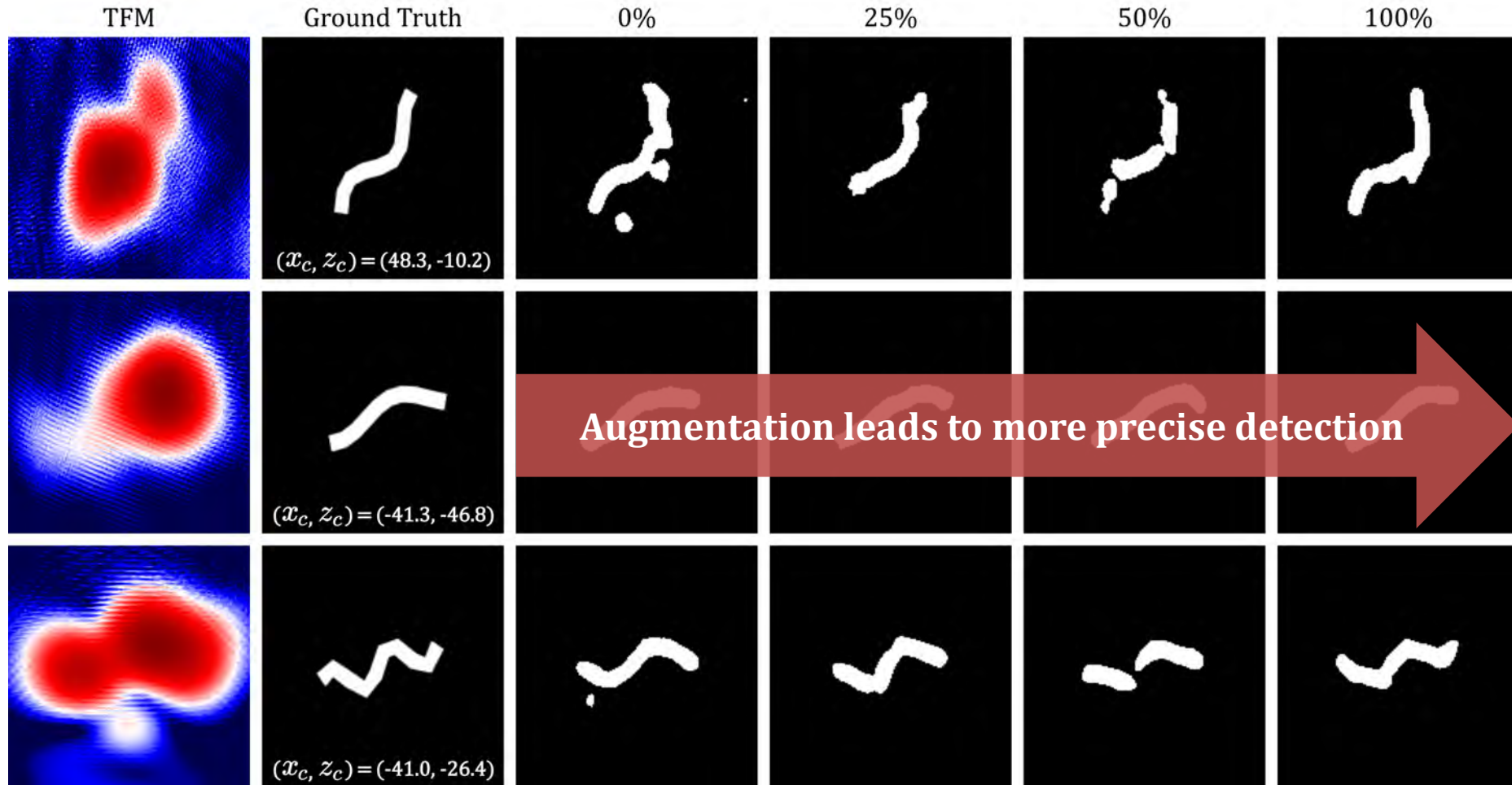


Proposed Method as a Neural Augmentation

- Generation capability provides significant benefits in addressing the data scarcity problems.
- By reversing the causal direction of our focused problem, we incorporate combinations of synthetic TFM cases generated by arbitrary defect information not present in the original dataset.



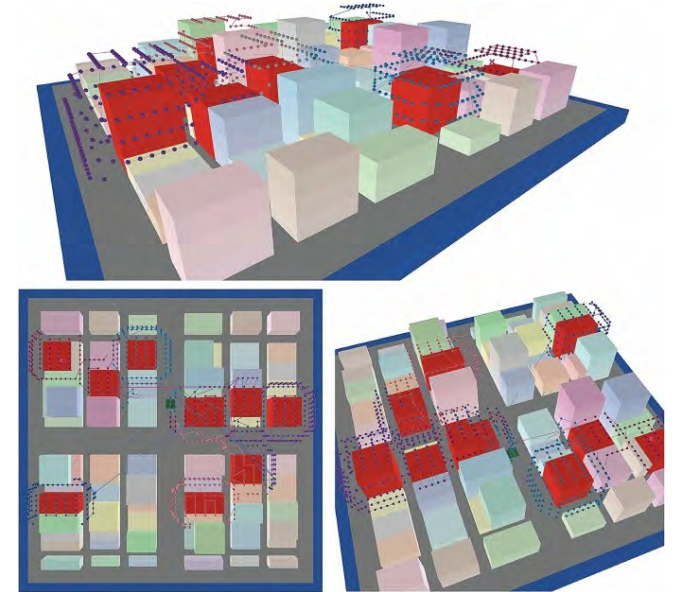
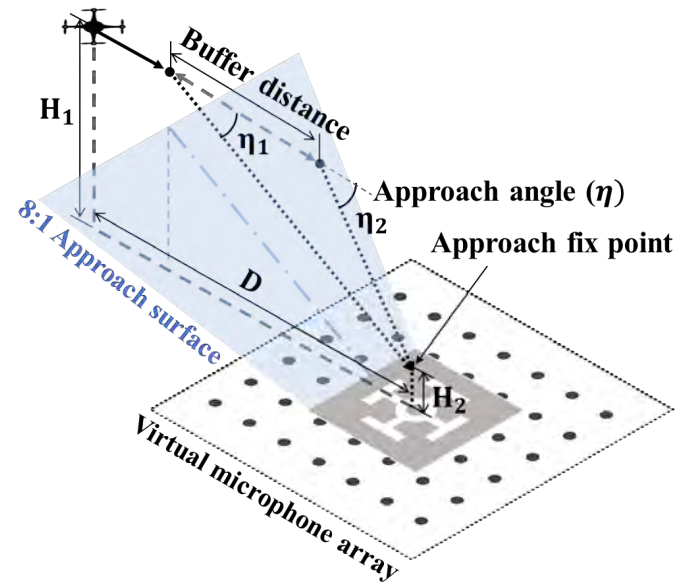
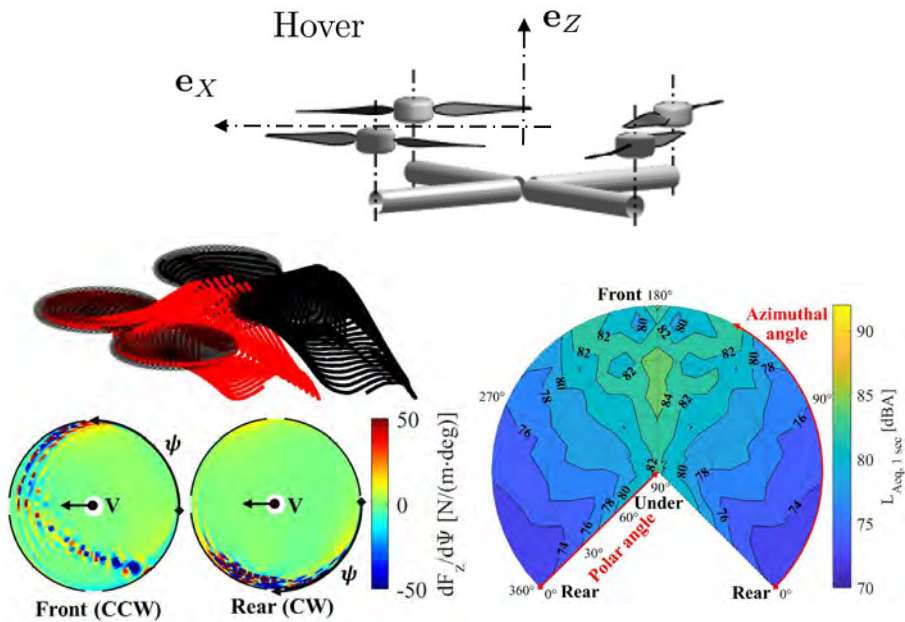
Proposed Method as a Neural Augmentation



Diffusion-based Noise Synthesis

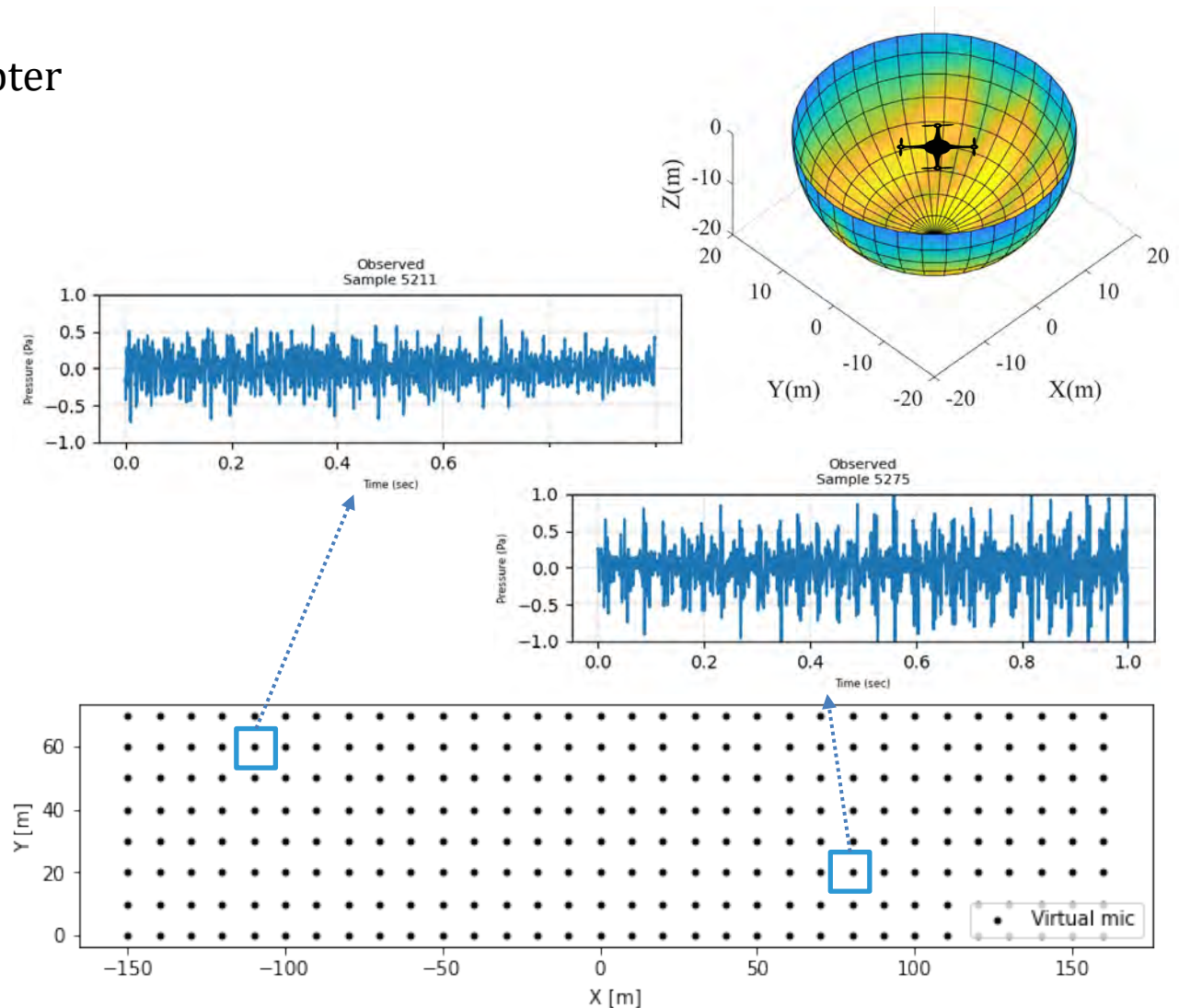
Necessity of Noise Analysis

- Optimization for a variety of systems
 - Acoustic treatments
 - Noise barriers
 - Sound proofing schemes



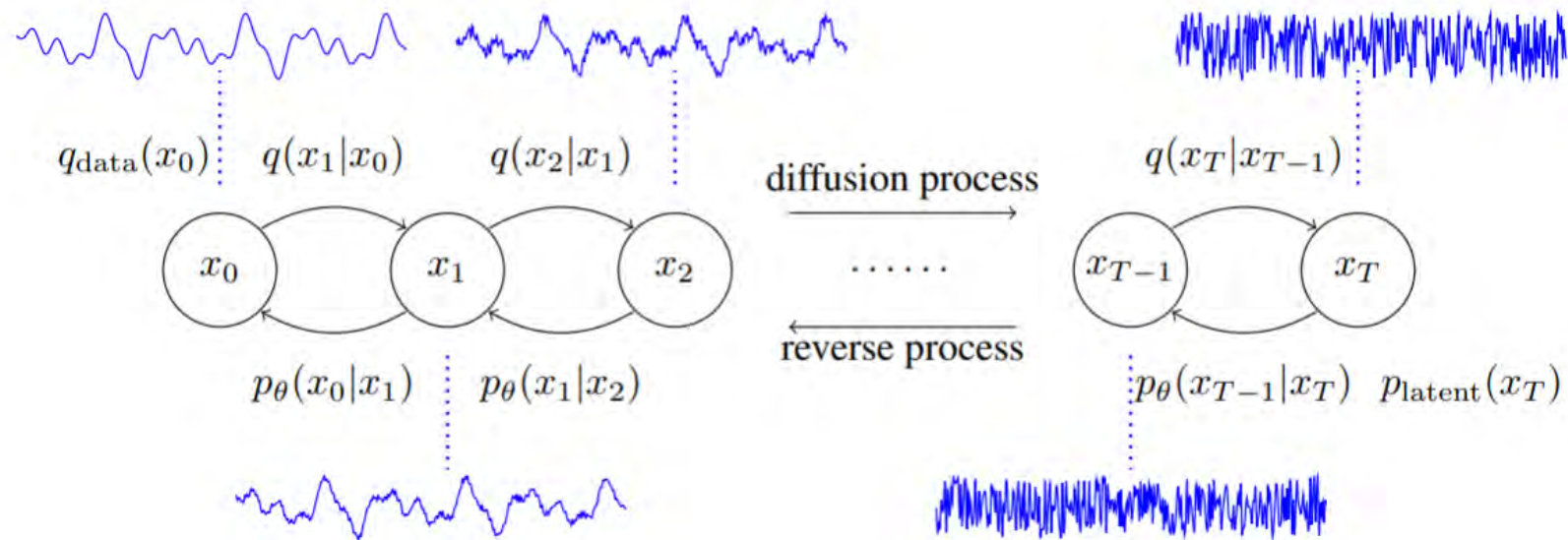
Problem Definition

- Virtual noise auralization for UAM quadcopter
- Given condition
 - Altitude
 - Velocity
 - Pitch angle
 - Locations (x, y)



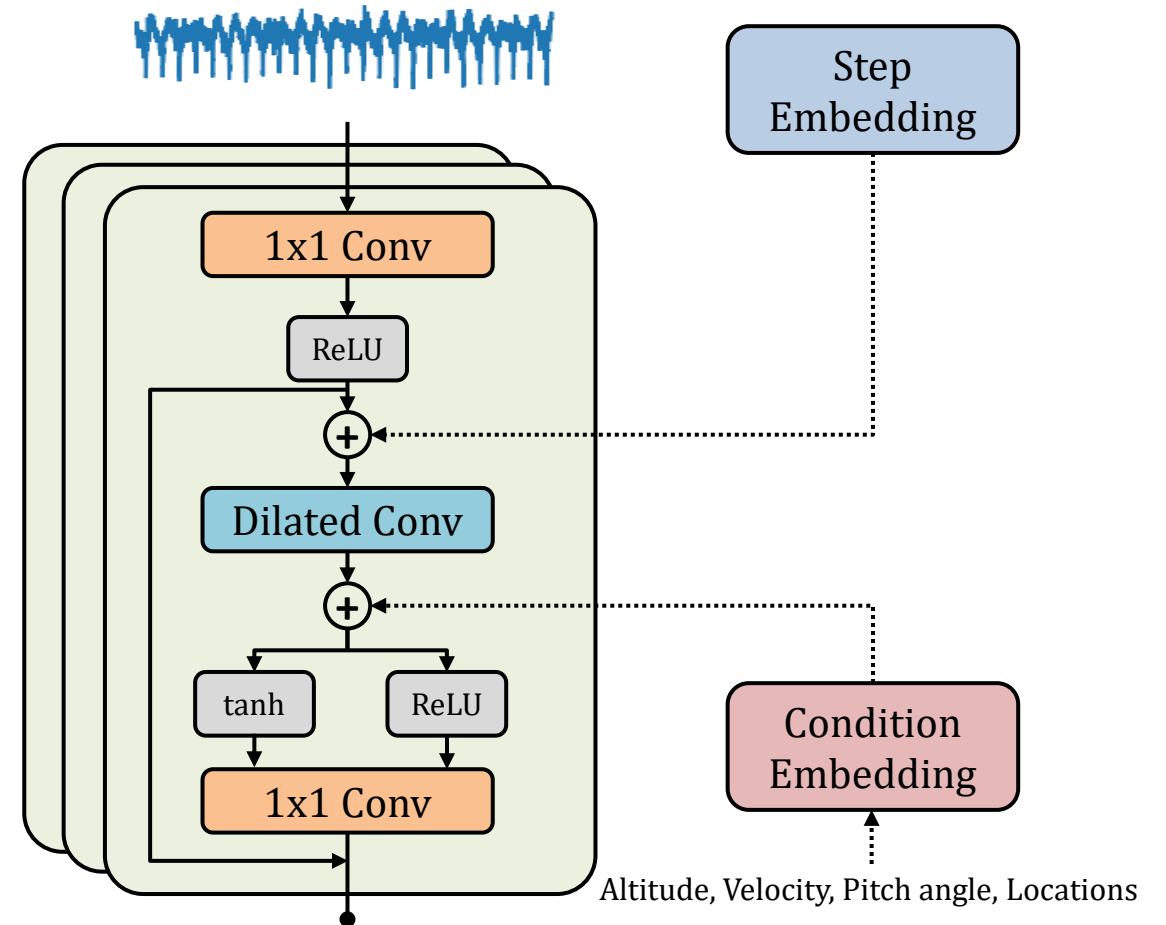
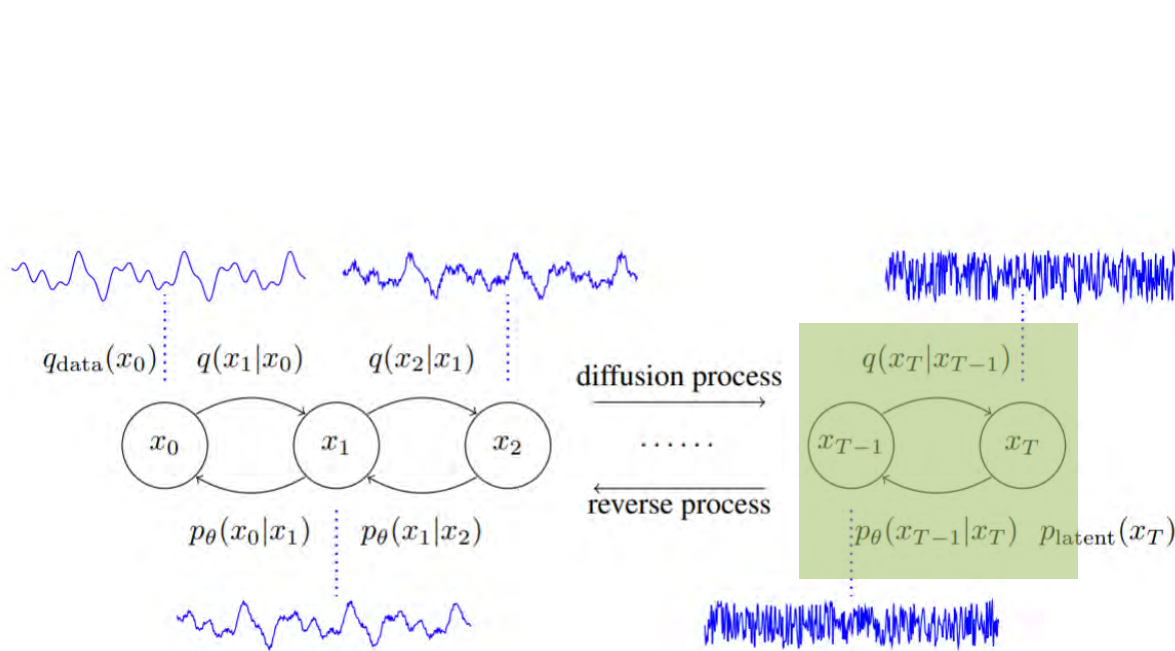
Diffusion Process for Sequential Data

- Progressively adding noise in a forward process and learning to reverse the noise for signal reconstruction

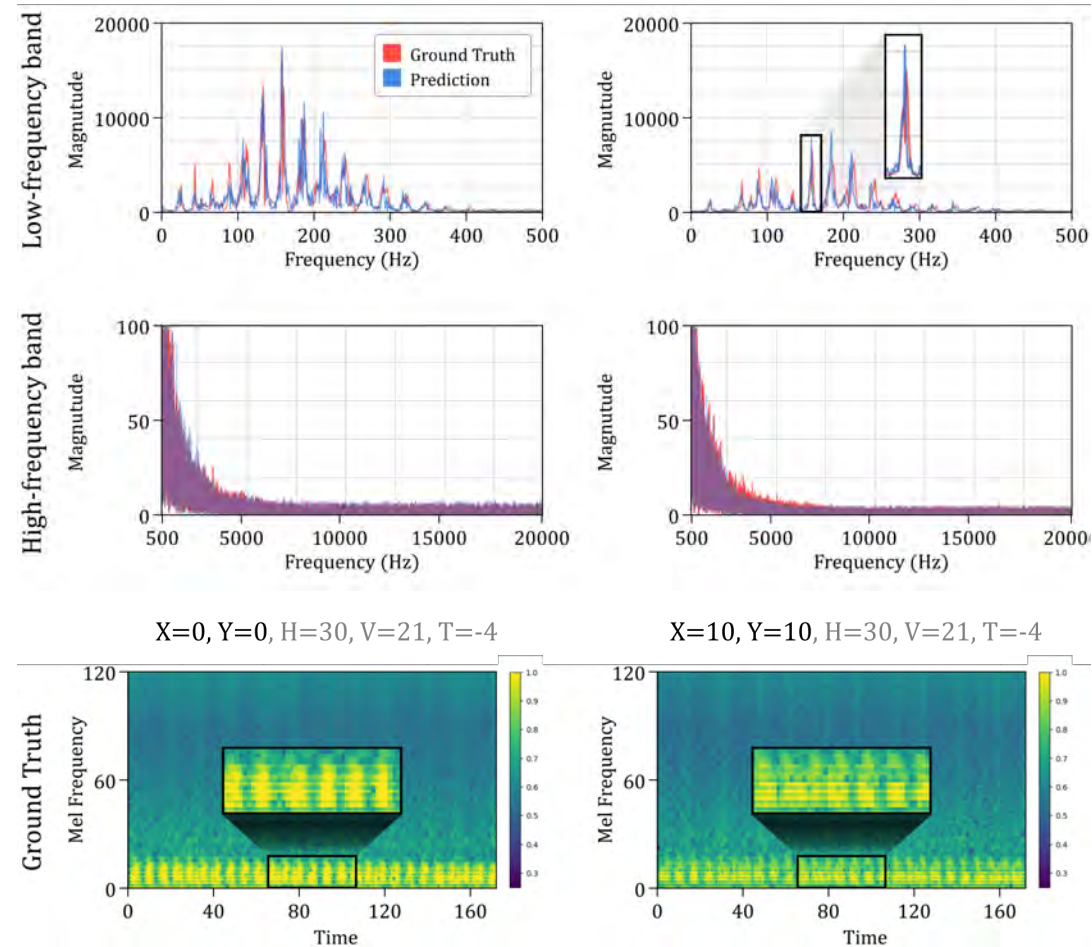


Diffusion Model for Noise Generation

- Diffusion-based approach for reconstructing noise auralization results while capturing temporal dependencies

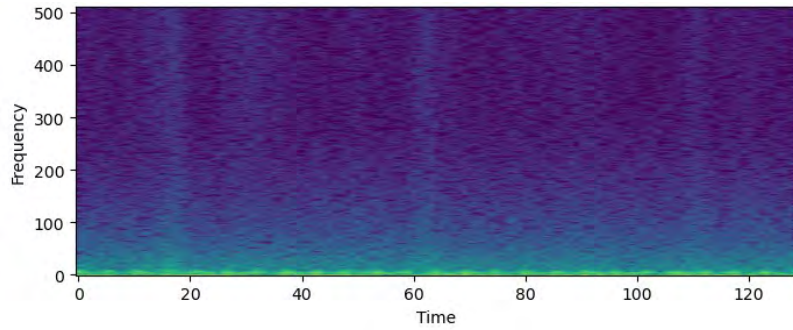


Diffusion Model for Noise Generation

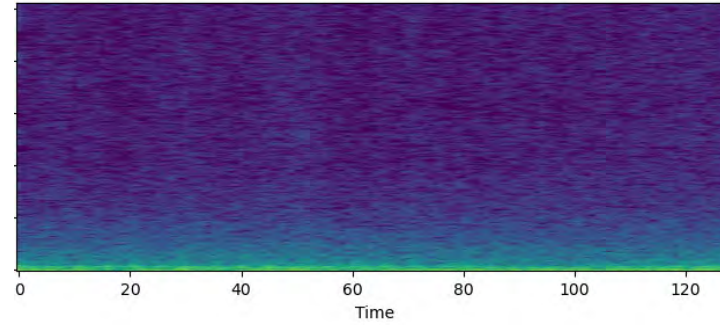


Diffusion Model for Noise Generation

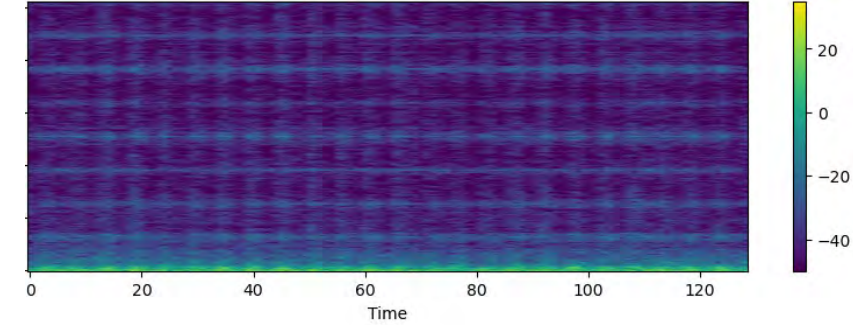
Ground Truth



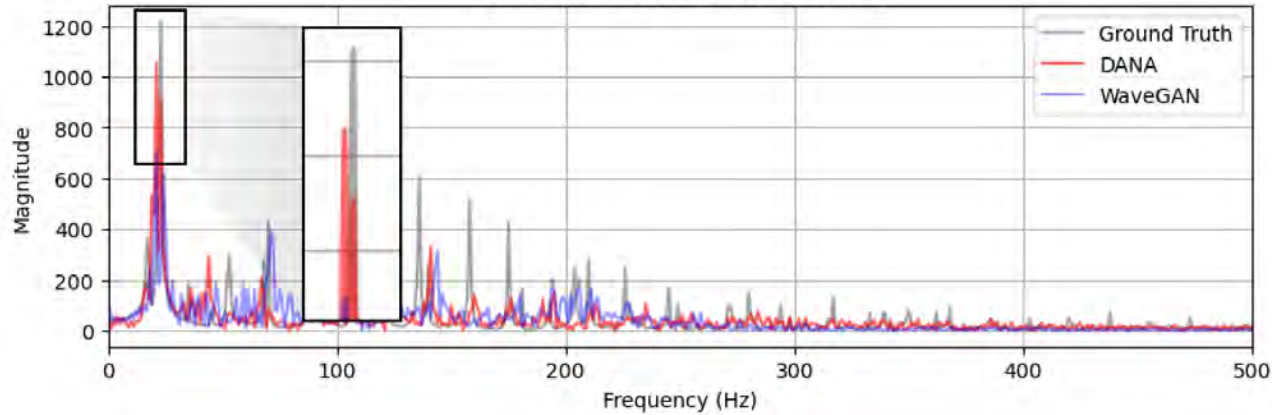
DANA



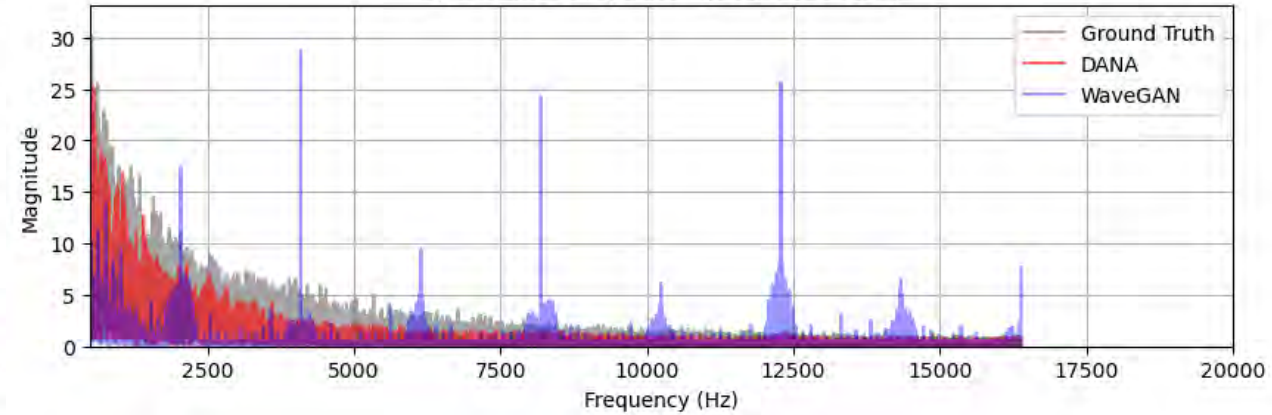
WaveGAN



Spectrum Comparison (0-500 Hz)



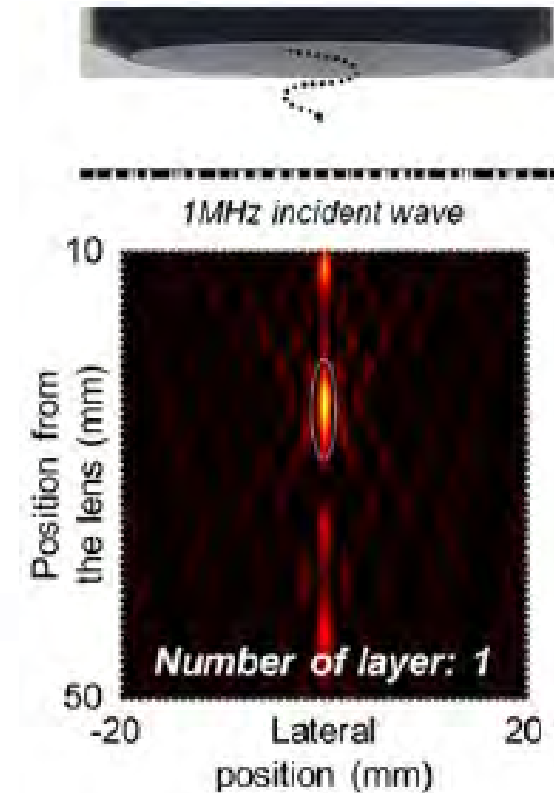
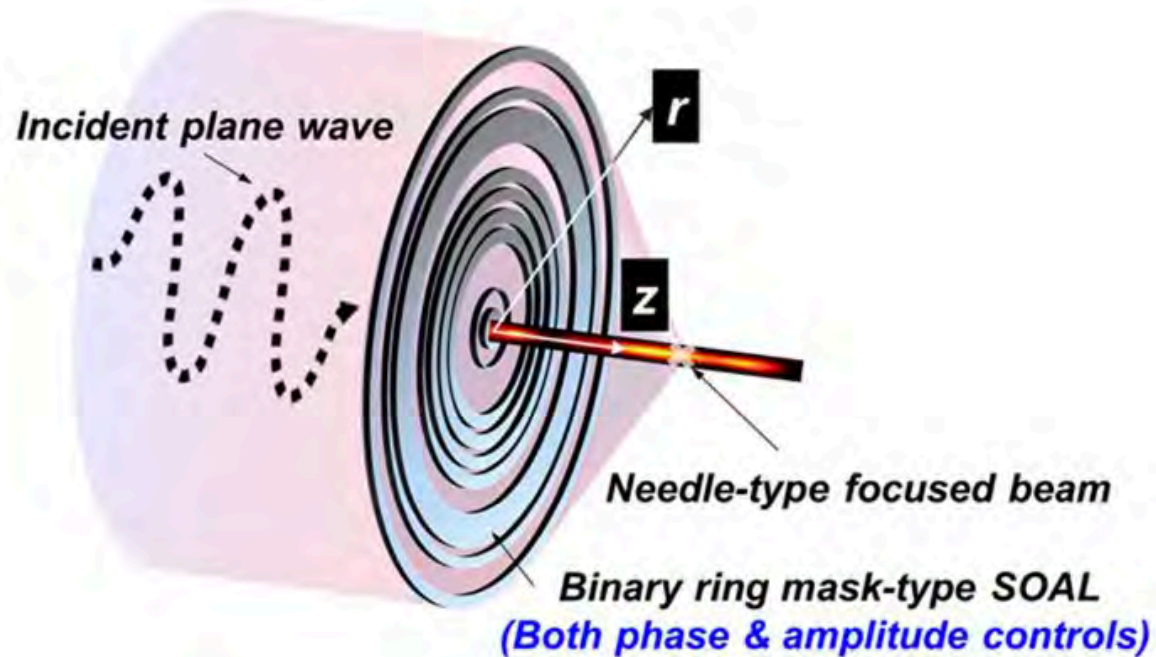
Spectrum Comparison (500-20000 Hz)



CombiDiff: Diffusion for Combinatorial Optimization

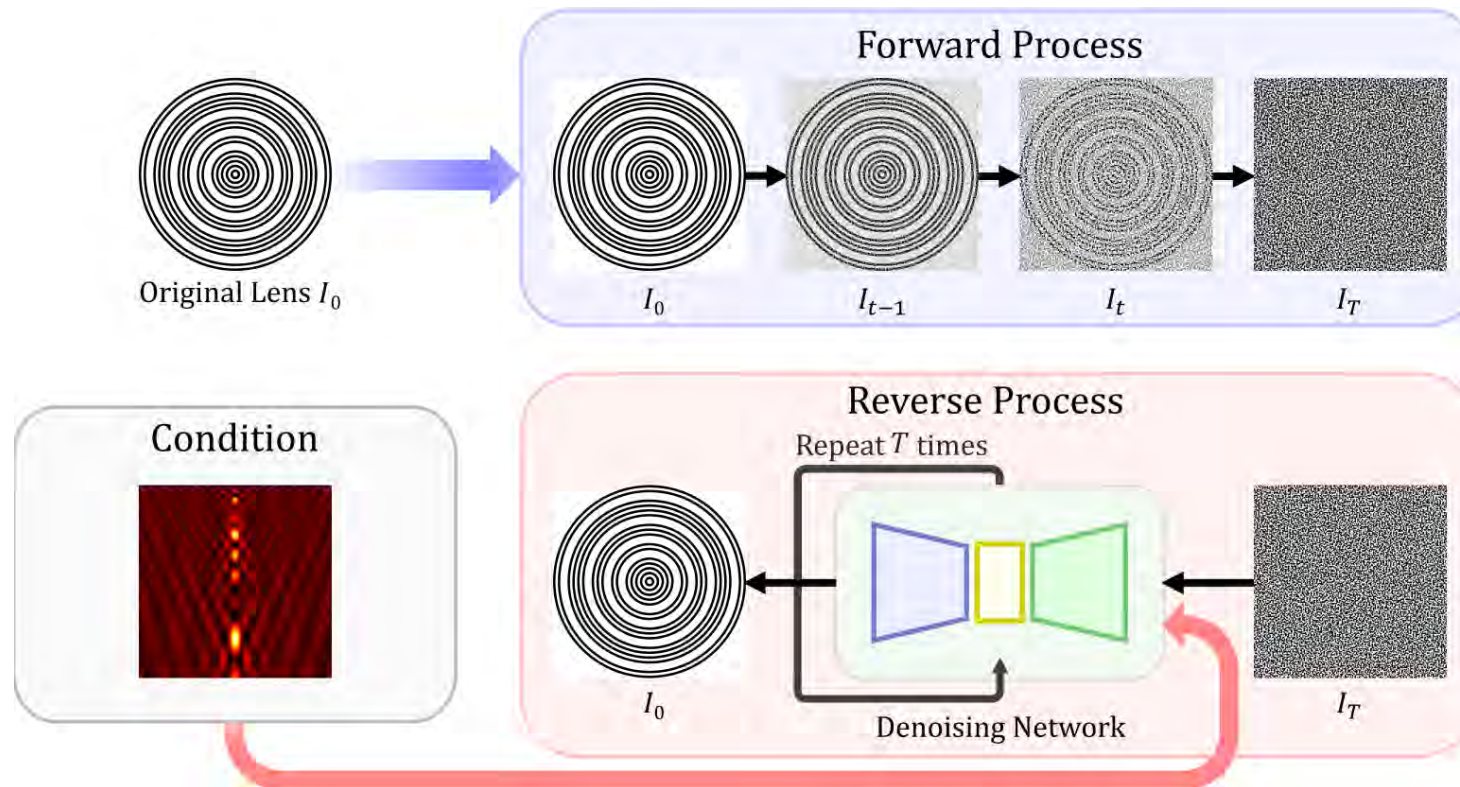
Acoustic Metalens

- The acoustic metalens is a type of ultrathin, subwavelength-thick metasurface that offers high-resolution control of acoustic wavefronts.
- Depending on specific design optimization, different focusing behaviors can be observed owing to interference phenomena within the structure.
 - Understanding of these phenomena is crucial for engineering a metalens that achieves the desired focal position and intensity.



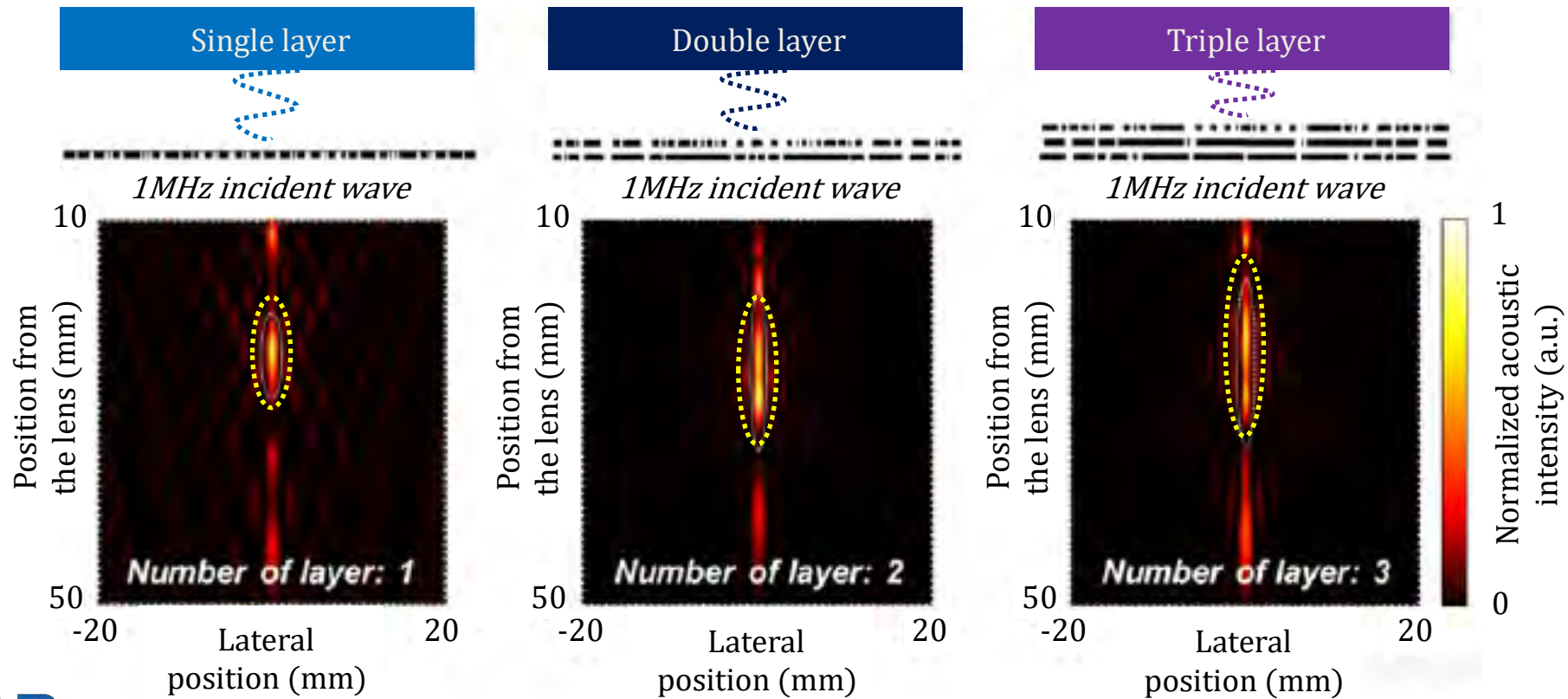
Proposed Framework: Learning to Denoise Diffusion

- Forward process (Diffusion)
 - Progressively adds noise to the original data I_0 over T steps
- Reverse process (Denoising)
 - Iteratively removes noise, recovering I_0 from I_T
 - Trains a neural network to invert the noising steps



Necessity of Combinatorial Optimization Formulation

- Stacking multiple metasurface lenses can extend the axial length of the focal region and reduce intensity variations.
- Accurately reflecting the acoustic pressure distribution after transmission through several metasurface layers is challenging.



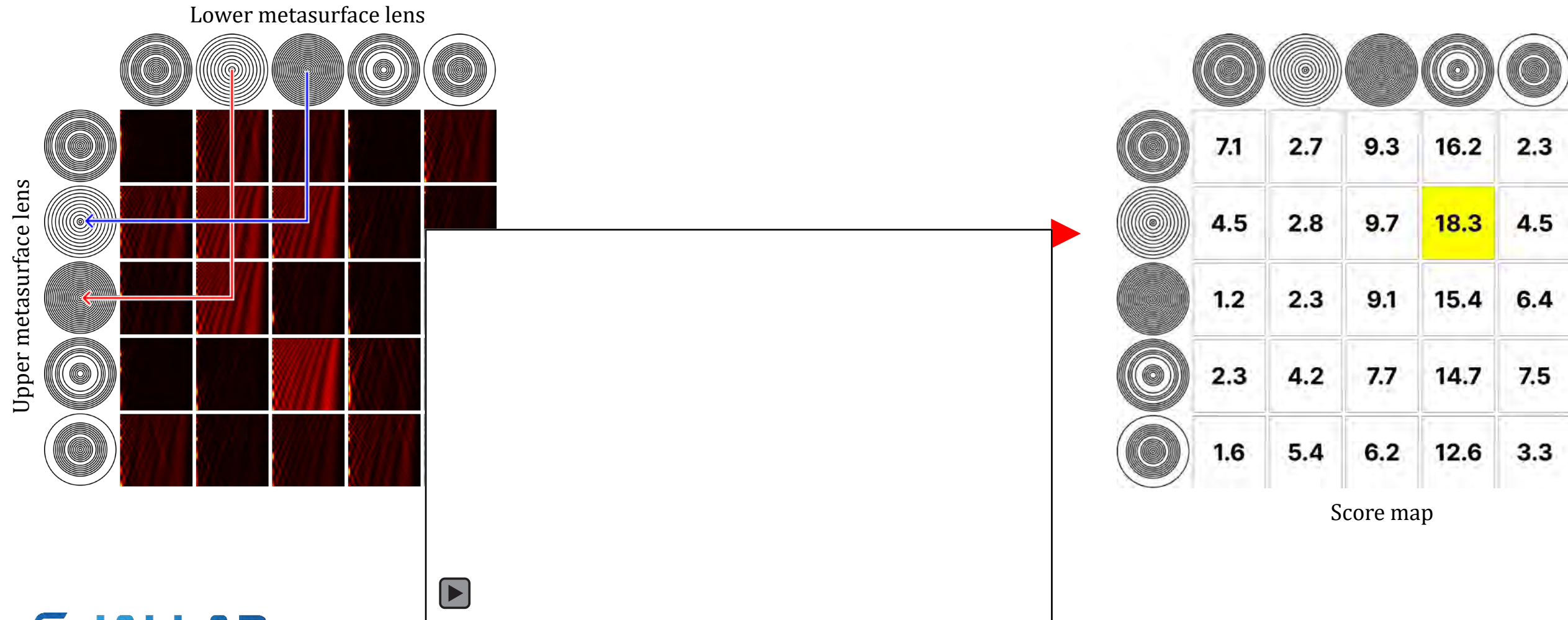
Necessity of Combinatorial Optimization Formulation

We formulate our problem as **combinatorial design optimization** for specific target acoustic pressure field.

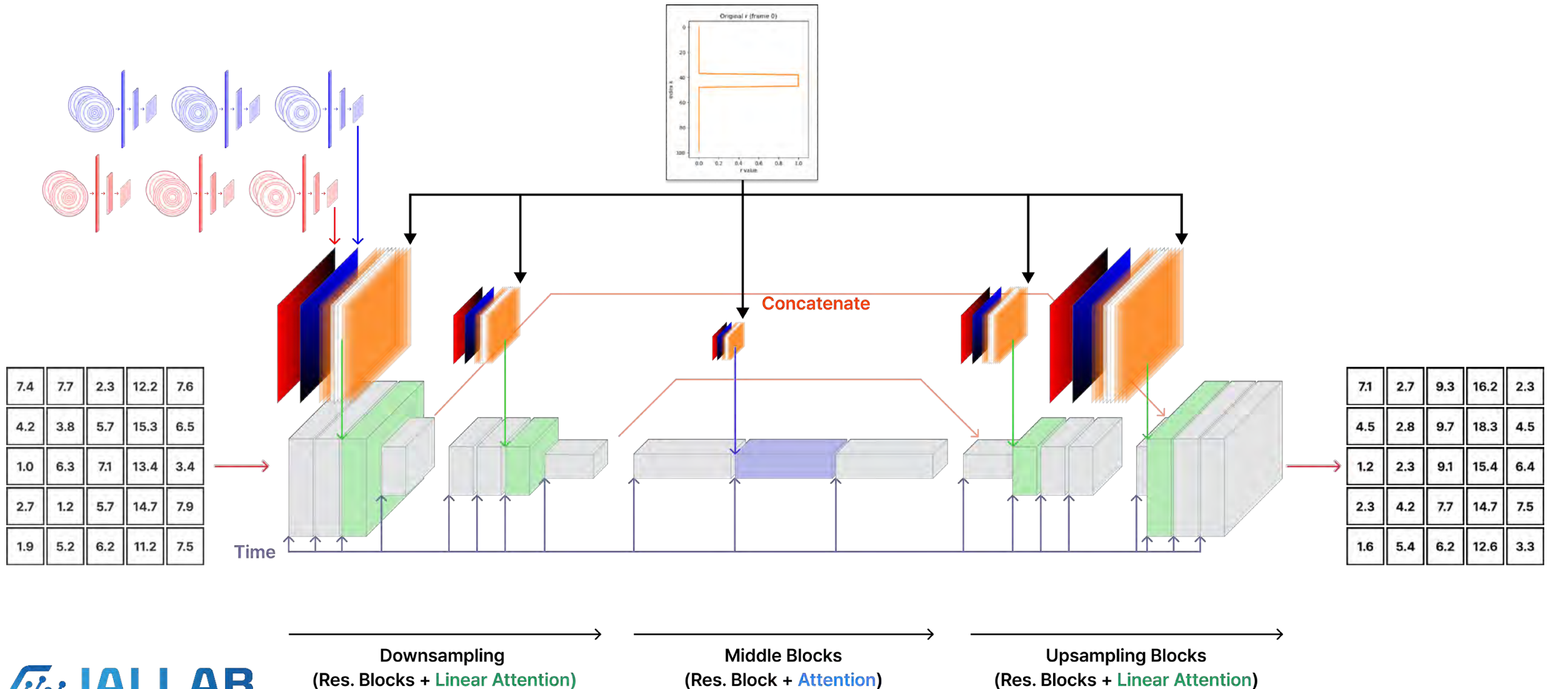
← Focusing region



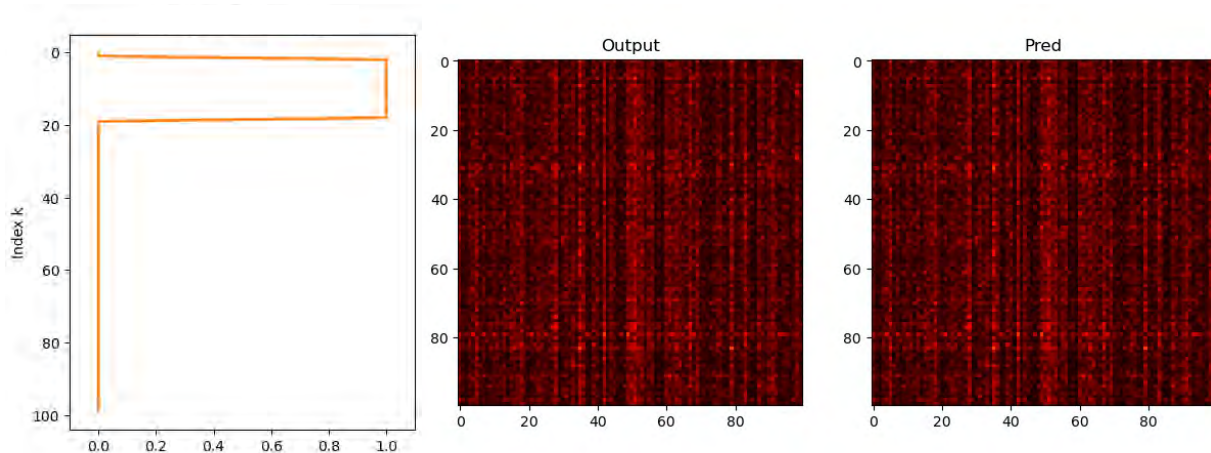
Diffusion Model as a Combinatorial Optimizer



Diffusion Model as a Combinatorial Optimizer



Qualitative Analysis: High Scoring Combinations

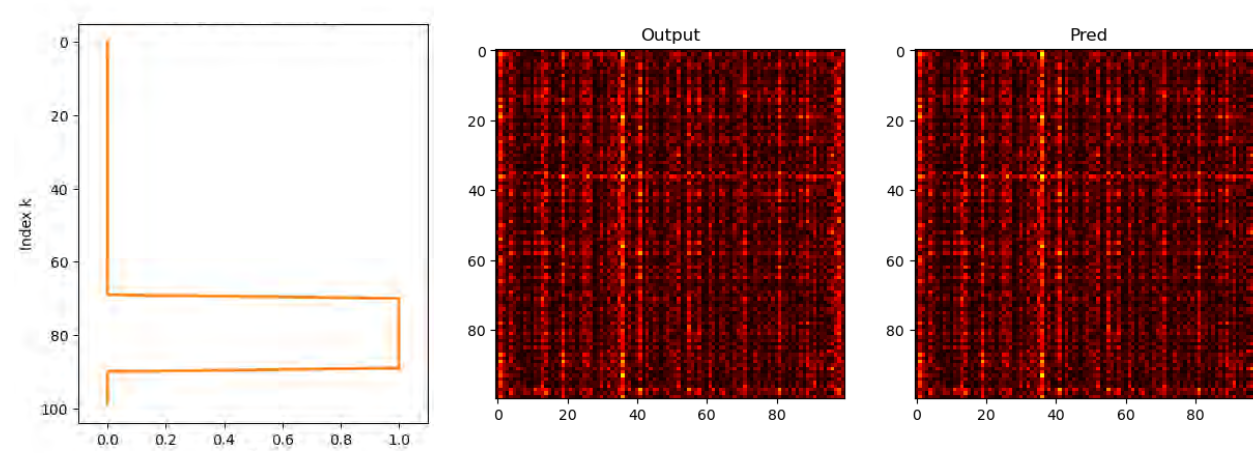


► Top-5 ground_truth pixels (row, col, value)

(83, 35) → 0.46556
 (83, 79) → 0.46361
 (31, 35) → 0.45076
 (79, 42) → 0.44103
 (79, 49) → 0.43514

► Top-5 predicted pixels (row, col, value)

(83, 35) → 0.46293
 (83, 79) → 0.45699
 (31, 35) → 0.45073
 (79, 42) → 0.44137
 (79, 49) → 0.43363

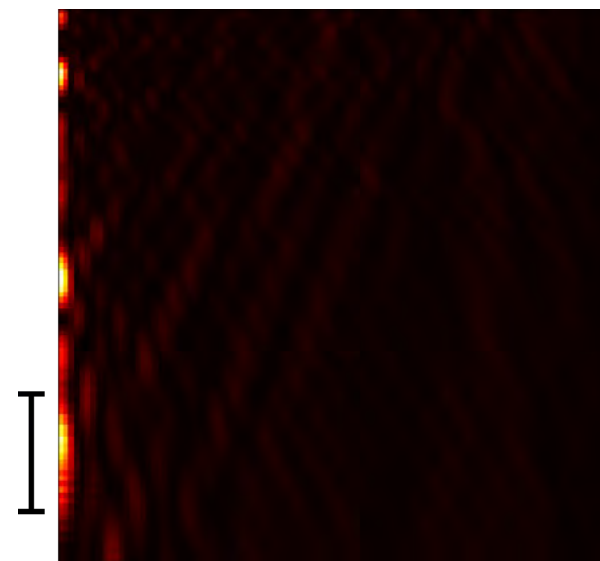


► Top-5 ground_truth pixels (row, col, value)

(36, 19) → 0.79624
 (19, 36) → 0.78201
 (36, 36) → 0.76340
 (1, 36) → 0.75842
 (93, 36) → 0.71675

► Top-5 predicted pixels

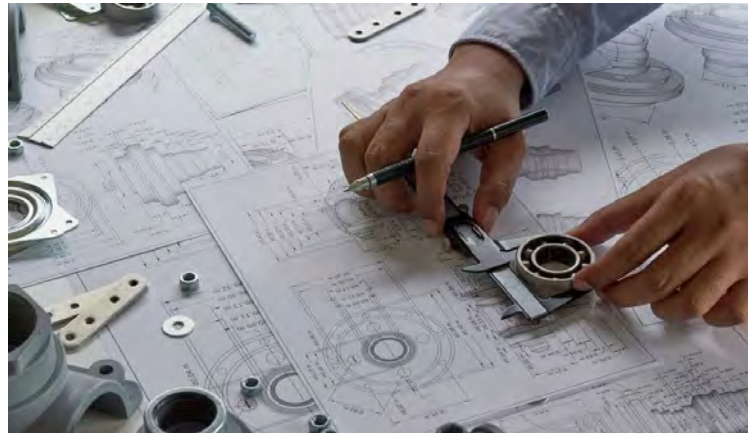
(36, 19) → 0.79885
 (19, 36) → 0.78727
 (36, 36) → 0.75398
 (1, 36) → 0.75152
 (93, 36) → 0.71775



Prompt-guided Diffusion Model for Generative Designs under Desired Properties

Research Motivation

- Traditional engineering design processes for optimizing desired performance are time-consuming and heavily reliant on iteration procedures and expert intuition.
- Recent advances in deep generative approaches offer the potential to autonomously explore engineering designs that satisfy desired dynamic properties.
- Remaining challenges
 - (-) Limited in generating and optimizing user-desired designs from low-level specifications, e.g., prompts.



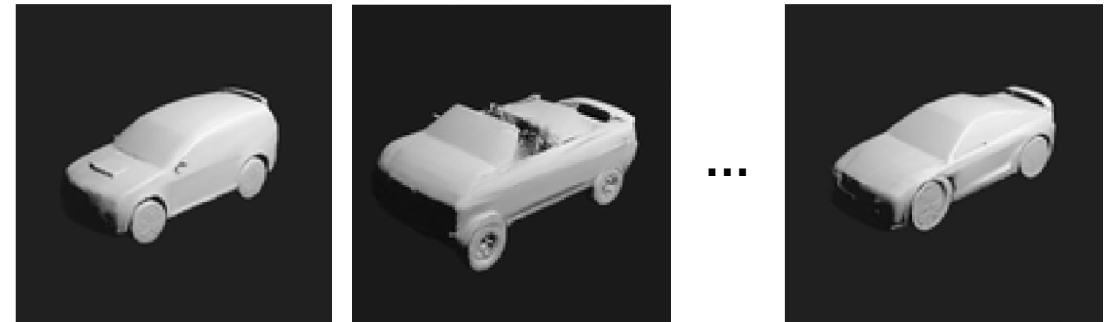
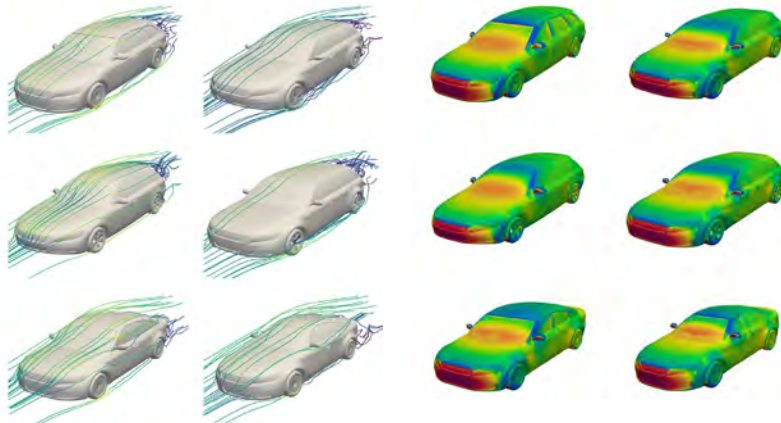
Our Focused Problem

- AI-enabled generative designs that satisfy *user prompts* as well as *desired dynamic properties*, which can significantly enhance the effectiveness and efficiency of engineering prototyping processes.

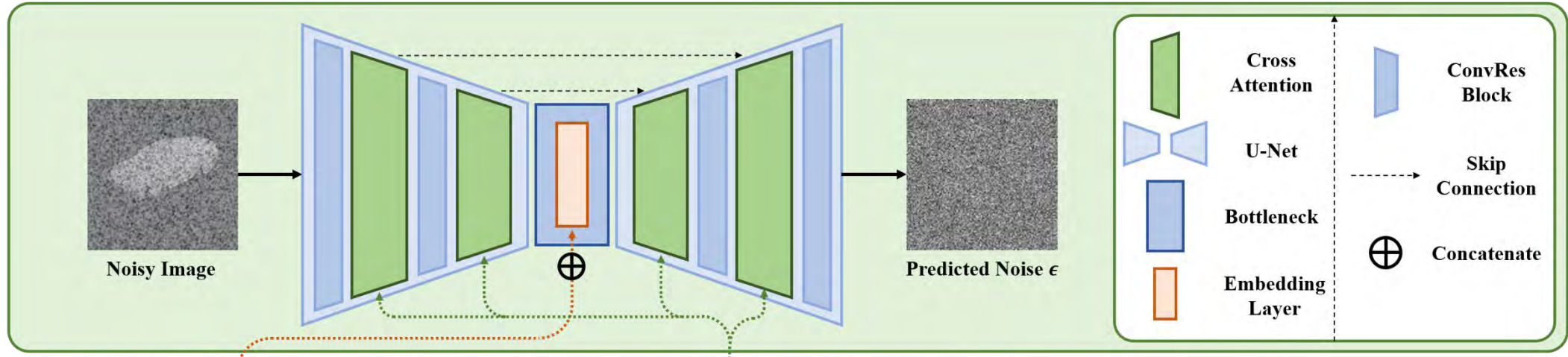
Language prompts

“Can I get vehicle design candidates with sporty and luxurious SUVs?”

Desired aerodynamic properties

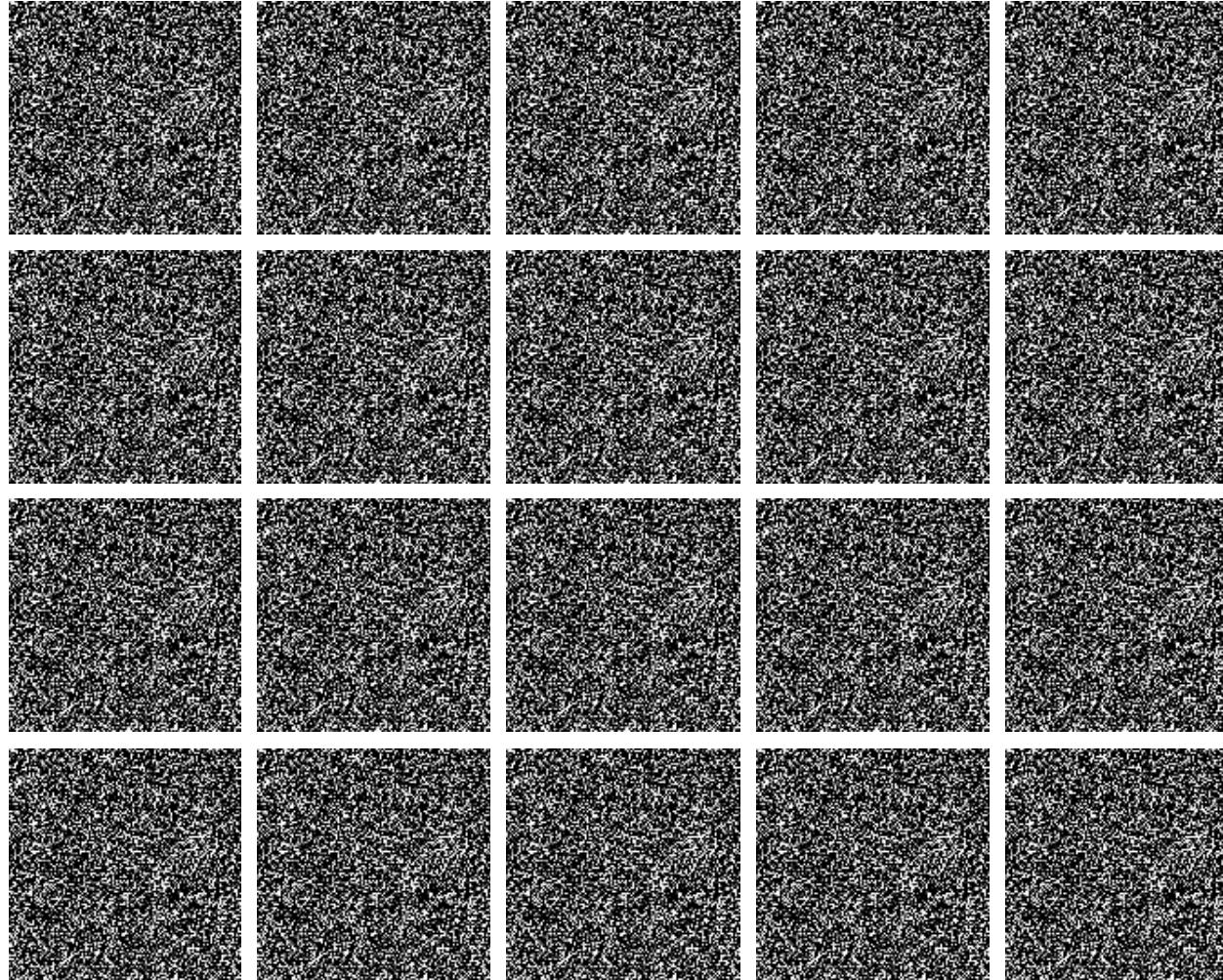


Prompt-engineered and Target-aware Diffusion Model

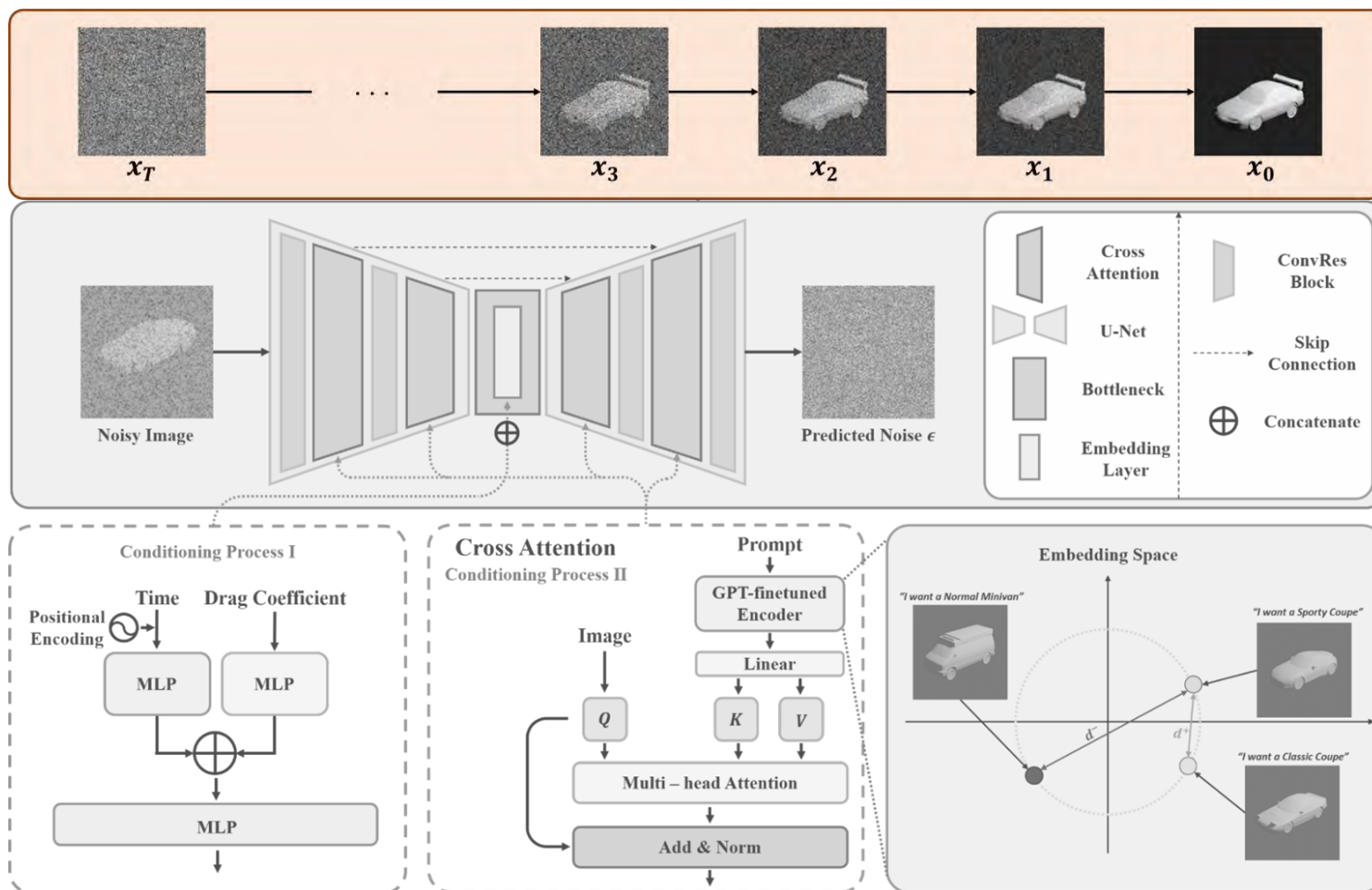


Generation Process

Q. "Design sporty sport cars with a drag coefficient of 0.23..."



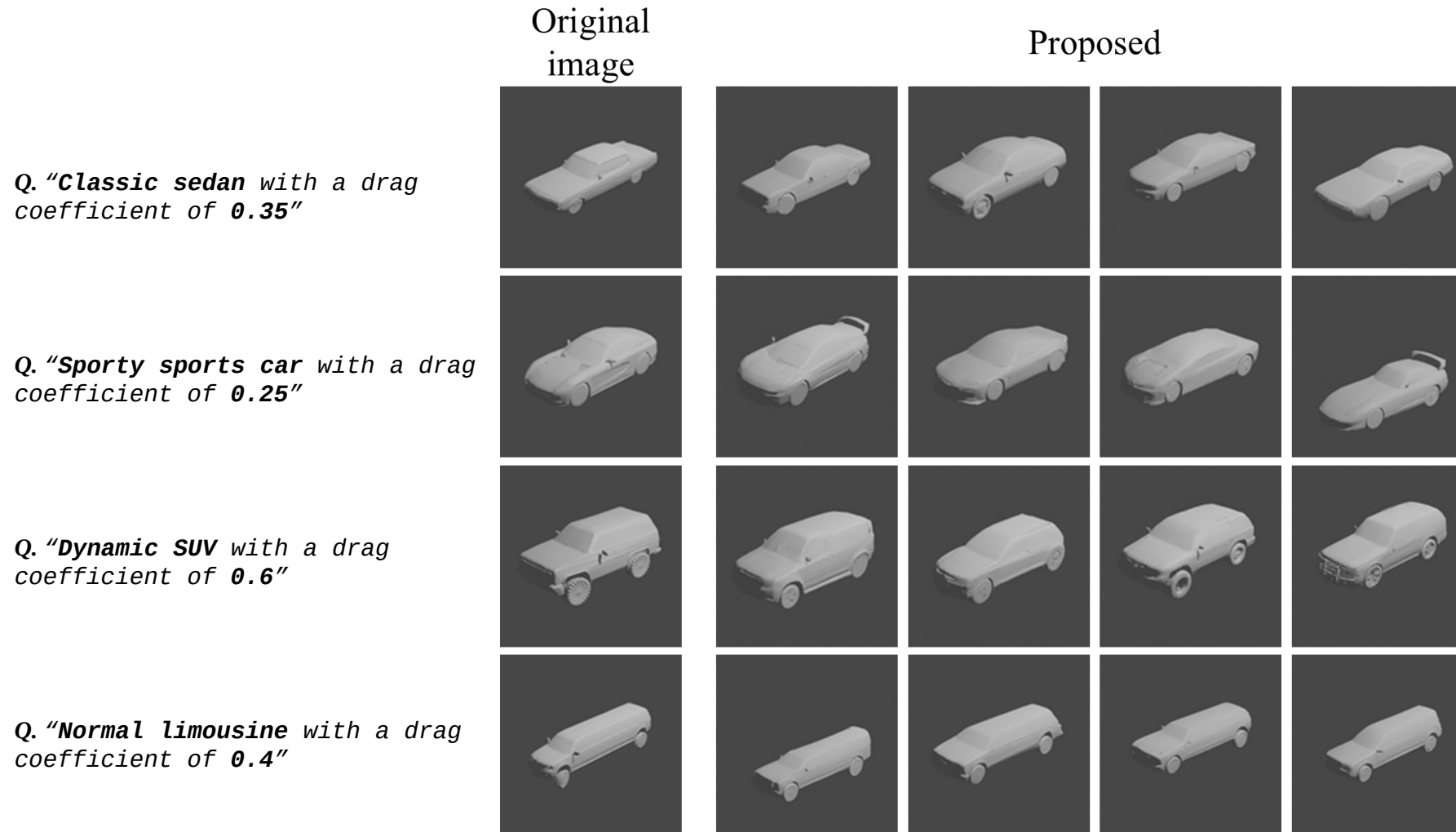
Generation Process



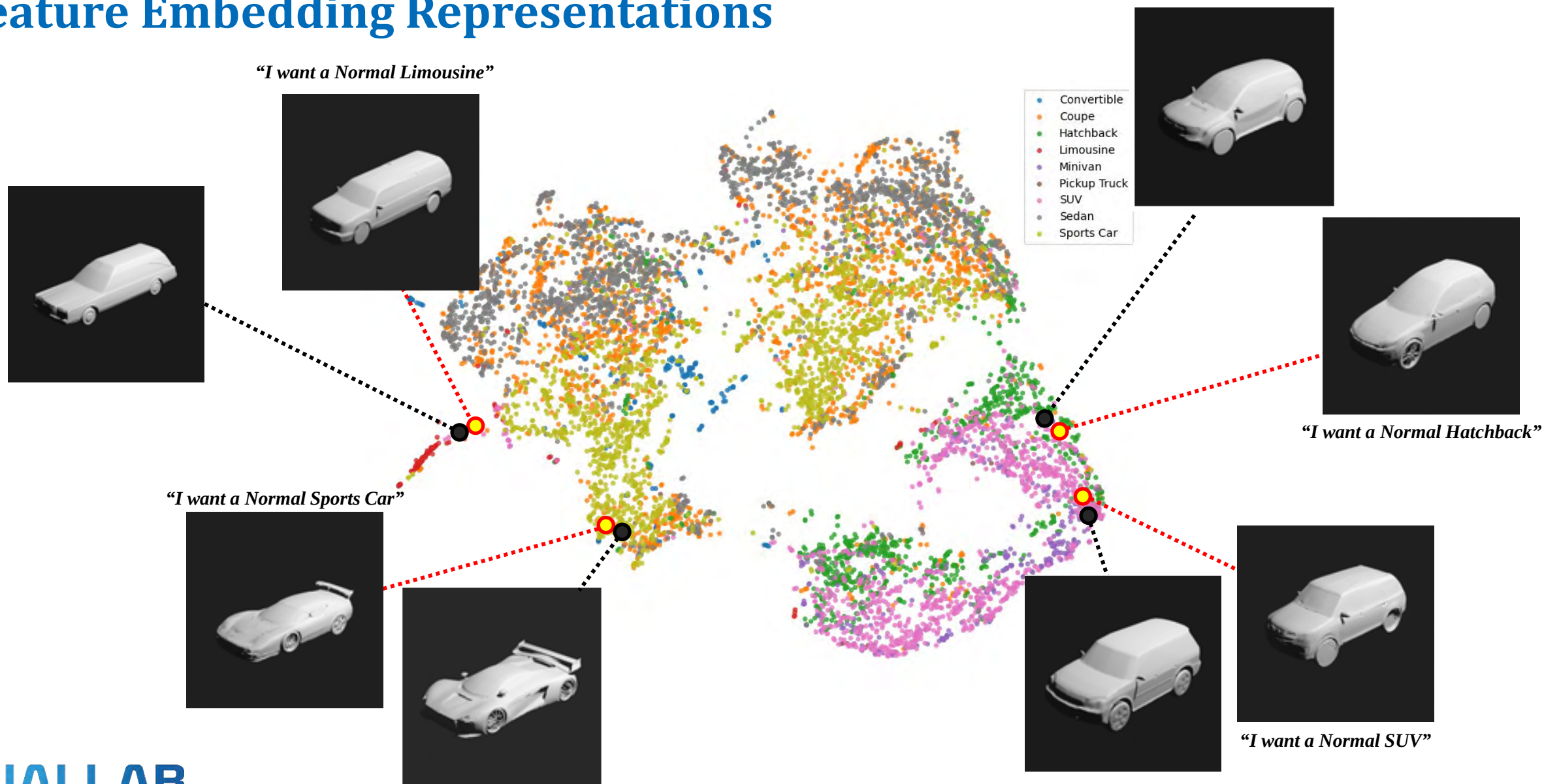
“I want a Sporty Sedan which Drag Coefficient is 0.4”



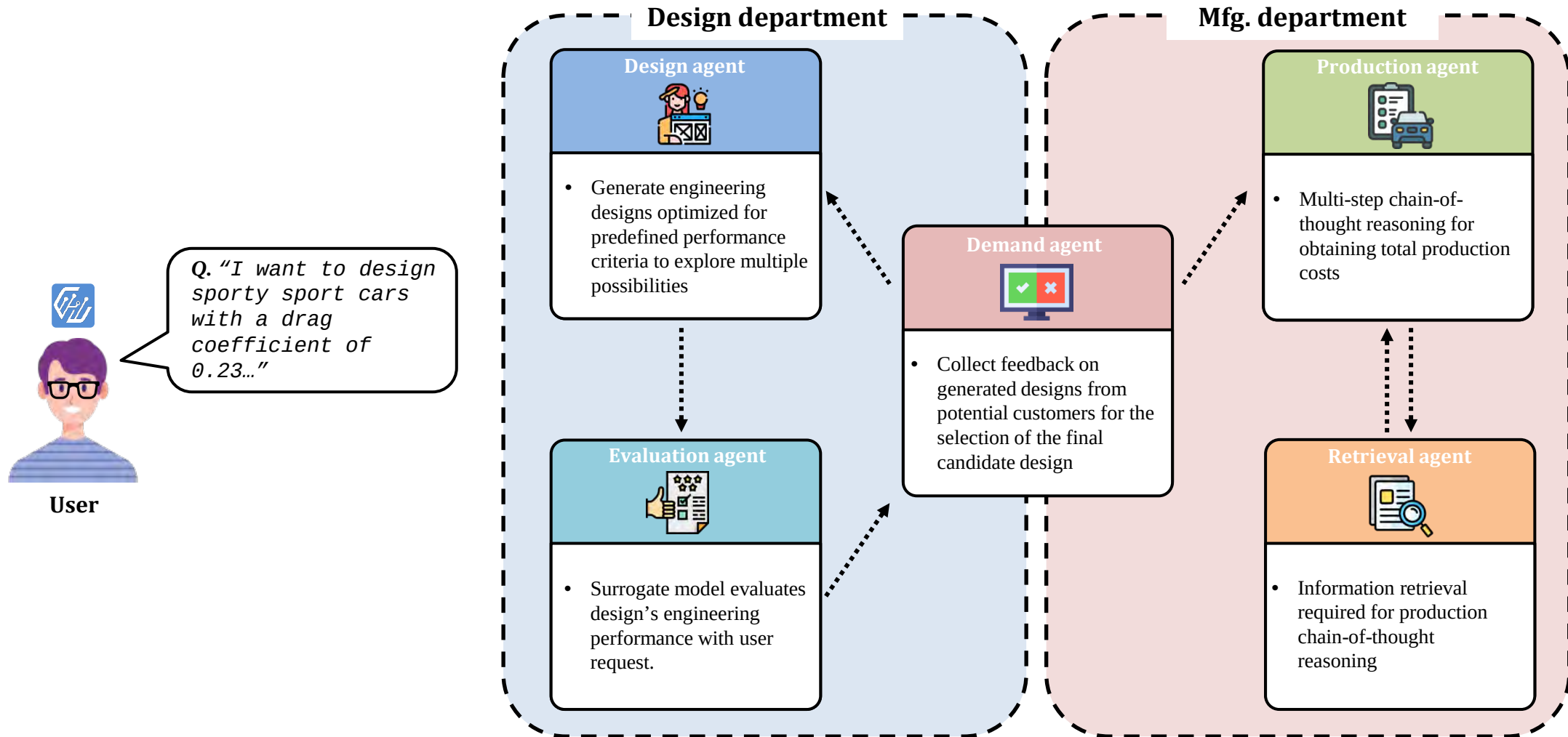
Prompt-guided Generative Designs



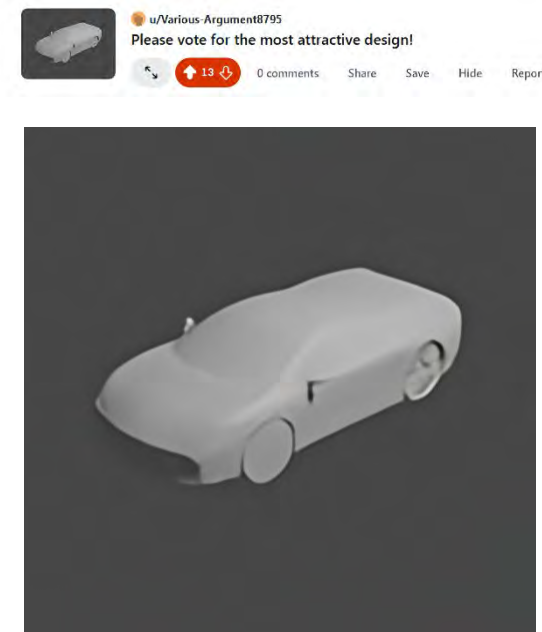
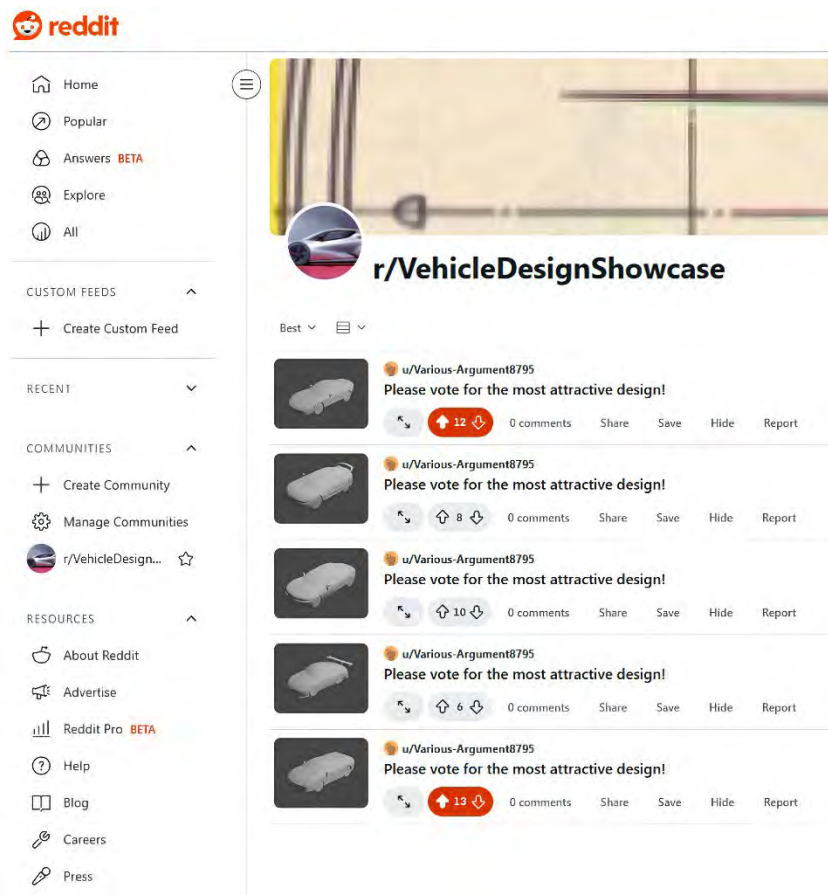
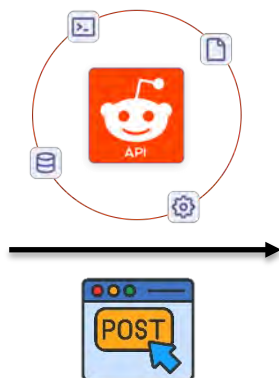
Feature Embedding Representations



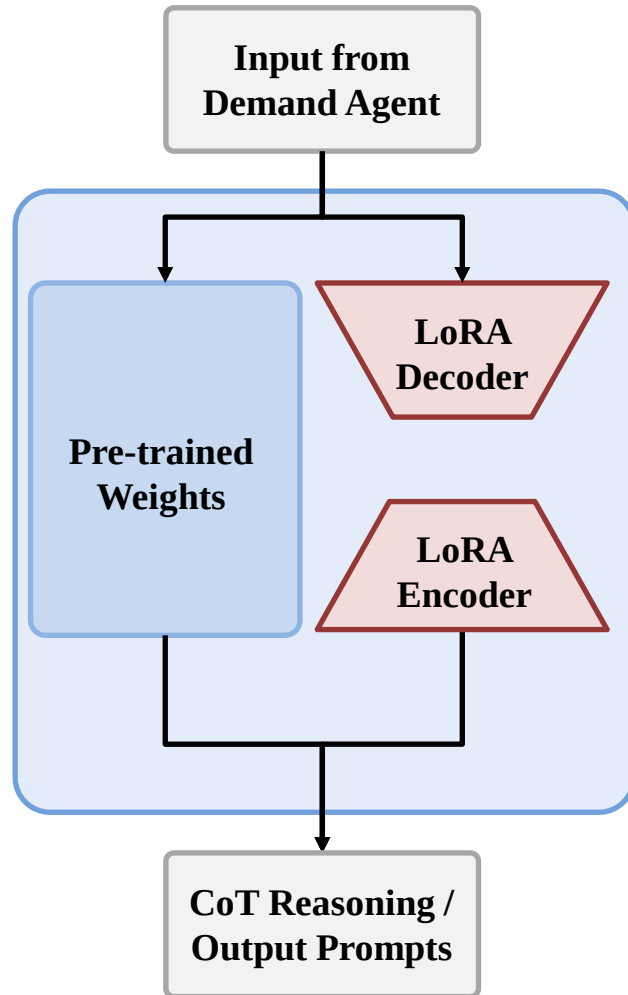
Towards Agentic AI: Multi-Agents for Design & Manufacturing



Demand Agent



Production Agent with Retrieval Agent



Q. Calculate the total production cost with labor cost, material cost, and manufacturing process cost for producing 400 vehicles per month."

<Total labor cost>

→ Step-by-step consideration of production yield, target production quantity, regular working hours, overtime pay, etc.

Step 1: Considering a 90% yield rate, to produce 400 vehicles, we need to calculate: $400 / 0.90 = 444.44$. Therefore, at least 445 vehicles must be produced.

Step 2: Since a Sportscar takes 20 hours to manufacture per vehicle, $445 \times 20 = 8900.00$ hours are required.

Step 3: Given a 5-day workweek, there are 4 weeks in a month, resulting in $5 \times 4 = 20.00$ working days per month.

Step 4: Required working hours per day are $8900.00 / 20.00 = 445.00$ hours/day.

Step 5: With 36 production lines, each line needs to operate for $445.00 / 36 = 12.36$ hours/day.

Step 6: Since the standard daily working hours are 8, additional overtime work of $12.36 - 8 = 4.36$ hours is required.

Step 7: Overtime pay is 20,000 KRW per hour; thus, overtime pay per person is $4.36 \times 20,000 = 87,200.00$ KRW.

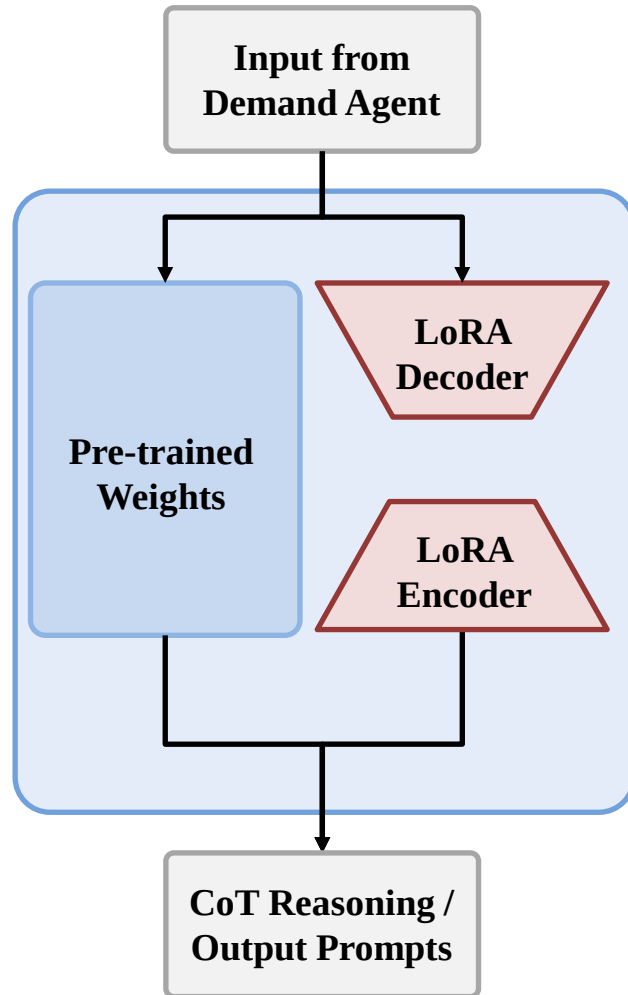
Step 8: Therefore, the daily wage per person is 150,000.00 KRW (basic pay) + 87,200.00 KRW (overtime) = 237,200.00 KRW.

Step 9: Working 20.00 days per month per person results in a monthly wage of $237,200.00 \times 20.00 = 4,744,000.00$ KRW per person.

Step 10: With 31 workers per line and 36 production lines, the total workforce is $31 \times 36 = 1116.00$ people.

Step 11: Thus, the total labor cost is $4,744,000.00 \times 1116.00 = 5,294,304,000.00$ KRW.

Production Agent with Retrieval Agent



<Total material cost>

→ Material cost per production unit, reflecting real-time raw material price (Search) and material ratio information from the internal database (RAG)

Step 12: Total material cost calculation for steel in KRW:
 $3233.00 \text{ (CNY/T)} \times 193.79 \text{ (KRW/CNY)} \times 0.001 \text{ (T/kg)} \times 1500.00 \text{ (kg)} \times 0.25 = 234,946.15 \text{ KRW.}$

Step 13: For CFRP:
 $15.00 \text{ (USD/kg)} \times 1390.67 \text{ (KRW/USD)} \times 1 \text{ (kg/kg)} \times 1500.00 \text{ (kg)} \times 0.30 = 9,387,022.50 \text{ KRW.}$

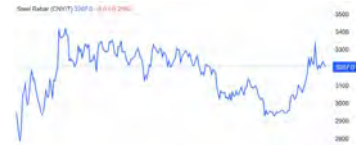
Step 14: For aluminum:
 $2571.70 \text{ (USD/T)} \times 1390.67 \text{ (KRW/USD)} \times 0.001 \text{ (T/kg)} \times 1500.00 \text{ (kg)} \times 0.40 = 2,145,831.62 \text{ KRW.}$

Step 15: For magnesium:
 $18100.00 \text{ (CNY/T)} \times 193.79 \text{ (KRW/CNY)} \times 0.001 \text{ (T/kg)} \times 1500.00 \text{ (kg)} \times 0.05 = 263,069.92 \text{ KRW.}$

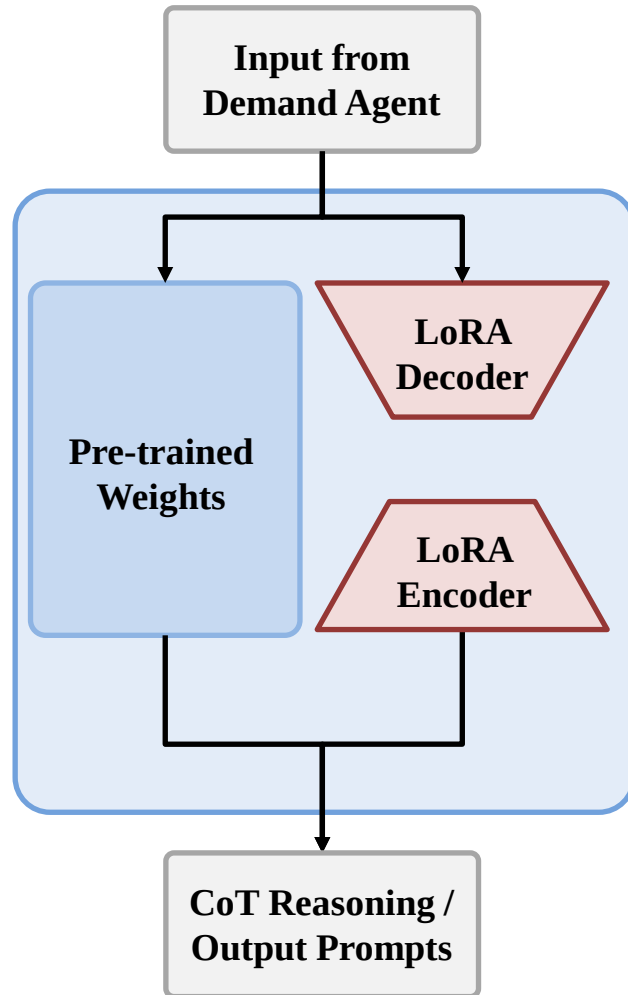
Step 16: Thus, the total material cost per vehicle is
 $234,946.15 + 9,387,022.50 + 2,145,831.62 + 263,069.92 = 12,030,870.20 \text{ KRW.}$

Step 17: Considering the yield rate of 90%, the total material cost for 445 vehicles is
 $12,030,870.20 \times 445.00 = 5,353,737,238.84 \text{ KRW.}$

Metals	Price
Gold	3397.28
Silver	38.335
Copper	4.4630
Steel	3207.00 ▼ 8.00
Lithium	71900 ▲ 800
Iron Ore CNY	772.50 ▲ 0.00
Platinum	1326.90 ▼ 23.30
HRC Steel	830.03 ▼ 7.97
Iron Ore	101.22 ▲ 0.01
Titanium	49.00 ▲ 0.00



Production Agent with Retrieval Agent



<Total manufacturing process cost>

→ Utilizing process ratios retrieved from the database via RAG,
calculate process costs based on production quantities adjusted for yield

Step 18: Manufacturing process cost calculation for Press process is $25,000,000.00 \times 0.18 = 4,500,000.00$ KRW.

Step 19: For the Body process, $30,000,000.00 \times 0.32 = 9,600,000.00$ KRW.

Step 20: For the Painting process, $40,000,000.00 \times 0.30 = 12,000,000.00$ KRW.

Step 21: For the Assembly process, $20,000,000.00 \times 0.20 = 4,000,000.00$ KRW.

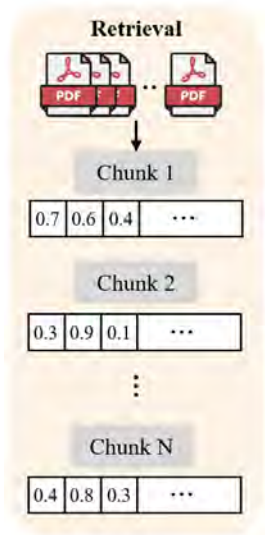
Step 22: Therefore, the expected total manufacturing process cost per vehicle is $4,500,000.00 + 9,600,000.00 + 12,000,000.00 + 4,000,000.00 = 30,100,000.00$ KRW.

Step 23: Considering the yield rate of 90%, the total manufacturing process cost for 445 vehicles is $30,100,000.00 \times 445 = 13,394,500,000.00$ KRW.

Step 24: In conclusion, the total production cost for 400 vehicles is:

$$\begin{aligned}
 \text{Total cost} &= \text{Labor cost} + \text{Material cost} + \text{Manufacturing process cost} \\
 &= 5,294,304,000.00 + 5,353,737,238.84 + 13,394,500,000.00 \\
 &= 24,042,541,238.84 \text{ KRW.}
 \end{aligned}$$

Step 25: Finally, the production cost per vehicle for manufacturing 400 vehicles/month is $24,042,541,238.84 \text{ KRW} / 400 = 60,106,353.10 \text{ KRW per vehicle.}$





Diffusion Model Summary

- Data generation by denoising
 - Gradually refines random noise (latent input) into high-quality images or audio
 - Allows for targeted modifications and edits
- Increasing attention for engineering applications
 - Generative designs
 - Neural augmentation
 - Virtual synthesis
 - Process optimization
 - ... *Many more potentials to discover !!*

감사합니다.