CS181: Computer Vision

Generative Modeling: Diffusion

November, 5th, 2026



Md Alimoor Reza
Associate Professor of Computer Science

Types of Generative Models

- Existing generative modeling techniques can be classified into four broad categories
  - Generative adversarial networks GAN (implicit generative models)
  - Variational auto-encoders VAE (likelihood-based models)
  - Diffusion models
  - Normalizing flows

Generative Model

- Generative modeling can be used on many downstream tasks and applications

- Image generation: generate new image after training from a given image dataset



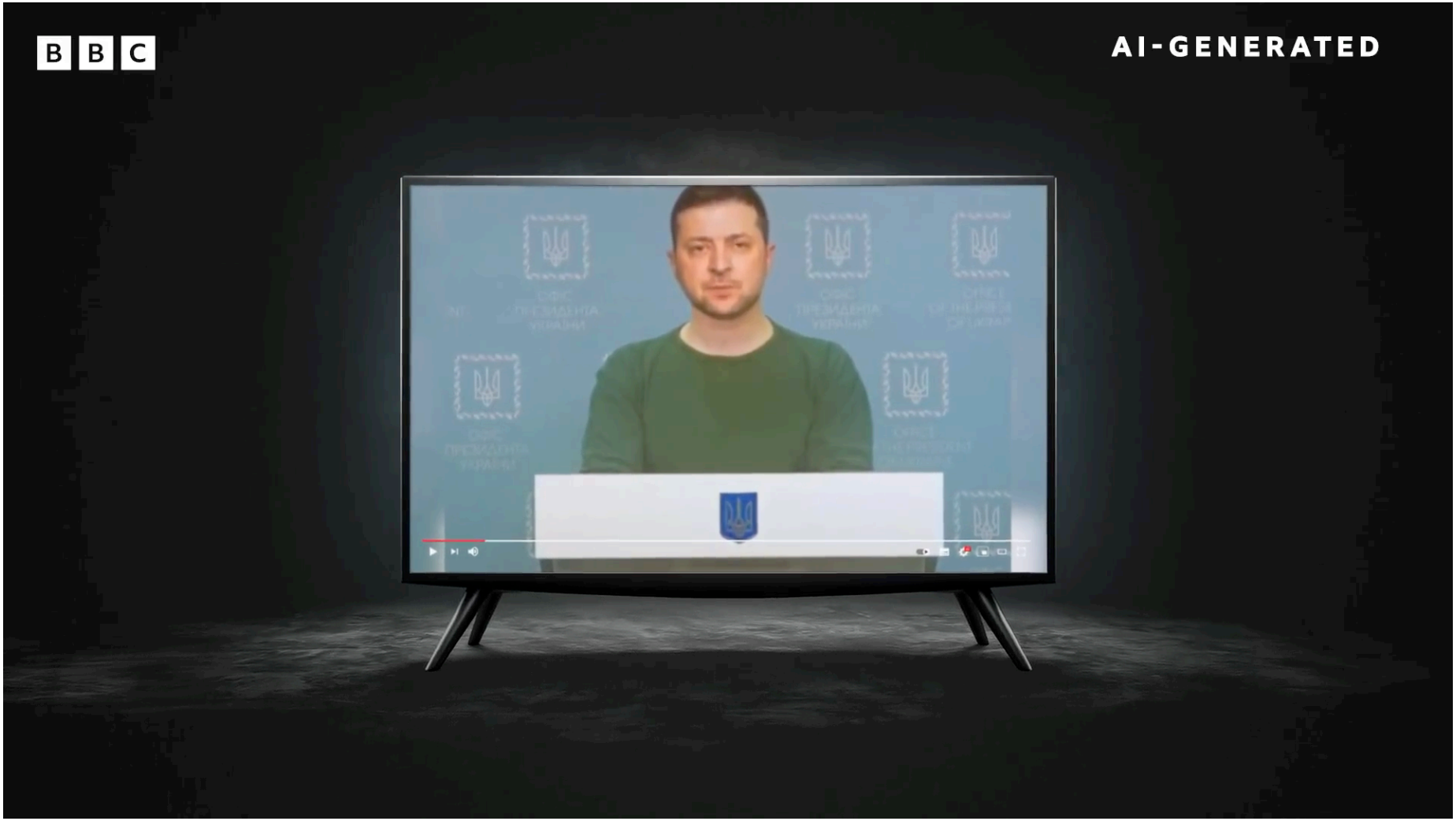
GAN-based



Diffusion-based

Diffusion Models Beat GANs on Image Synthesis - arxiv'21

Implications of Generative Models



Reference: The Rise of Deepfakes: How AI Changed Porn, Politics, and Reality - BBC World Service Documentaries

What a state-of-the-art AI Model sees?



I am attaching a photo. Explain what do you see in this image?

The image shows a **busy street and market area in Lhasa, Tibet**, with the famous **Potala Palace** dominating the hillside in the background.

Here are the main elements:

- **Potala Palace:** The large white-and-red structure on the mountain is the Potala Palace, historically the winter residence of the Dalai Lamas and one of Tibet's most iconic landmarks.
- **Mountainous landscape:** The palace sits against a dramatic, high-altitude mountain backdrop, characteristic of the Tibetan Plateau.
- **Traditional Tibetan architecture:** The foreground buildings have flat roofs, whitewashed walls, decorative window frames, and colorful Tibetan-style ornamentation.
- **Market stalls:** Numerous stalls are selling brightly colored textiles, clothing, scarves, and other goods. The colorful umbrellas and awnings make the lower part of the image particularly lively.
- **Religious/cultural elements:** You can see **Tibetan prayer flags** and other decorative religious elements around the buildings.
- **Chinese signage:** Several storefronts have Chinese-language signs, reflecting the location's contemporary urban environment.
- **People and street life:** The plaza is crowded with pedestrians and vendors, giving the scene the feel of a busy tourist and commercial district.

Overall, it appears to be a **street-level view of Lhasa with the Potala Palace in the background**, combining Tibetan religious heritage, traditional architecture, tourism, and everyday urban life.

Reference: What ChatGPT sees.

Change the style of an image (style transfer)

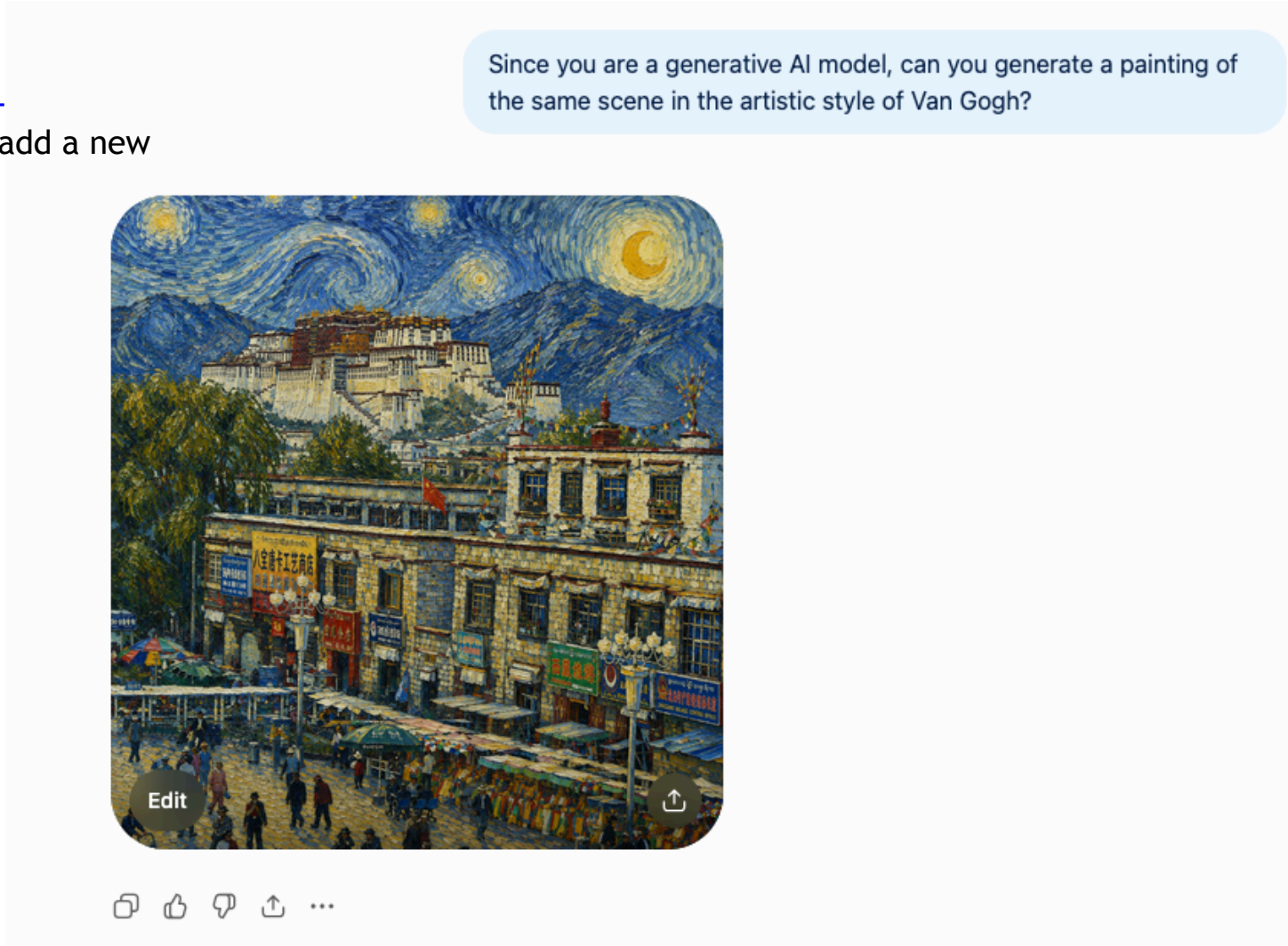
- GPT-Image Model:
  - Take an image and add a new style to it



Reference: Picture with an artistic sytel using a Generative-AI Model

Change the style of an image (style transfer)

- GPT-Image Model:
  - Take an image and add a new style to it



Reference: Picture with an artistic sytel using a Generative-AI Model

Generate a sequence of images (animation)

- GPT-Image-2.6 Model:
  - Generate a image sequence
  - Animation, graphics interchange format (GIF)
  - Comes with Codex, ChatGPT, and API

GPT-Image is not open source model. You can't modify or make your own.



Reference: Animation using a Generative-AI Model

Opensource PyTorch implementation: Try Pretrained Diffusion Models

- Try out some pretrained diffusion models (link also provided on Blackboard):

https://github.com/alimoorreza/CS181-fall26-notes/blob/main/day22_generative_model_diffusion.ipynb

a lot of open source diffusion models are hosted on Huggingface. You can modify or make your own.

Types of Generative Models

- Existing generative modeling techniques can be classified into four broad categories
  - Generative adversarial networks GAN (implicit generative models)
  - Variational auto-encoders VAE (likelihood-based models)
  - Diffusion models (eg, DDPM)
  - Normalizing flows

Diffusion Process

- Goal: learn structure of the probability density of a dataset



- Recover structure by reverse time; in other words, recover the data distribution by:
  - starting from uniform distribution and running backward dynamics



Diffusion Process

- **Forward process:** destroy all structure in data using diffusion process



- **Reverse process:** train a DNN model to reverse the diffusion process

U-Net



Diffusion Process

- **Forward process:** converts any complex data distribution into a tractable distribution



- **Reverse process:** learn a finite-time reversal of this diffusion process into generative model



Denoising Diffusion Probabilistic Model (DDPM)

- **During training,** diffusion model is trained in semi-supervised manner as follows (pseudocode):
  - **Step#1:** Take an image from the training set
  - **Step#2:** Apply to it some random noise T times (eg, T=1000. Basically generate training samples in a semi-supervised manner)

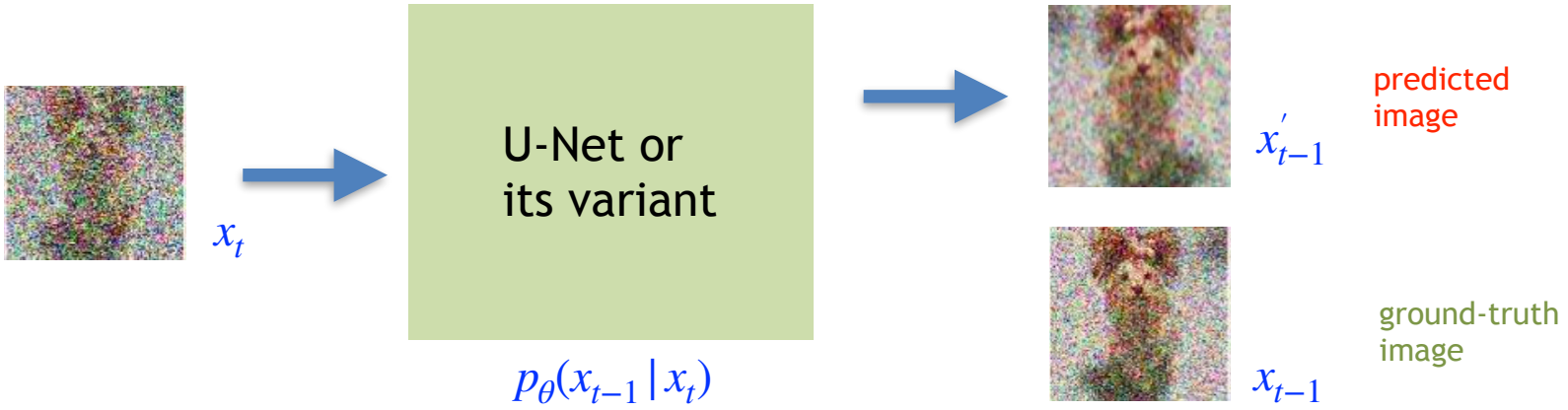


  - **Step#3:** Give this noisy image x_t to the model (eg, U-Net) along with the value of t
  - **Step#4:** Compute a loss from the output of the model (eg, U-Net) and the noised image x_{t-1}

Denoising Diffusion Probabilistic Model (DDPM)

- **During training,** diffusion model is trained in semi-supervised manner as follows:

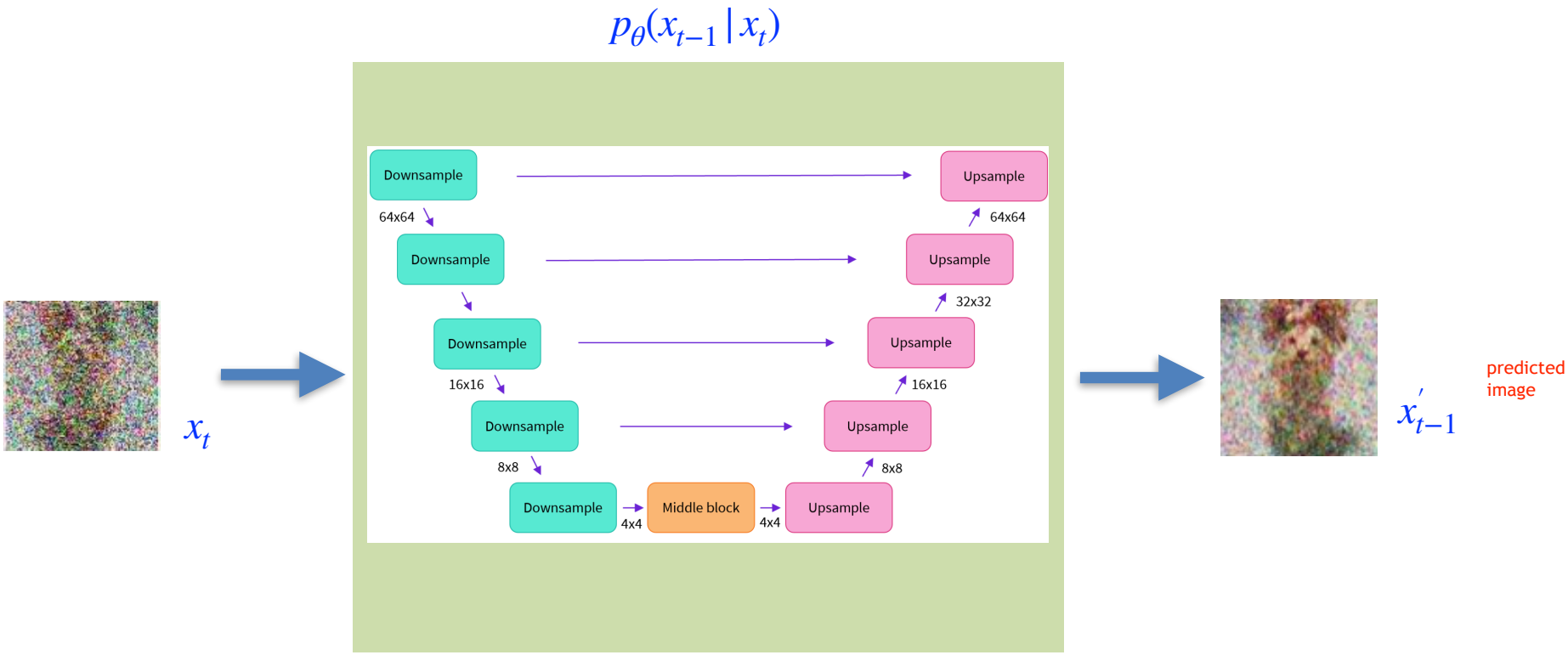
  - **Step#4:** Compute a loss from the output of the model (eg, U-Net) and the noised image x_{t-1}



In other words, the model is trying to predict the noises we added in the semi-supervised way in a sequence (or chain) of ***t*** steps

Denoising Diffusion Probabilistic Model (DDPM)

- **U-Net model:**
  - To compute a loss from the output of the model (eg, U-Net) and the noised image during training
  - To predict the next image in the chain from the given current noisy image during testing



Denoising Diffusion Probabilistic Model (DDPM)

- **During training,** diffusion model is trained in semi-supervised manner as follows (PyTorch implementation):

```
for epoch in range(config.num_epochs):
    progress_bar = tqdm(total=len(train_dataloader), disable=not accelerator.is_local_main_process)
    progress_bar.set_description(f"Epoch {epoch}")

    for step, batch in enumerate(train_dataloader):
        clean_images = batch['images']

        # Sample noise to add to the images
        noise = torch.randn(clean_images.shape).to(clean_images.device)
        bs = clean_images.shape[0]

        # Sample a random timestep for each image (eg, between 0 and 1000)
        timesteps = torch.randint(0, noise_scheduler.num_train_timesteps, (bs,), device=clean_images.device).long(

        # Add noise to the clean images according to the noise magnitude at each timestep
        # (this is the forward diffusion process)
        noisy_images = noise_scheduler.add_noise(clean_images, noise, timesteps)

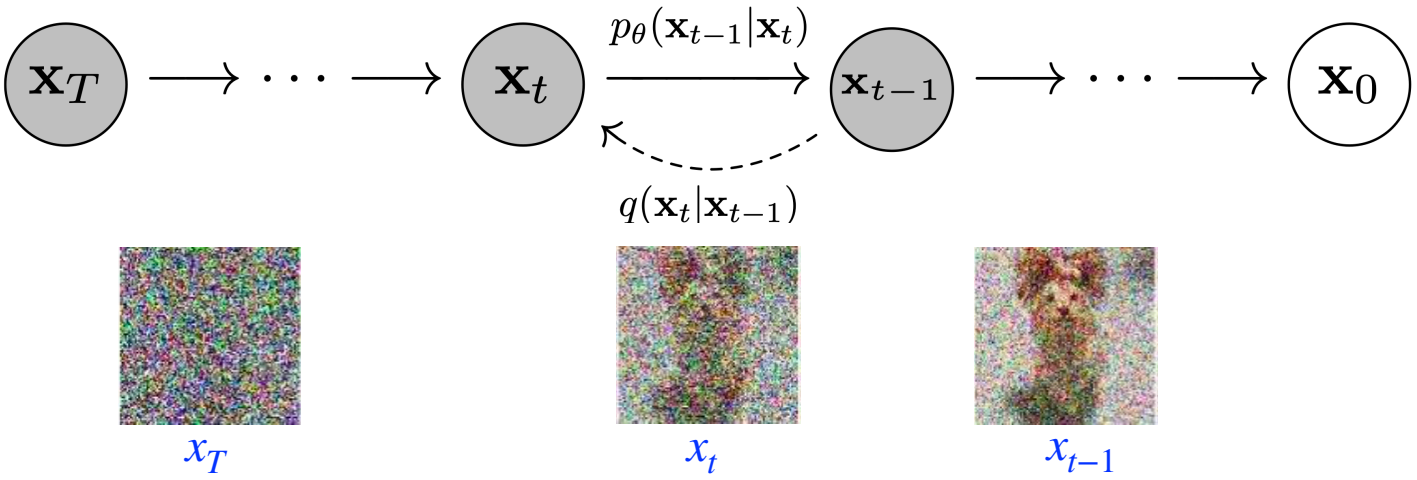
        with accelerator.accumulate(model):
            # Predict the noise residual
            noise_pred = model(noisy_images, timesteps, return_dict=False)[0]
            loss = F.mse_loss(noise_pred, noise)
            accelerator.backward(loss)

            accelerator.clip_grad_norm_(model.parameters(), 1.0)
            optimizer.step()
            lr_scheduler.step()
            optimizer.zero_grad()
```

In DDPM, the model predicts the noises we added in the semi-supervised way in a sequence (or chain) of ***t*** steps

Denoising Diffusion Probabilistic Model (DDPM)

- **During testing,** diffusion model can be used to iteratively transform a random noise to generate an image (running the reverse process only)



DDPM: Forward Encoder

- Suppose we take an image x (from a training set) and blend it with a Gaussian **'noise'** independently for each pixel to give noise-corrupted image z_1

$$z_1 = \sqrt{1-\beta_1}x + \sqrt{\beta_1}\epsilon_1$$

Gaussian noise distribution: $\epsilon_1 \sim \mathcal{N}(\epsilon_1 | 0, I)$

Variance of the noise distribution: β_1

The choice of $\sqrt{1-\beta_1}$, and $\sqrt{\beta_1}$ ensures that the mean of the distribution of z_t is closer to zero than z_{t-1}. Also the variance of z_t is closer to unit matrix than the variance of z_{t-1}

DDPM: Forward Encoder

- The transformation in previous slide can be written as Gaussian distribution of the following form:

$$q(z_1 \mid x) = \mathcal{N}(z_1 \mid \sqrt{1-\beta_1}x, \beta_1 I)$$

- We then repeat the process with additional independent Gaussian noise steps to give a sequence of increasingly noisy images:

$$z_2, z_{t-1}, z_t, \ldots, z_T$$

- You can assume these are the latent variables in our diffusion generative model

DDPM: Forward Encoder

- Each successive $z_2, z_{t-1}, z_t, \ldots, z_T$ image is given as follows:

$$z_t = \sqrt{1-\beta_t} z_{t-1} + \sqrt{\beta_t}\epsilon_t$$

Gaussian noise distribution: $\epsilon_t \sim \mathcal{N}(\epsilon_t | 0, I)$

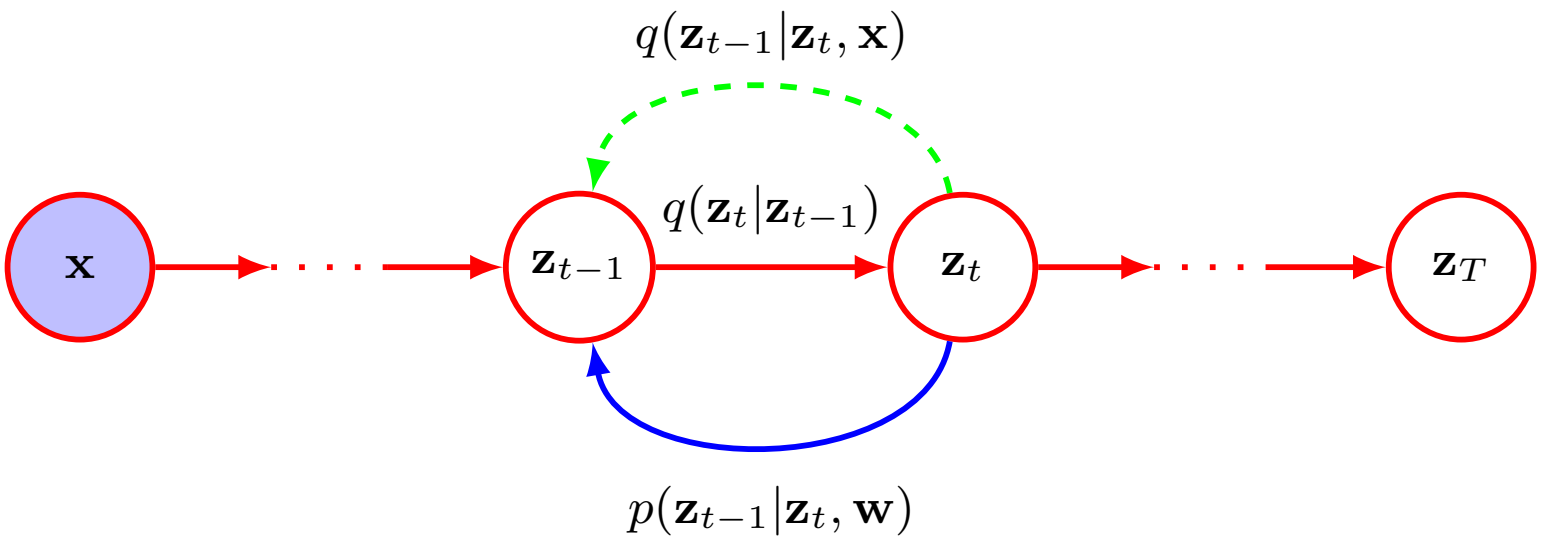
Variance of the noise distribution: β_t

- The above transformation can be written as Gaussian distribution of the following form:

$$q(z_t | z_{t-1}) = \mathcal{N}(z_t | \sqrt{1-\beta_t} z_{t-1}, \beta_t I)$$

DDPM: Forward Encoder

- The sequence of conditional distributions forms a Markov chain and can be expressed as probabilistic graphical model as shown below:



PyTorch Implementation: Diffusion Model

- DDPM diffusion model: training + inference

https://github.com/alimoorreza/CS181-fall26-notes/blob/main/day22_diffusers_training_example.ipynb

PyTorch Implementation: Diffusion Model

- Other generative model training/testing from Diffuser:

https://github.com/huggingface/diffusers/tree/main/examples

Task	Accelerate	Datasets	Implementation
Unconditional Image Generation	✅	✅	Open In Colab
Text-to-Image fine-tuning	✅	✅	
Textual Inversion	✅	-	Open In Colab
Dreambooth	✅	-	Open In Colab
ControlNet	✅	✅	Notebook
InstructPix2Pix	✅	✅	Notebook
Reinforcement Learning for Control	-	-	Notebook (diffusion-based policy learning) Notebook (Diffusion-based planning)