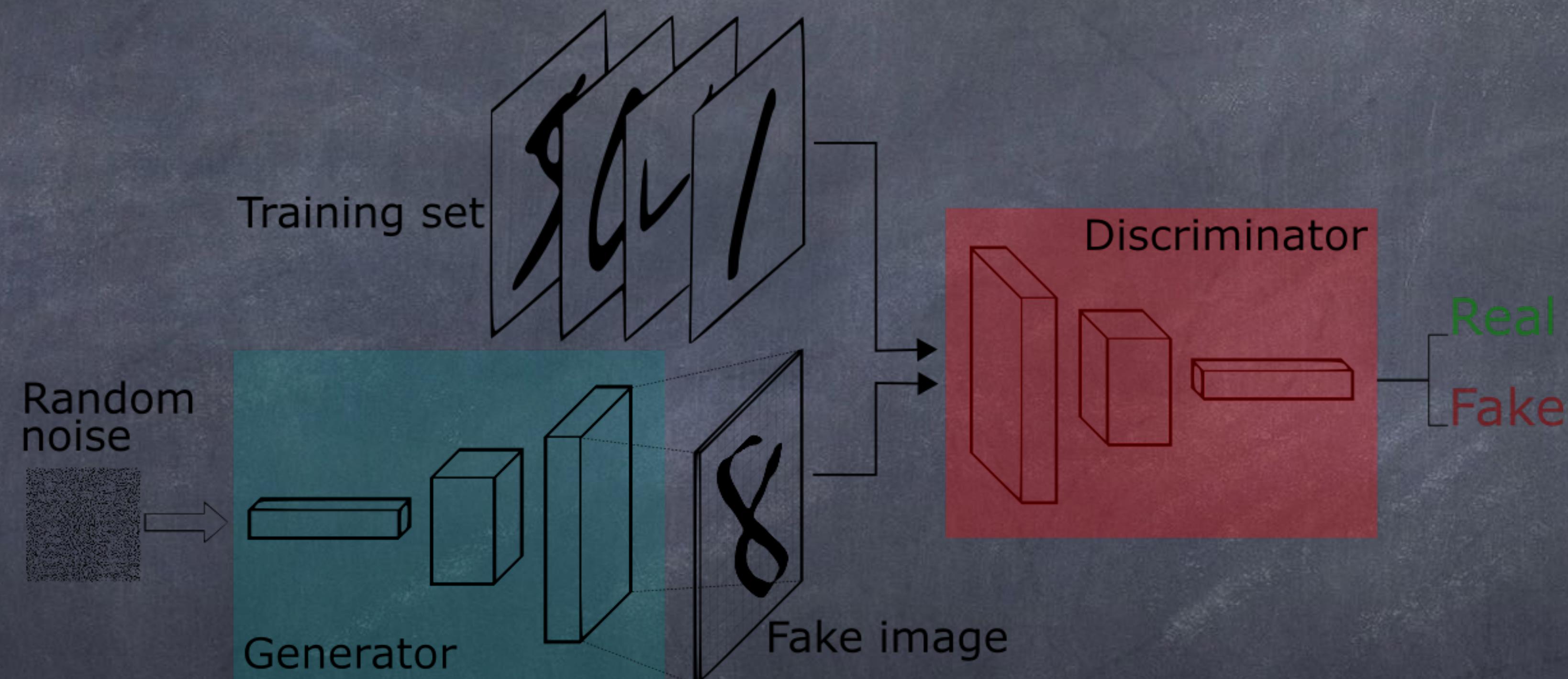


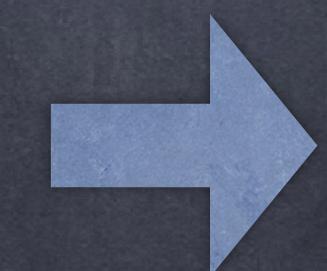
Image Generation

Sadeep Jayasumana

Generative Adversarial Networks (GANs)



Generative Adversarial Networks (GANs)



FAKE!



Real



Real or Fake?



Generative Adversarial Networks (GANs)

for number of training iterations **do**

for k steps **do**

- Sample minibatch of m noise samples $\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}\}$ from noise prior $p_g(\mathbf{z})$.
- Sample minibatch of m examples $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\mathbf{x})$.
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^m \left[\log D(\mathbf{x}^{(i)}) + \log (1 - D(G(\mathbf{z}^{(i)}))) \right].$$

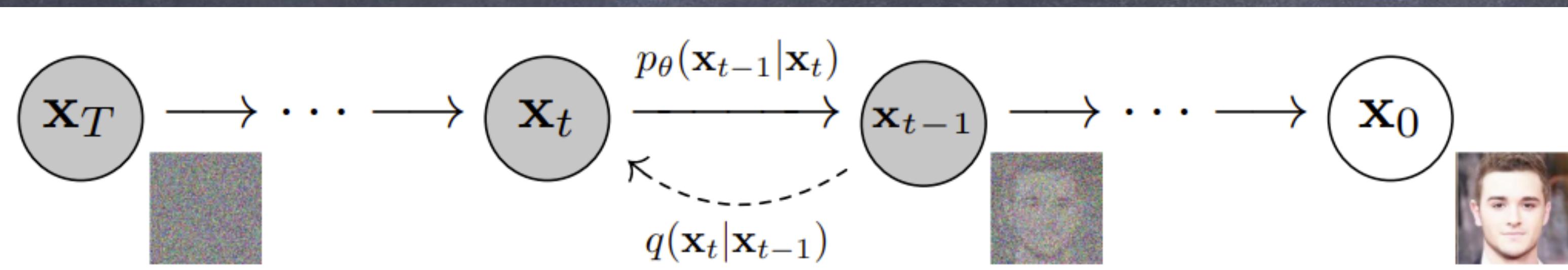
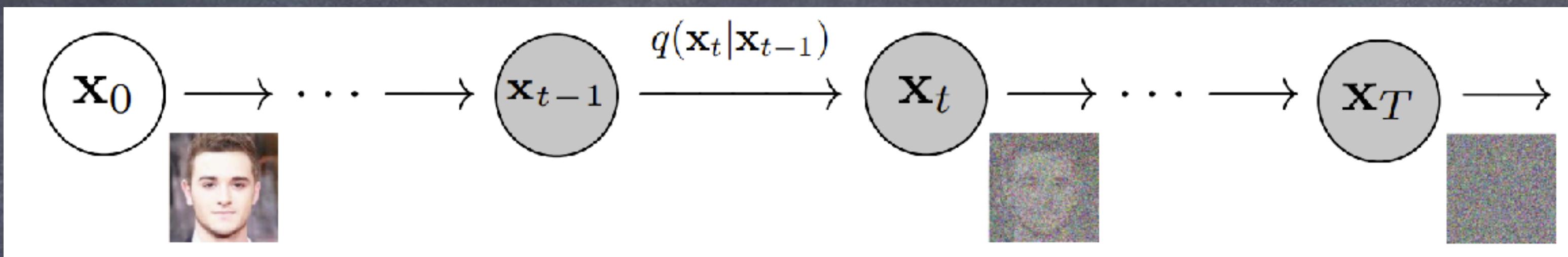
end for

- Sample minibatch of m noise samples $\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}\}$ from noise prior $p_g(\mathbf{z})$.
- Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^m \log (1 - D(G(\mathbf{z}^{(i)}))).$$

end for

Diffusion Models



Diffusion Models

$$q(x_t | x_{t-1}) = \mathcal{N}(x_t; \sqrt{1 - \beta_t} x_{t-1}, \beta_t I)$$

With $\beta_1 < \beta_2 < \dots < \beta_T$

$$\alpha_t := 1 - \beta_t$$

$$\bar{\alpha}_t := \prod_{s=1}^T \alpha_s$$

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

- When β_t values are small, the reverse process is also Gaussian

$$p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t)) \text{ for } 1 < t \leq T.$$

$$\boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t) = \sigma_t^2 \mathbf{I}$$

Diffusion Models

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

```

Algorithm 2 Sampling

```

1:  $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ 
2: for  $t = T, \dots, 1$  do
3:    $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$  if  $t > 1$ , else  $\mathbf{z} = \mathbf{0}$ 
4:    $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left( \mathbf{x}_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \epsilon_{\theta}(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$ 
5: end for
6: return  $\mathbf{x}_0$ 

```

```

rng = 0

def train_step(batch):
    rng, tsrng = np.random.randint(0, 100000, size=(2,))
    timestep_values = generate_timestamp(tsrng, batch.shape[0])

    noised_image, noise = forward_noise(rng, batch, timestep_values)
    with tf.GradientTape() as tape:
        prediction = unet(noised_image, timestep_values)

        loss_value = loss_fn(noise, prediction)

    gradients = tape.gradient(loss_value, unet.trainable_variables)
    opt.apply_gradients(zip(gradients, unet.trainable_variables))

    return loss_value

```

```

x = tf.random.normal((1,32,32,1))

for i in tqdm(range(timesteps-1)):
    t = np.expand_dims(np.array(timesteps-i-1, np.int32), 0)
    pred_noise = unet(x, t)
    x = ddpm(x, pred_noise, t)

def ddpm(x_t, pred_noise, t):
    alpha_t = np.take(alpha, t)
    alpha_t_bar = np.take(alpha_bar, t)

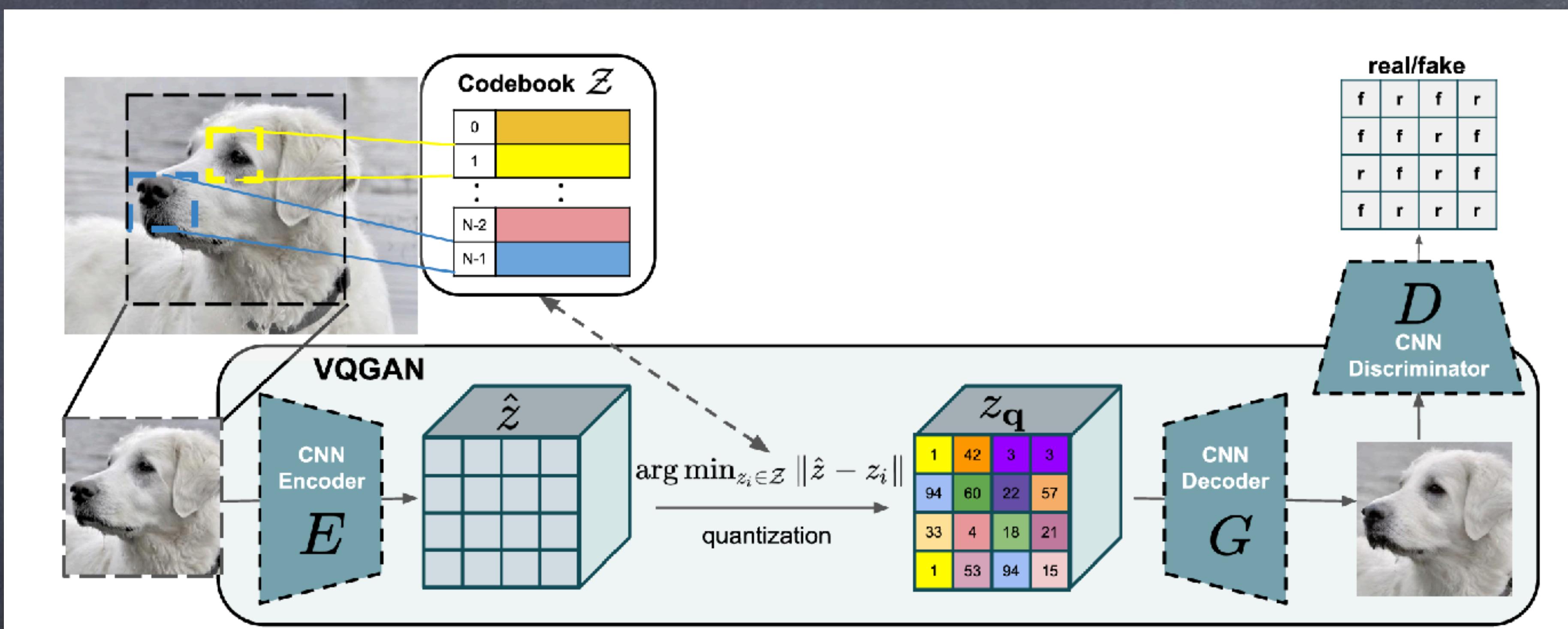
    eps_coef = (1 - alpha_t) / (1 - alpha_t_bar) ** .5
    mean = (1 / (alpha_t ** .5)) * (x_t - eps_coef * pred_noise)

    var = np.take(beta, t)
    z = np.random.normal(size=x_t.shape)

    return mean + (var ** .5) * z

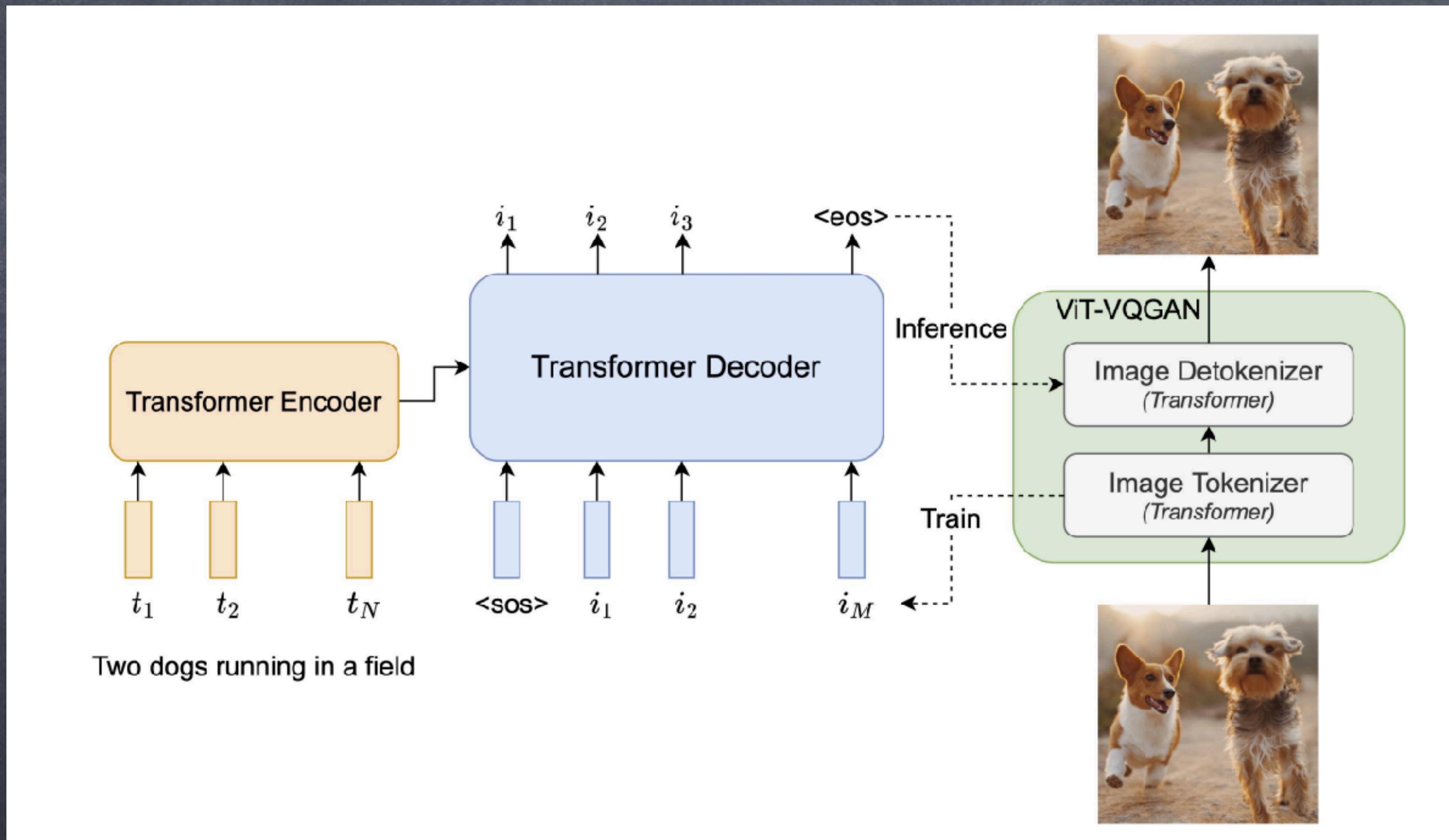
```

VQGAN



Reference: Taming Transformers for High-Resolution Image Synthesis, 2020

Autoregressive Models



Reference: Scaling Autoregressive Models for Content-Rich Text-to-Image Generation, 2022

Latent Diffusion Models (LDMs)

- Diffusion in the (continuous) latent space.
- State-of-the-art in 2024 based on the publicly available information.
- StableDiffusion - An open-source version of LDMS

Off-topic: Distillation

- Useful when inference time resources are limited.
- You first learn how to solve the problem with a big, teacher model since learning with bigger models is generally easier.
- Then you train a smaller student model to mimic the teacher.
- The student has access to “dark knowledge”.

$$D_{\text{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right)$$



Thank You!