Deep Learning lecture 11 Structures in Deep Learning

Yi Wu, IIIS

Spring 2025

Apr-28

Today's Topic

- Structured Priors in Deep Generative Model
  - Discrete/structured latent variables
- Deep Learning for Structured Data
  - Graph neural networks
- Applications

Today's Topic

- Structured Priors in Deep Generative Model
  - Discrete/structured latent variables
- Deep Learning for Structured Data
  - Graph neural networks
- Applications

Recap: Generative Model

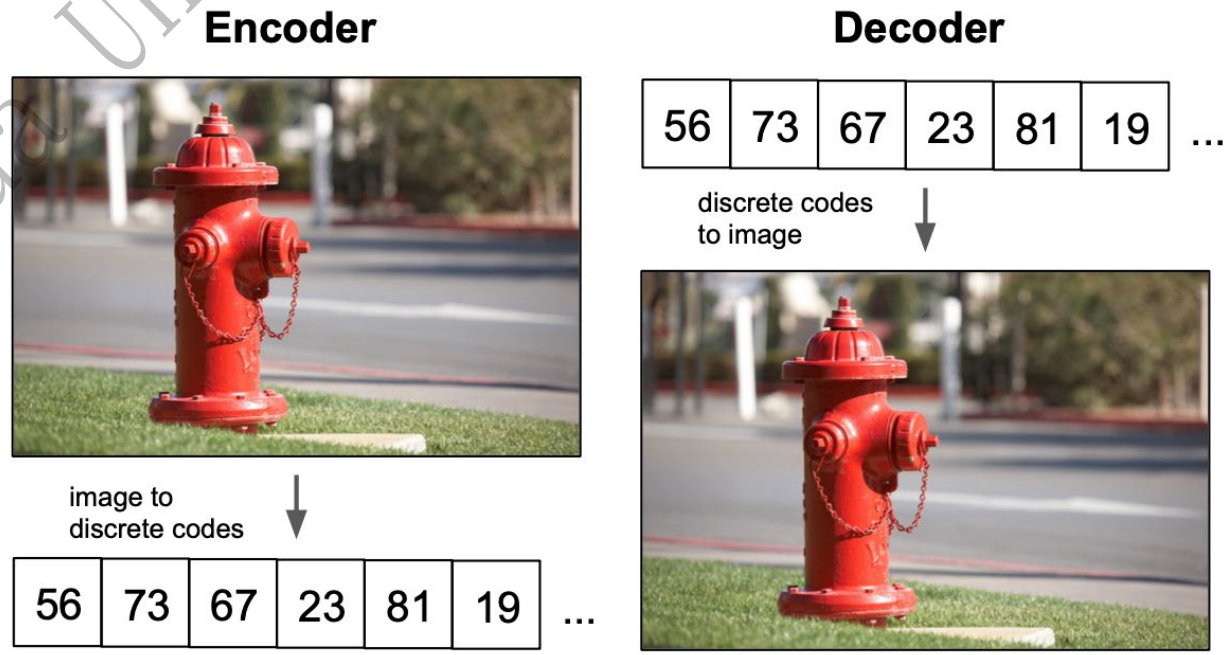
- Principle in Generative Model (Lec 4~7)
  - Prior: $z \sim p(z)$
    - Typically Gaussian prior $z \sim N(0, I)$
  - Non-linear transformation: $x = f(z; \theta)$ or $x \sim p(x|z; \theta)$
  - Learning θ
    - MLE (flow); ELBO (VAE); Classification (GAN)
- Issues with Gaussian prior
  - Mode collapse issue ($N(0, I)$ is uni-modal)
  - Blurry samples (interpolation between samples)
  - Tricky variance constraints (β in VAE learning, truncation in GAN)

Recap: Generative Model

- Principle in Generative Model (Lec 4~7)
  - Prior: $z \sim p(z)$
    - Typically Gaussian prior $z \sim N(0, I)$
  - Non-linear transformation: $x = f(z; \theta)$ or $x \sim p(x|z; \theta)$
  - Learning θ
    - MLE (flow); ELBO (VAE); Classification (GAN)
- Issues with Gaussian prior
  - Mode collapse issue ($N(0, I)$ is uni-modal)
  - Blurry samples (interpolation between samples)
  - Tricky variance constraints (β in VAE learning, truncation in GAN)
- More powerful/expressive $p(z)$?

Enhanced Latent Variable

- Goal: more powerful $p(z)$ than $N(0, I)$
- A simple enhancement: discrete latent variable z
  - $z_i \sim \text{Unif}(0,1,2,\dots,K-1)$
  - Pros
    - Naturally multi-modal
    - No tricky variance of Gaussian
- Discrete latent variable model
  - $z_i \sim p(z_i) = \text{Unif}(0,1,2,\dots,K-1)$
  - $x \sim p(x|z;\theta)$
- *Learning?*



VAE with Discrete Latent

- VAE with Gaussian Prior (Recap)
  - Decoder: $p(x|z;\theta)$; Encoder:$q(z|x;\phi) = N(\mu(x;\phi), \sigma(x;\phi))$
  - $J(\theta,\phi) = E_{z\sim q(z|x;\phi)}[\log p(x|z;\theta)] - KL(q(z|x;\phi)||N(0,I))$ (ELBO)
  - $= E_{\epsilon\sim N(0,I)}[\log p(x|\mu(x;\phi) + \sigma(x;\phi)\cdot\epsilon;\theta)] - KL(q(z|x;\phi)||N(0,I))$

VAE with Discrete Latent

- VAE with Gaussian Prior (Recap)
  - Decoder: $p(x|z;\theta)$; Encoder:$q(z|x;\phi) = N(\mu(x;\phi), \sigma(x;\phi))$
  - $J(\theta,\phi) = E_{z\sim q(z|x;\phi)}[\log p(x|z;\theta)] - KL(q(z|x;\phi)||N(0,I))$ (ELBO)
  - $= E_{\epsilon\sim N(0,I)}[\log p(x|\mu(x;\phi) + \sigma(x;\phi)\cdot\epsilon;\theta)] - KL(q(z|x;\phi)||N(0,I))$
    - Reparameterization trick for ***Gaussian*** variable
  - In the follow content, we use $f(x,z;\theta)$ to denote the loss induced by $\log p(x|z;\theta)$

VAE with Discrete Latent

- VAE with Gaussian Prior (Recap)
  - Decoder: $p(x|z;\theta)$; Encoder:$q(z|x;\phi) = N(\mu(x;\phi), \sigma(x;\phi))$
  - $J(\theta,\phi) = E_{\epsilon\sim N(0,I)}[f(x, \epsilon\cdot\sigma(x;\phi) + \mu(x;\phi);\theta)] - KL(q||p)$
- Discrete Latent Variable z .

VAE with Discrete Latent

- VAE with Gaussian Prior (Recap)
  - Decoder: $p(x|z;\theta)$; Encoder:$q(z|x;\phi) = N(\mu(x;\phi), \sigma(x;\phi))$
  - $J(\theta,\phi) = E_{\epsilon\sim N(0,I)}[f(x, \epsilon\cdot\sigma(x;\phi)+\mu(x;\phi);\theta)] - KL(q||p)$
- Discrete Latent Variable z
  - Encoder $q_i(z_i|x;\phi) = \text{softmax}(l_i(x;\phi))$
  - $J(\theta,\phi) = E_{z\sim q(x;\phi)}[f(x,z;\theta)] - KL(q||\text{unifom}(0,K-1))$

VAE with Discrete Latent

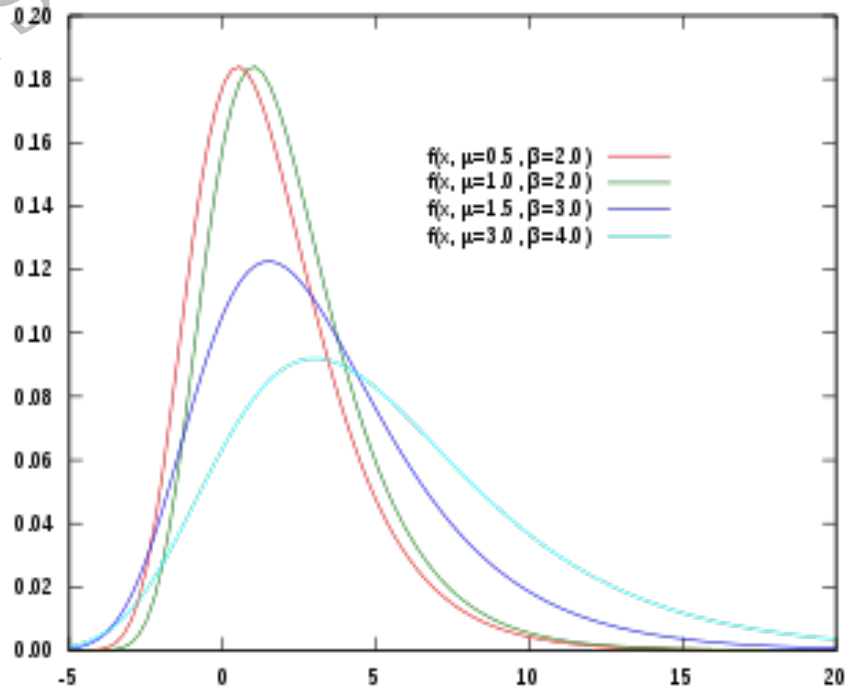
- VAE with Gaussian Prior (Recap)
  - Decoder: $p(x|z;\theta)$; Encoder:$q(z|x;\phi) = N(\mu(x;\phi), \sigma(x;\phi))$
  - $J(\theta,\phi) = E_{\epsilon\sim N(0,I)}[f(x, \epsilon\cdot\sigma(x;\phi)+\mu(x;\phi);\theta)] - KL(q||p)$
- Discrete Latent Variable z
  - Encoder $q_i(z_i|x;\phi) = \text{softmax}(l_i(x;\phi))$
  - $J(\theta,\phi) = E_{z\sim q(x;\phi)}[f(x, z;\theta)] - KL(q||\text{unifom}(0, K-1))$
  - Reparameterization trick?
    - Remark:
      - we have learned marginalization trick for semi-supervised VAE.
      - But here the latent dimension can be very high

VAE with Discrete Latent

- VAE with Gaussian Prior (Recap)
  - Decoder: $p(x|z;\theta)$; Encoder:$q(z|x;\phi) = N(\mu(x;\phi), \sigma(x;\phi))$
  - $J(\theta,\phi) = E_{\epsilon\sim N(0,I)}[f(x, \epsilon\cdot\sigma(x;\phi) + \mu(x;\phi);\theta)] - KL(q||p)$
- Discrete Latent Variable z
  - Encoder $q_i(z_i|x;\phi) = \text{softmax}(l_i(x;\phi))$
  - $J(\theta,\phi) = E_{z\sim q(x;\phi)}[f(x, z;\theta)] - KL(q||\text{unifom}(0, K-1))$
  - Reparameterization trick?
    - We need a trick for categorical distribution!
    - Current format: $z_i \sim q_i = \text{Categorical}(\exp(l_i))$
    - Desired format for backprop: $z_i = g_q(l_i + \epsilon)$ & $\epsilon\sim p^*$
      - ***How to choose p^*?***

Gumbel-Softmax Distribution

- Gumbel Distribution $\text{Gumbel}(\mu, \beta)$ $(\beta > 0)$
  - $P(x) = \frac{1}{\beta} e^{-(z+e^{-z})}$ where $z = \frac{x-\mu}{\beta}$
  - CDF $F(x; \mu, \beta) = e^{-e^{-(x-\mu)/\beta}}$
    - Also called *Generalized Extreme Value Distribution Type-I*
  - Generating Gumbel Samples
    - $Q(\mu, \beta) = \mu - \beta \log(-\log(U))$, where $U \sim \text{unifom}(0,1)$
  - Reparameterization trick
    - $u \sim \text{Uniform}(0,1)$
    - $\epsilon = -\log(-\log(u)) = Q(0,1)$
    - $X \sim \text{Gumbel}(\mu)$
    - $X = \mu + \beta \cdot \epsilon$



Gumbel-Softmax Distribution

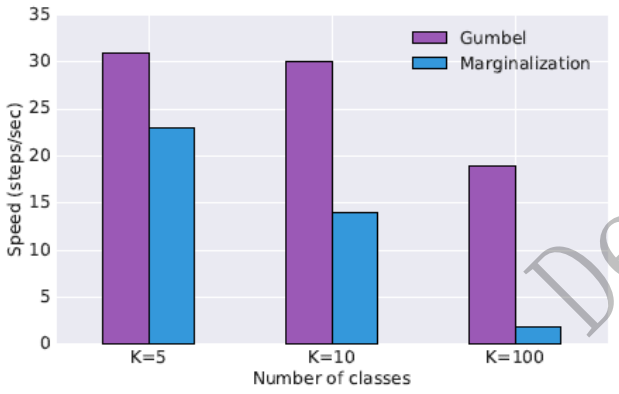
- Gumbel Distribution $\mathrm{Gumbel}(\mu, \beta)$ $(\beta > 0)$
  - Generating Gumbel Samples
    - $Q(\mu, \beta) = \mu - \beta \log(-\log(U))$, where $U \sim \mathrm{unifom}(0,1)$
    - $Q(\mu, \beta) \sim \mathrm{Gumbel}(\mu, \beta) = \mu + \beta \cdot Q(0,1)$
- Gumbel Distribution Properties (assume $\beta = 1$)
  - $X \sim \mathrm{Gumbel}(\mu_X)$ & $Y \sim \mathrm{Gumbel}(\mu_Y)$
    - $X - Y \sim \mathrm{logistic}(\mu_x - \mu_y)$
    - $P(X > Y) = \mathrm{CDF}\left(\mathrm{logistic}(\mu_x - \mu_y)\right) = \frac{1}{1+\exp(\mu_y - \mu_x)} = \mathrm{softmax}(0; \mu_x, \mu_y)$
  - Multi-dimensional extension
    - $k = \arg\max \mathrm{Gumbel}(\mu_i) = \arg\max(\mu_i + Q(0,1))$
    - $k \sim \mathrm{softmax}(\mu_i)$

VAE with Discrete Latent

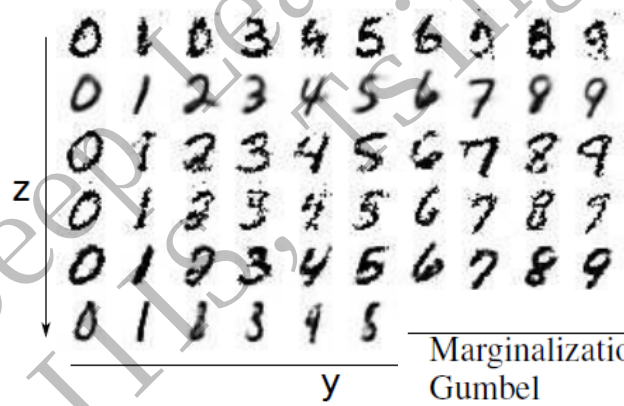
- VAE with Gumbel-Softmax Trick (Jang et al, Google, ICLR 2017)
  - Decoder $p(x|z;\theta)$ & $p(z_i) = \text{uniform}(0, K-1)$
  - Encoder: $q_i(z_i|x;\phi) = \text{softmax}(l_i(x;\phi))$ (logits)
    - $z_i \leftarrow \text{onehot}(k)$ where $k \sim \text{softmax}(l_i(x;\phi))$
    - $KL(q||p)$ can be computed in closed-form
  - Reparameterization trick
    - $z_i = \text{onehot}(\arg\max(l_i(x;\phi) + Q(0,1)))$
  - Differentiable approximation of $\text{onehot}(\arg\max(\cdot))$
    - $z_i \approx \lim_{\tau \to 0} \text{softmax}\left(\frac{l_i(x;\phi)+Q(0,1)}{\tau}\right)$ (τ as temperature)
- Overall loss function (with annealing τ; subscript omitted)
  - $J(\theta,\phi) = E_{\epsilon \sim Q(0,1)}\left[f\left(x, \text{softmax}\left(\frac{l(x;\phi)+\epsilon}{\tau}\right);\theta\right)\right] - KL(q||p)$
  - Code reference: https://blog.evjang.com/2016/11/tutorial-categorical-variational.html

VAE with Discrete Latent

- VAE with Gumbel-Softmax Trick (Jang et al, Google, ICLR 2017)
  - More efficient than marginalization
    - E.g., expand full expression of conditioned likelihood (Lecture 6, semi-supervised VAE)
  - Reparameterization trick for low-dimensional discrete z
    - About 100 dimensions in practice
  - What if we want ***really high-dimensional*** z?

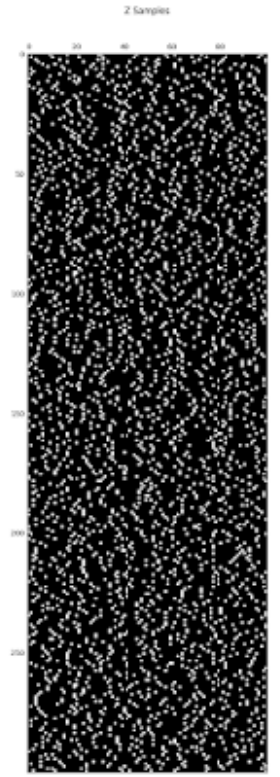


(a)



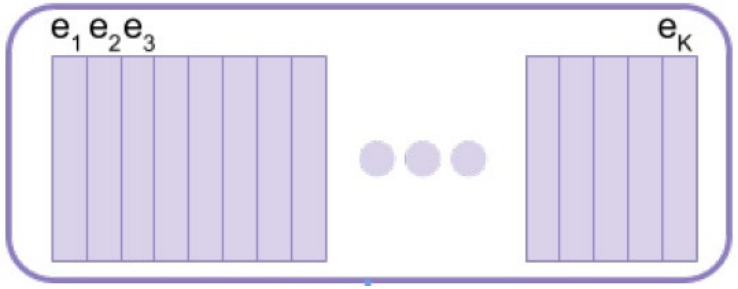
(b)

	ELBO	Accuracy
Marginalization	-106.8	92.6%
Gumbel	-109.6	92.4%
ST Gumbel-Softmax	-110.7	93.6%



VAE with Discrete Latent

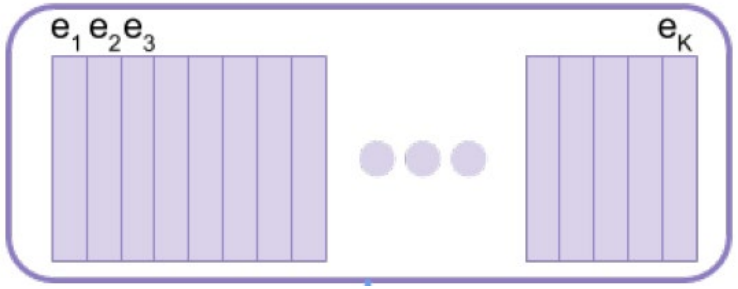
- Vector-Quantization VAE (VQ-VAE, DeepMind, NIPS 2017)
  - Dictionary embeddings $E \in \mathbb{R}^{d \times K}$
    - Embedding for factor i: $e_i \in \mathbb{R}^d$
  - Encoder $q_i(z|x) = NN(E, z_e(x;\phi)_i)$
    - $NN(E, v)$: nearest neighbor of v in E
    - $NN(E, v) = e_k, \quad k = \arg\min_i |v - e_i|_2$



  - Pros
    - $KL(q||p)$ becomes a constant
    - No need of reparameterization $L(\theta, \phi) = E_x[f(x, q(z|x))] = E_x[f(x, NN(E, z_e(x;\phi)); \theta)]$
    - Naturally works for arbitrarily high dictionary size
  - Issues
    - How to learn the dictionary E and $z_{e\theta}$? (NN is non-differentiable)
    - How to sample? (No prior distribution any more; $q(z|x)$ is a delta function)

VAE with Discrete Latent

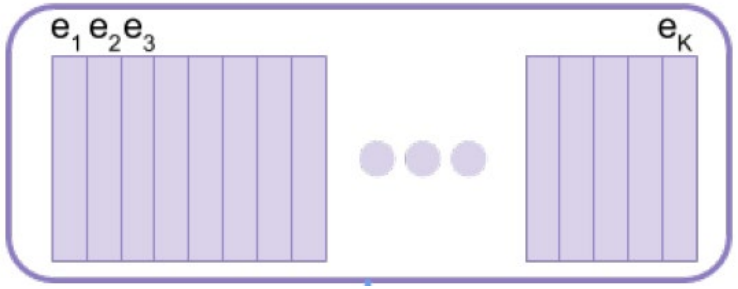
- Vector-Quantization VAE (VQ-VAE, DeepMind, NIPS 2017)
  - Dictionary embeddings $E \in \mathbb{R}^{d \times K}$
    - Embedding for factor i: $e_i \in \mathbb{R}^d$
  - Encoder $q_i(z|x) = NN(E, z_e(x;\phi)_i)$
    - $NN(E, v)$: nearest neighbor of v in E
    - $NN(E, v) = e_k, \quad k = \arg\min_i |v - e_i|_2$

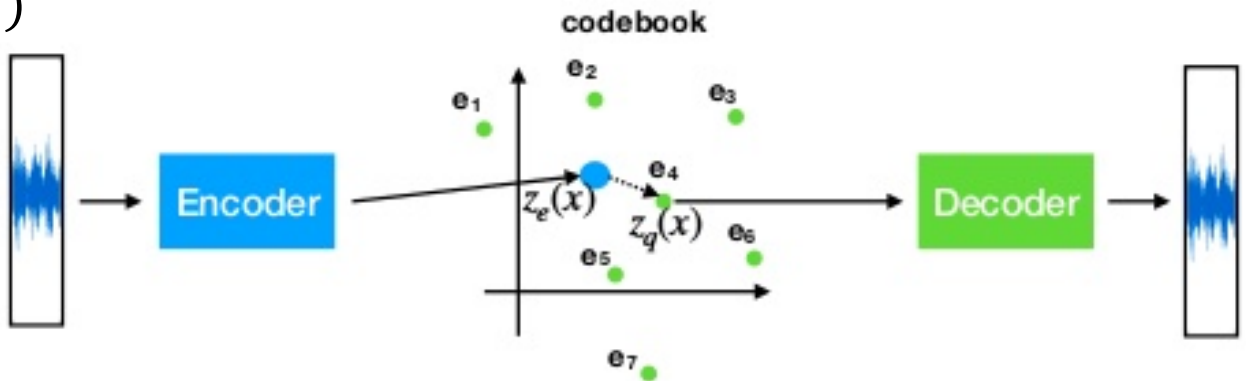


  - Pros
    - $KL(q||p)$ becomes a constant
    - No need of reparameterization $L(\theta, \phi) = E_x[f(x, q(z|x))] = E_x[f(x, NN(E, z_e(x;\phi)); \theta)]$
    - Naturally works for arbitrarily high dictionary size
  - Issues
    - How to learn the dictionary E and z_{e_θ}? (NN is non-differentiable)
    - How to sample? (No prior distribution any more; $q(z|x)$ is a delta function)

VAE with Discrete Latent

- Vector-Quantization VAE (VQ-VAE, DeepMind, NIPS 2017)
  - Dictionary embeddings $E \in \mathbb{R}^{d \times K}$
  - Encoder $q_i(z|x) = NN(E, z_e(x;\phi)_i)$
    - $NN(E, v) = e_k, \quad k = \arg\min_i |v - e_i|_2$
  - Loss function
    - $L(x;\theta,\phi) = f(x, z_q(x);\theta)$
    - where $z_q(x)_i = e_{z_i} = NN(E, z_e(x;\phi)_i)$
  - Issue #1: learn E
    - Idea: make e_{z_i} move towards $z_e(x;\phi)_i$
    - Use additional loss $\left|e_{z_i} - z_e(x;\phi)_i\right|_2^2$

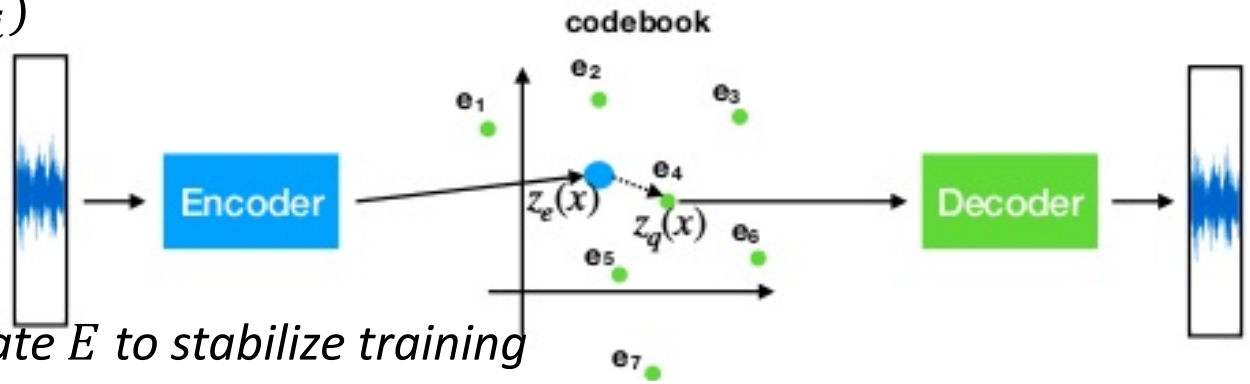




VAE with Discrete Latent

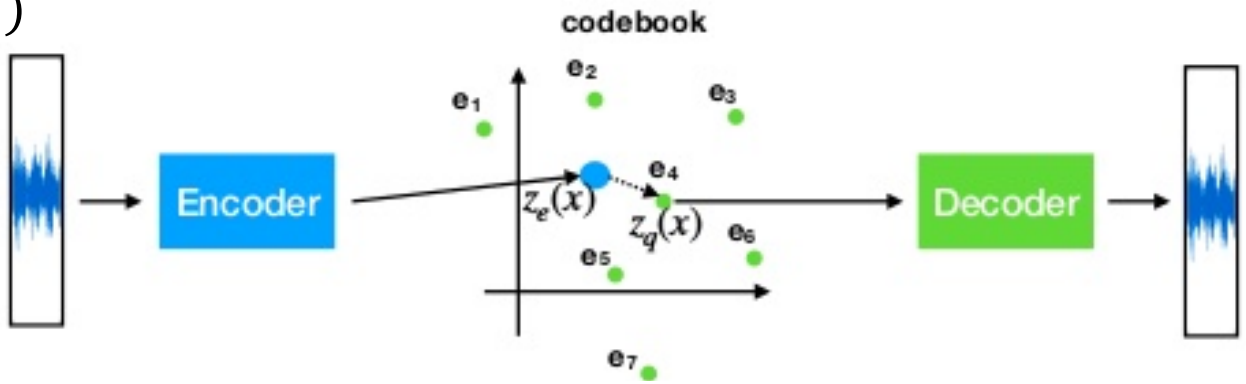
- Vector-Quantization VAE (VQ-VAE, DeepMind, NIPS 2017)
  - Dictionary embeddings $E \in \mathbb{R}^{d \times K}$
  - Encoder $q_i(z|x) = NN(E, z_e(x;\phi)_i)$
    - $NN(E, v) = e_k, \quad k = \arg\min_i |v - e_i|_2$
  - Loss function
    - $L(x;\theta,\phi) = f\big(x, z_q(x);\theta\big) + |\boldsymbol{e_z} - \mathrm{sg}[z_e(x;\phi)]|_2^2$
    - where $z_q(x)_i = e_{z_i} = NN(E, z_e(x;\phi)_i)$
  - Issue #1: learn E
    - Idea: make e_{z_i} move towards $z_e(x;\phi)_i$
    - Use additional loss $|\boldsymbol{e_{z_i}} - z_e(x;\phi)_i|_2^2$
    - *Trick: use moving average of $\overline{z_e}$ to update E to stabilize training*

sg[]: stop-gradient operator



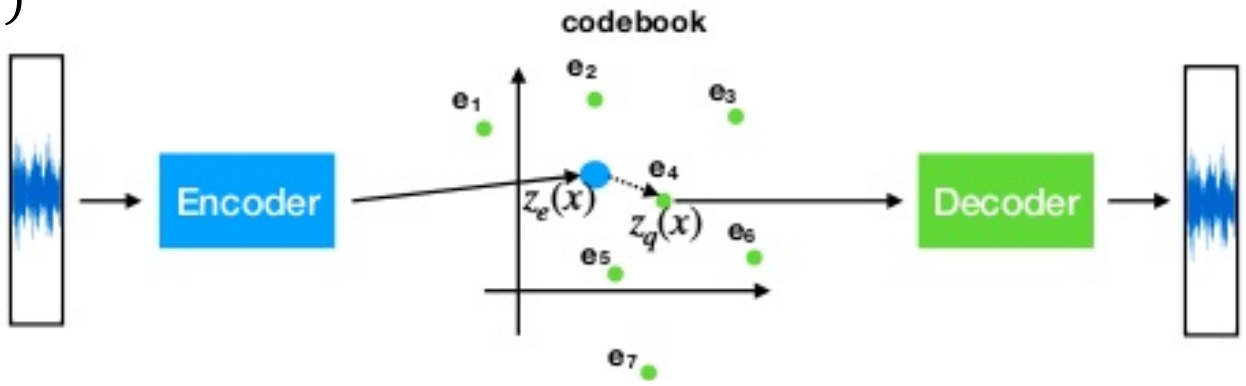
VAE with Discrete Latent

- Vector-Quantization VAE (VQ-VAE, DeepMind, NIPS 2017)
  - Dictionary embeddings $E \in \mathbb{R}^{d \times K}$
  - Encoder $q_i(z|x) = NN(E, z_e(x;\phi)_i)$
    - $NN(E, v) = e_k, \quad k = \arg\min_i |v - e_i|_2$
  - Loss function
    - $L(x;\theta,\phi) = f\big(x, z_q(x);\theta\big) + |e_z - \mathrm{sg}[z_e(x;\phi)]|_2^2$
    - where $z_q(x)_i = e_{z_i} = NN(E, z_e(x;\phi)_i)$
  - Issue #2: learn encoder $z_e(x;\phi)$
    - Idea: move $z_e(x;\phi)_i$ towards e_{z_i}
    - Use additional loss $\left|z_e(x;\phi)_i - e_{z_i}\right|_2^2$



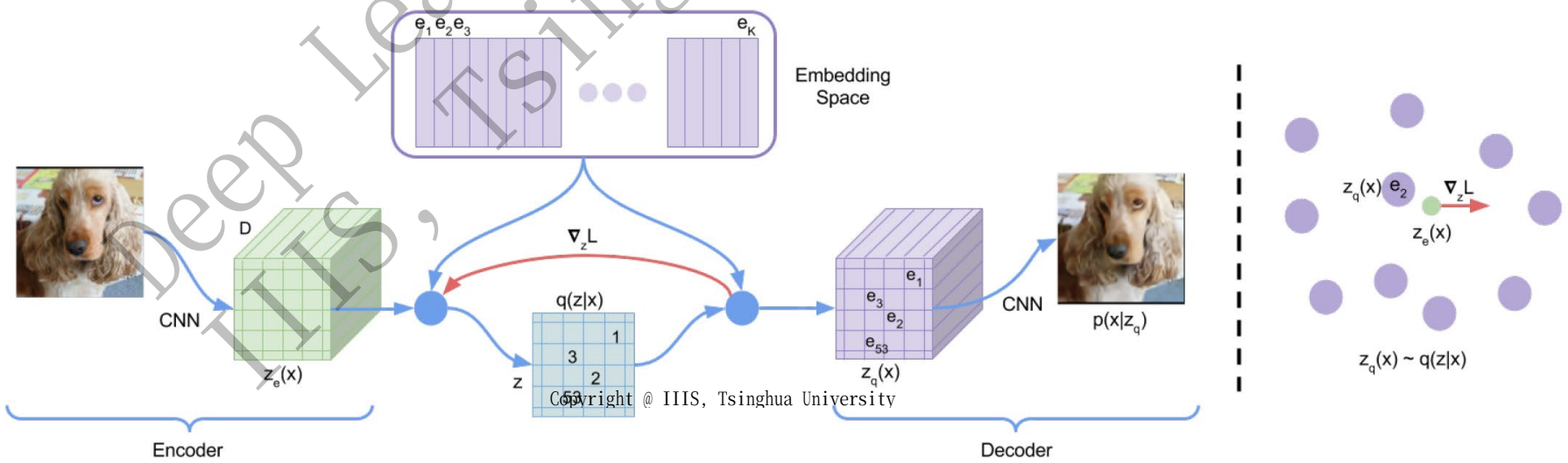
VAE with Discrete Latent

- Vector-Quantization VAE (VQ-VAE, DeepMind, NIPS 2017)
  - Dictionary embeddings $E \in \mathbb{R}^{d \times K}$
  - Encoder $q_i(z|x) = NN(E, z_e(x;\phi)_i)$
    - $NN(E, v) = e_k, \quad k = \arg\min_i |v - e_i|_2$
  - Loss function
    - $L(x;\theta,\phi) = f\big(x, z_q(x);\theta\big) + |e_z - \mathrm{sg}[z_e(x;\phi)]|_2^2 + \beta|\mathrm{sg}[e_z] - z_e(x;\phi)|_2^2$
    - where $z_q(x)_i = e_{z_i} = NN(E, z_e(x;\phi)_i)$
  - Issue #2: learn encoder $z_e(x;\phi)$
    - Idea: move $z_e(x;\phi)_i$ towards e_{z_i}
    - Use additional loss $|z_e(x;\boldsymbol{\phi})_i - e_{z_i}|_2^2$
    - *The paper chooses* $\beta = 0.25$



VAE with Discrete Latent

- Vector-Quantization VAE (VQ-VAE, DeepMind, NIPS 2017)
  - Encoder $q_i(z|x) = NN(E, z_e(x;\phi)_i)$
    - $E \in \mathbb{R}^{d\times K}, \quad NN(E,v) = e_k, \quad k = \arg\min_i |v - e_i|_2$
  - Loss function
    - $L(x;\theta,\phi) = f\big(x, z_q(x);\theta\big) + |e_z - \mathrm{sg}[z_e(x;\phi)]|_2^2 + \beta|\mathrm{sg}[e_z] - z_e(x;\phi)|_2^2$
    - where $z_q(x)_i = e_{z_i} = NN(E, z_e(x;\phi)_i)$



VAE with Discrete Latent

- Vector-Quantization VAE (VQ-VAE, DeepMind, NIPS 2017)
  - Encoder $q_i(z|x) = NN(E, z_e(x;\phi)_i)$
    - $E \in \mathbb{R}^{d\times K}, \quad NN(E,v) = e_k, \quad k = \arg\min_i |v - e_i|_2$
  - Loss function
    - $L(x;\theta,\phi) = f\big(x, z_q(x);\theta\big) + |e_z - \mathrm{sg}[z_e(x;\phi)]|_2^2 + \beta|\mathrm{sg}[e_z] - z_e(x;\phi)|_2^2$
    - where $z_q(x)_i = e_{z_i} = NN(E, z_e(x;\phi)_i)$
  - Issue #3: how to sample z?
    - Trivial solution: sample from uniform distribution over e
    - The posterior $z_q(x)$ can be very different from uniform!
      - *Remark: we do not have an effective KL constraint in VQ-VAE*
    - Idea: learn a prior distribution $p(z;\psi)$ to approximate the marginal posterior $z_q(x)$

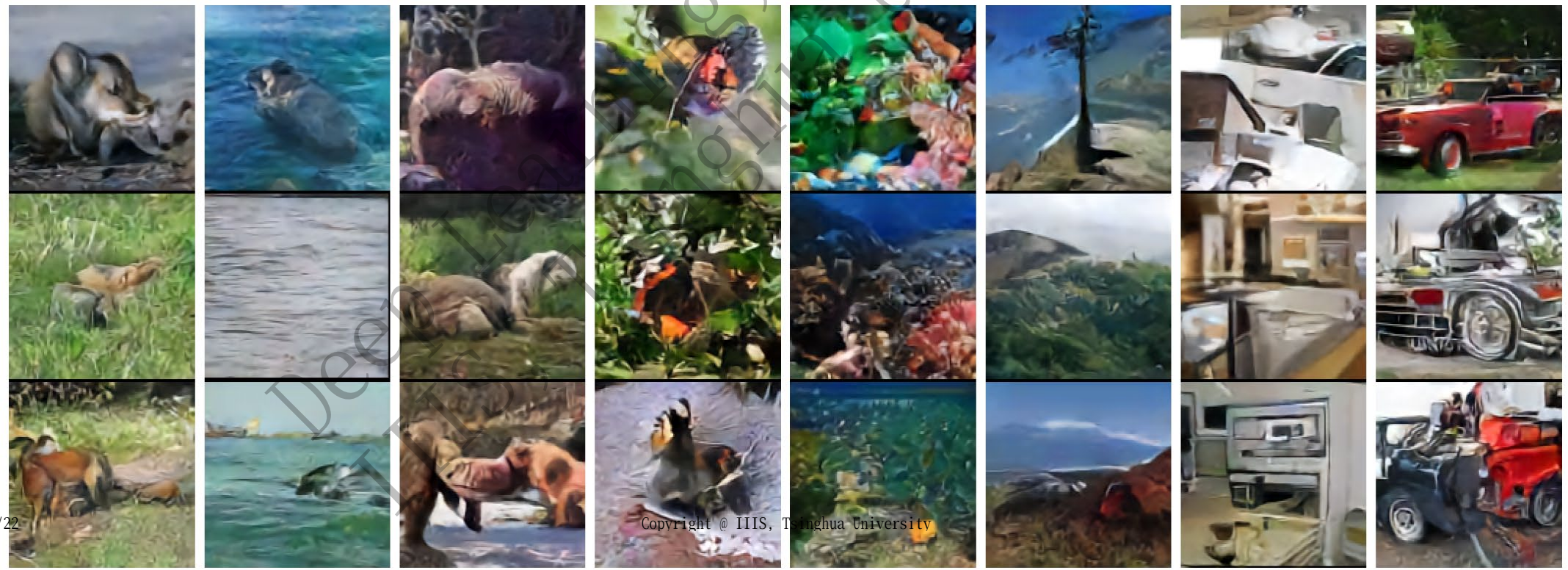
VAE with Discrete Latent

- Vector-Quantization VAE (VQ-VAE, DeepMind, NIPS 2017)
  - Encoder $q_i(z|x) = NN(E, z_e(x;\phi)_i)$
    - $E \in \mathbb{R}^{d\times K}, \quad NN(E, v) = e_k, \quad k = \arg\min_i |v - e_i|_2$
  - Loss function
    - $L(x;\theta,\phi) = f\big(x, z_q(x);\theta\big) + |e_z - \mathrm{sg}[z_e(x;\phi)]|_2^2 + \beta|\mathrm{sg}[e_z] - z_e(x;\phi)|_2^2$
    - where $z_q(x)_i = e_{z_i} = NN(E, z_e(x;\phi)_i)$
  - Learn $p(z;\psi)$ for the marginal posterior
    - After z_e and e are both trained, learn another model $p(z;\psi)$ for sampling
    - $p(z;\psi)$: autoregressive model over discrete z to mimic $z_q(x)$
      - PixelCNN for images
      - WaveNet for audio sequences

VAE with Discrete Latent

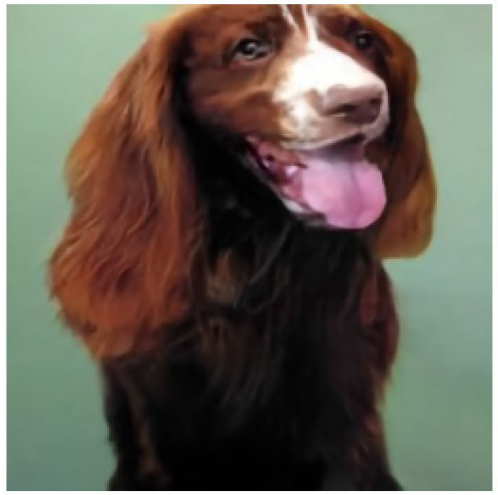
- Vector-Quantization VAE (VQ-VAE, DeepMind, NIPS 2017)
  - VAE + dictionary learning + vector quantization + PixelCNN prior
  - Stable learning + high quality generation + no mode collapse

Let's scale it up!

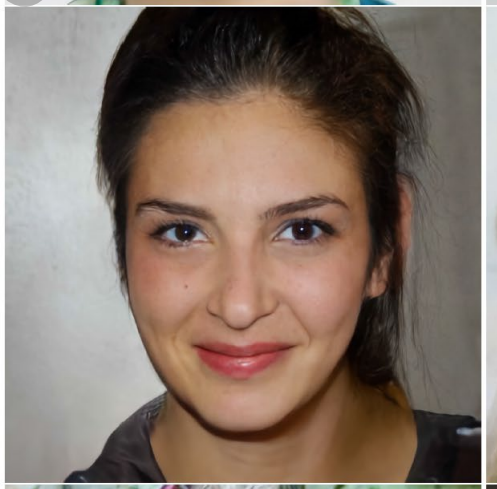
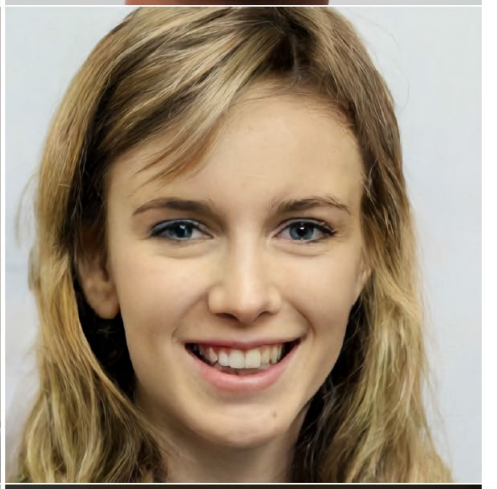


VAE with Discrete Latent

- VQ-VAE 2 (DeepMind, NIPS 2019)
  - Large-scale hierarchical VQ-VAE
  - Better sample quality than BigGAN (ICLR 2019)





VAE with Discrete Latent

- VQ-VAE 2 (DeepMind, NIPS 2019)
  - Main idea: bi-level VAE
    - Bottom level conditions on top level
    - Both level can access to input x
  - Top level: semantic structures
    - $h_{top} = E_{top}(x)$ (ResNet encoding)
    - $e_{top} = VQ(h_{top})$ (discrete latent)
  - Bottom level: textures
    - $h_{bottom} = E_{bottom}(x, e_{top})$ (ResNet)
    - $e_{bottom} = VQ(h_{bottom})$
  - Reconstruction
    - $\hat{x} = D(e_{top}, e_{bottom})$ (ResNet upsampling)

Algorithm 1 VQ-VAE training (stage 1)

Require: Functions E_{top} , E_{bottom}, D, $\mathbf{x}$ (batch of training images)

$\mathbf{h}_{top} \leftarrow E_{top}(\mathbf{x})$

▷ quantize with top codebook eq 1

$\mathbf{e}_{top} \leftarrow Quantize(\mathbf{h}_{top})$

$\mathbf{h}_{bottom} \leftarrow E_{bottom}(\mathbf{x}, \mathbf{e}_{top})$

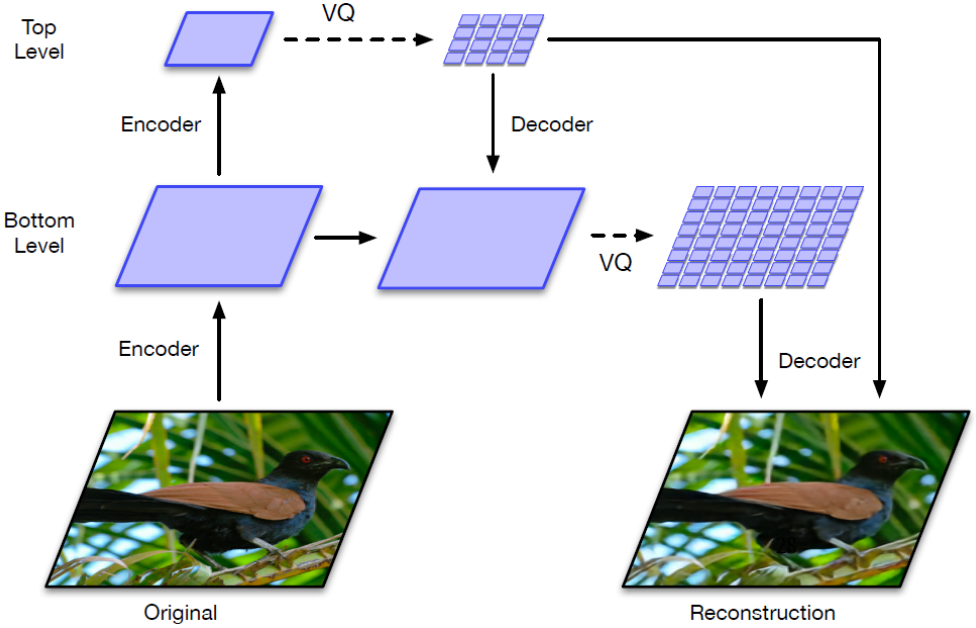
▷ quantize with bottom codebook eq 1

$\mathbf{e}_{bottom} \leftarrow Quantize(\mathbf{h}_{bottom})$

$\hat{\mathbf{x}} \leftarrow D(\mathbf{e}_{top}, \mathbf{e}_{bottom})$

▷ Loss according to eq 2

$\theta \leftarrow Update(\mathcal{L}(\mathbf{x}, \hat{\mathbf{x}}))$



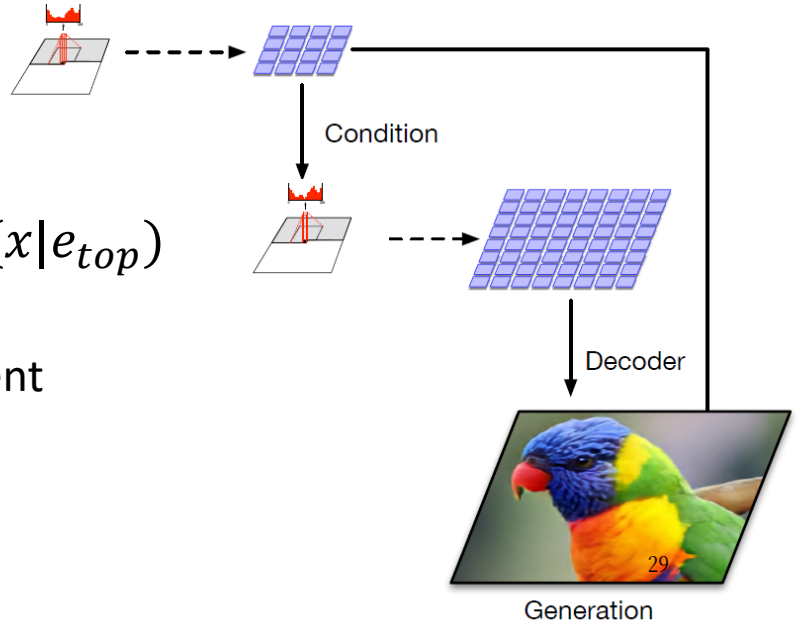
VAE with Discrete Latent

- VQ-VAE 2 (DeepMind, NIPS 2019)
  - Main idea: bi-level VAE
    - Bottom level conditions on top level
    - Both level can access to input x
  - Learned prior $p(e_{top}, e_{bottom})$ for sampling
    - $p_{top}(e_{top})$: PixelSnail for $E_{top}(x)$
      - PixelSnail: PixelCNN + Attention (ICML2018)
      - Powerful models for global structures
    - $p_{bottom}(e_{bottom}|e_{top})$: conditioned PixelCNN for $E_{bottom}(x|e_{top})$
      - PixelCNN only for efficiency
      - Bottom level focuses on textures so PixelCNN can be sufficient
    - Generation
      - Sample e_{top} and then e_{bottom}
      - $x = D(e_{top}, e_{bottom})$

Algorithm 2 Prior training (stage 2)

$\mathbf{T}_{top}, \mathbf{T}_{bottom} \leftarrow \emptyset$ ▷ training set
for $\mathbf{x} \in$ training set **do**
  $\mathbf{e}_{top} \leftarrow Quantize(E_{top}(\mathbf{x}))$
  $\mathbf{e}_{bottom} \leftarrow Quantize(E_{bottom}(\mathbf{x}, \mathbf{e}_{top}))$
  $\mathbf{T}_{top} \leftarrow \mathbf{T}_{top} \cup \mathbf{e}_{top}$
  $\mathbf{T}_{bottom} \leftarrow \mathbf{T}_{bottom} \cup \mathbf{e}_{bottom}$
end for
$p_{top} = \texttt{TrainPixelCNN}(\mathbf{T}_{top})$
$p_{bottom} = \texttt{TrainCondPixelCNN}(\mathbf{T}_{bottom}, \mathbf{T}_{top})$

▷ Sampling procedure
while true **do**
  $\mathbf{e}_{top} \sim p_{top}$
  $\mathbf{e}_{bottom} \sim p_{bottom}(\mathbf{e}_{top})$
  $\mathbf{x} \leftarrow D(\mathbf{e}_{top}, \mathbf{e}_{bottom})$
end while



VAE with Discrete Latent

- VQ-VAE 2 (DeepMind, NIPS 2019)
  - More levels can be used to further improve the generation quality

VAE with Discrete Latent

- VQ-VAE 2 (DeepMind, NIPS 2019)
  - Better sample quality and diversity; also stable training
  - Worse FID/IS than GAN (due to tiny blurry patches)
  - Better Classification Accuracy Score (CAS) score
    - Train a model using *generated samples*; measure the accuracy on ImageNet test set

	Top-1 Accuracy	Top-5 Accuracy
BigGAN deep	42.65	65.92
VQ-VAE	54.83	77.59
VQ-VAE after reconstructing	58.74	80.98
Real data	73.09	91.47

Table 2: Classification Accuracy Score (CAS) [25] for the real dataset, BigGAN-deep and our model.

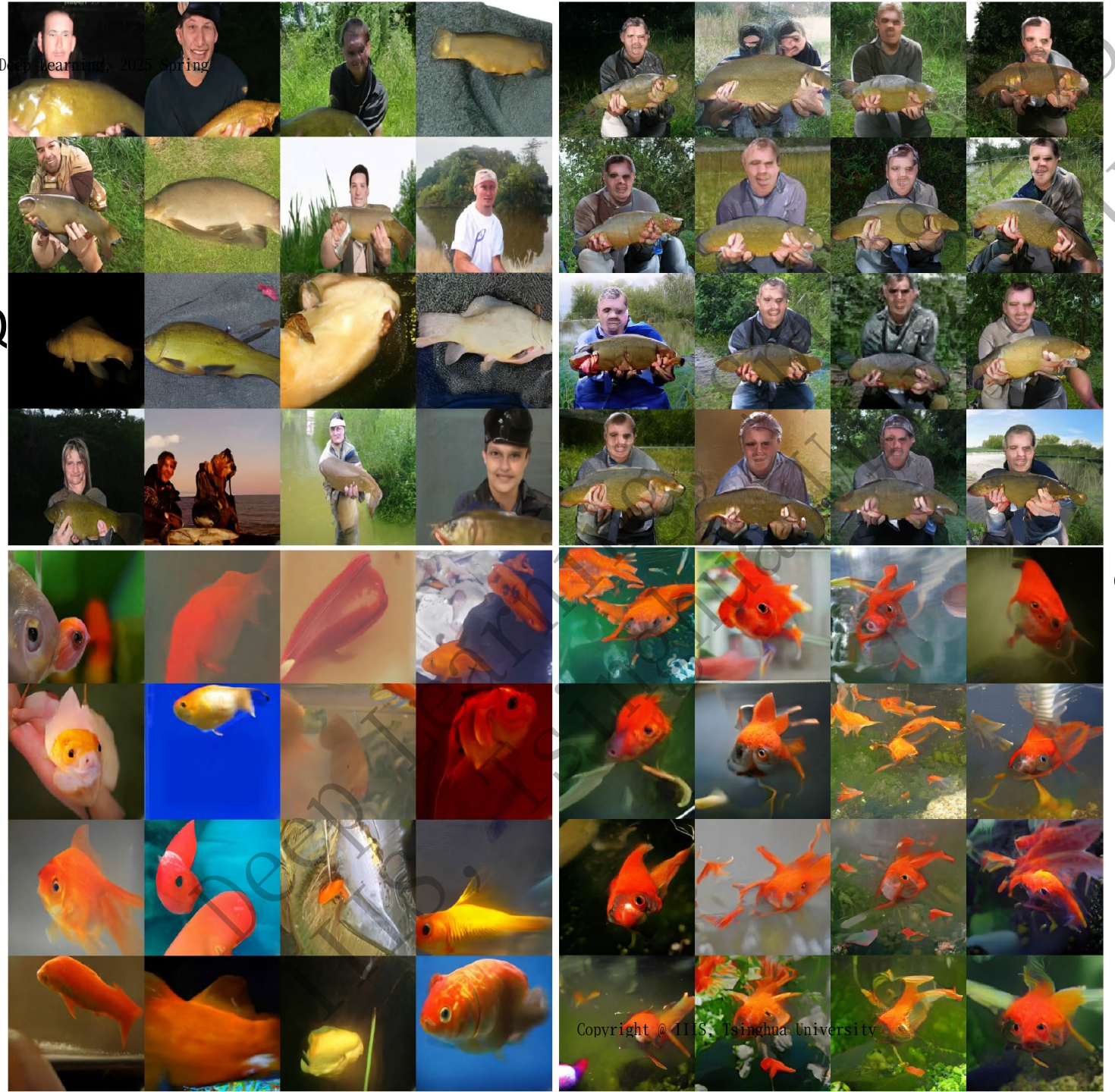
VAE with Discrete Latent

- VQ-VAE 2 (DeepMind, NIPS 2019)
  - Better sample quality and diversity; also stable training
  - Worse FID/IS than GAN (due to tiny blurry patches)
  - Better
    - Trai Net test set

VQ-VAE (Proposed) BigGAN deep

VA

- VQ
  -
  -
  -



on ImageNet test set

Let's scale it up further!

VAE with Discrete Latent

- DALL-E (OpenAI, 2021)
  - Named under the artist Salvador Dalí and Pixar's WALL·E.

VAE with Discrete Latent

- DALL-E (OpenAI, 2021)
  - Zero-shot text to image generation

TEXT PROMPT an illustration of a baby daikon radish in a tutu walking a dog

AI-GENERATED IMAGES

Edit prompt or view more images↓

TEXT PROMPT an armchair in the shape of an avocado. . .

AI-GENERATED IMAGES

Edit prompt or view more images↓

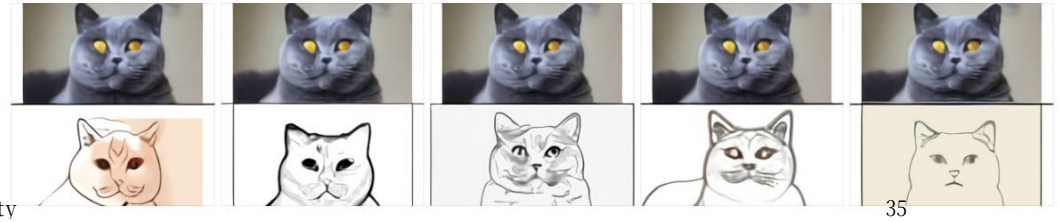
TEXT PROMPT a store front that has the word ‘openai’ written on it. . . .

AI-GENERATED IMAGES

Edit prompt or view more images↓

TEXT & IMAGE PROMPT the exact same cat on the top as a sketch on the bottom

AI-GENERATED IMAGES



Edit prompt or view more images↓

VAE with Discrete Latent

- DALL-E (OpenAI, 2021)
  - Zero-shot text to image generation
  - Key Ideas
    - Discrete VAE using ResNet with 8192 codebook size & 1024 image tokens
      - Temperature Annealed Gumbel-Softmax trick (No VQ; weighted average over codebook)
      - β-VAE with $\beta \to 6.6$; temperature $\tau \to 1/16$; lrate decay; Cosine scheduling
    - 12B Parameter Sparse Transformer model over paired Text-Image data (prior)
      - 64 layers with 62 heads with head size 64
      - Full-attention over texts; cross attention over text-image; sparse attention within images
    - Large-Scale Training (250M Text-Image pairs) with lots of tricks
      - Distributed training + mixed precision training
      - Log-Laplacian for loss function

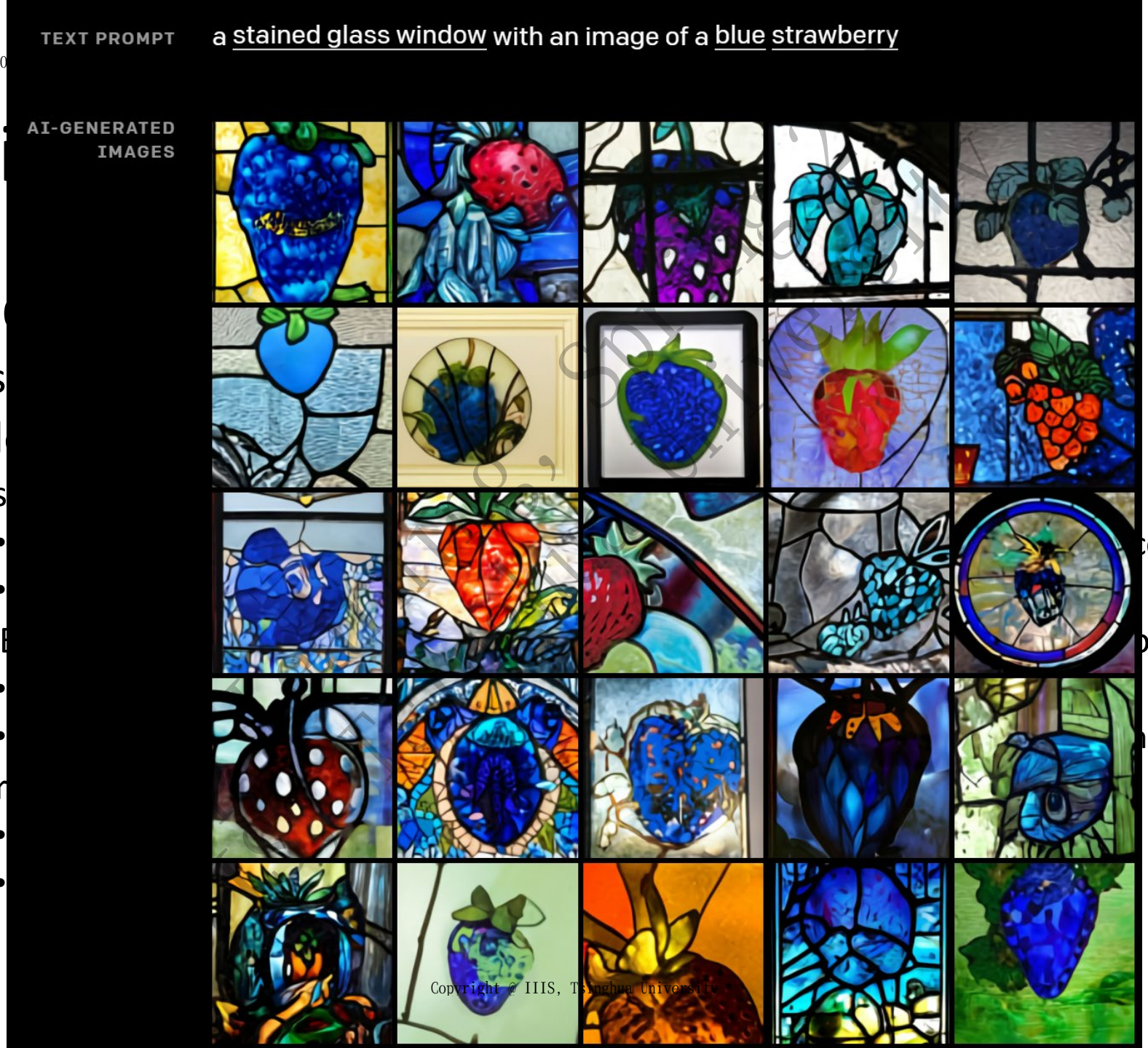
VAE wi

- DALL-E (
  - Zero-s
  - Key Id
    - Dis
      -
      -
    - 12E
      -
      -
    - Lar
      -
      -

codebook)

or)

ain images



VAE

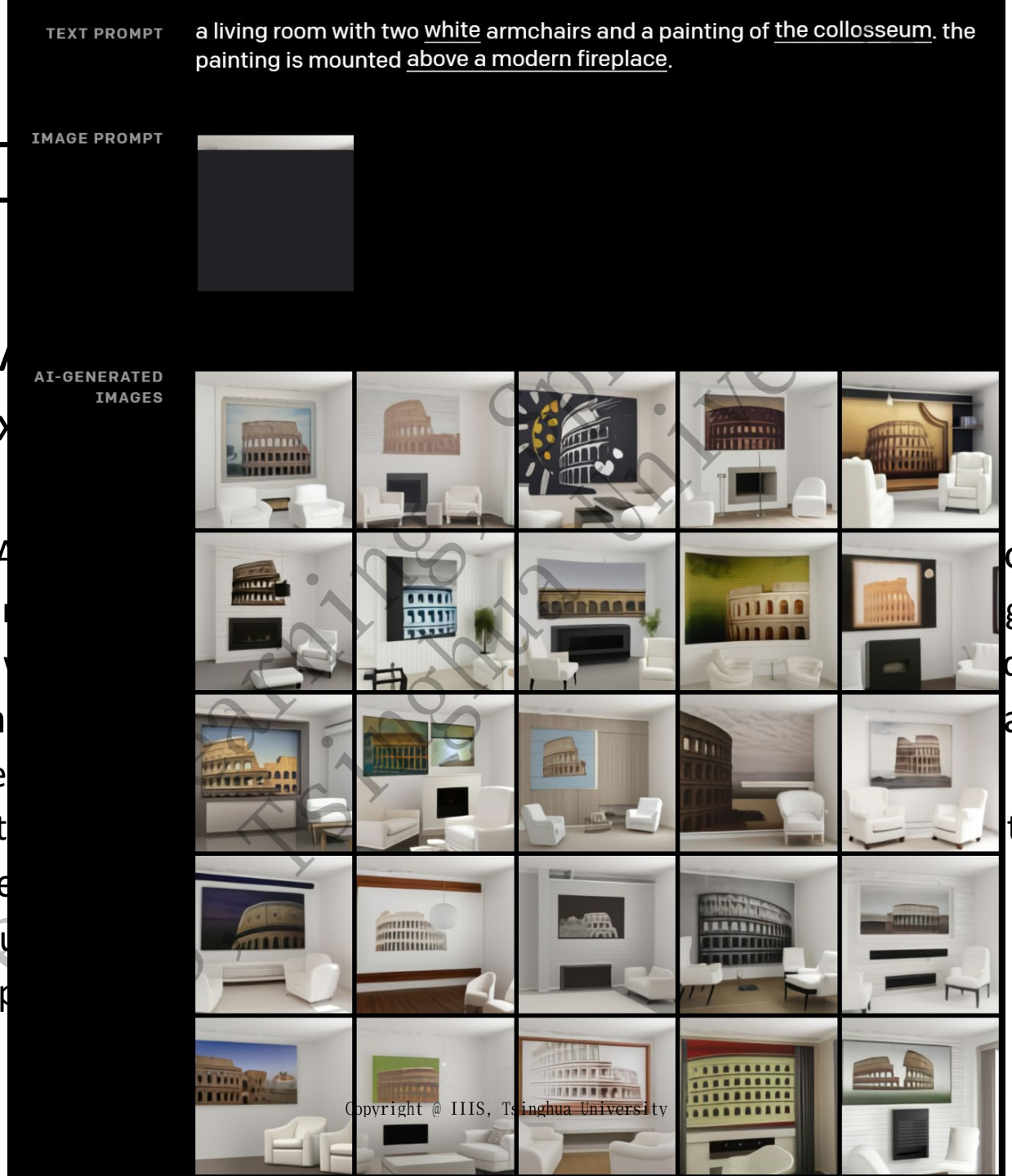
- DALL
  - Ze
  - Ke

ebook)

mages

VAE with D

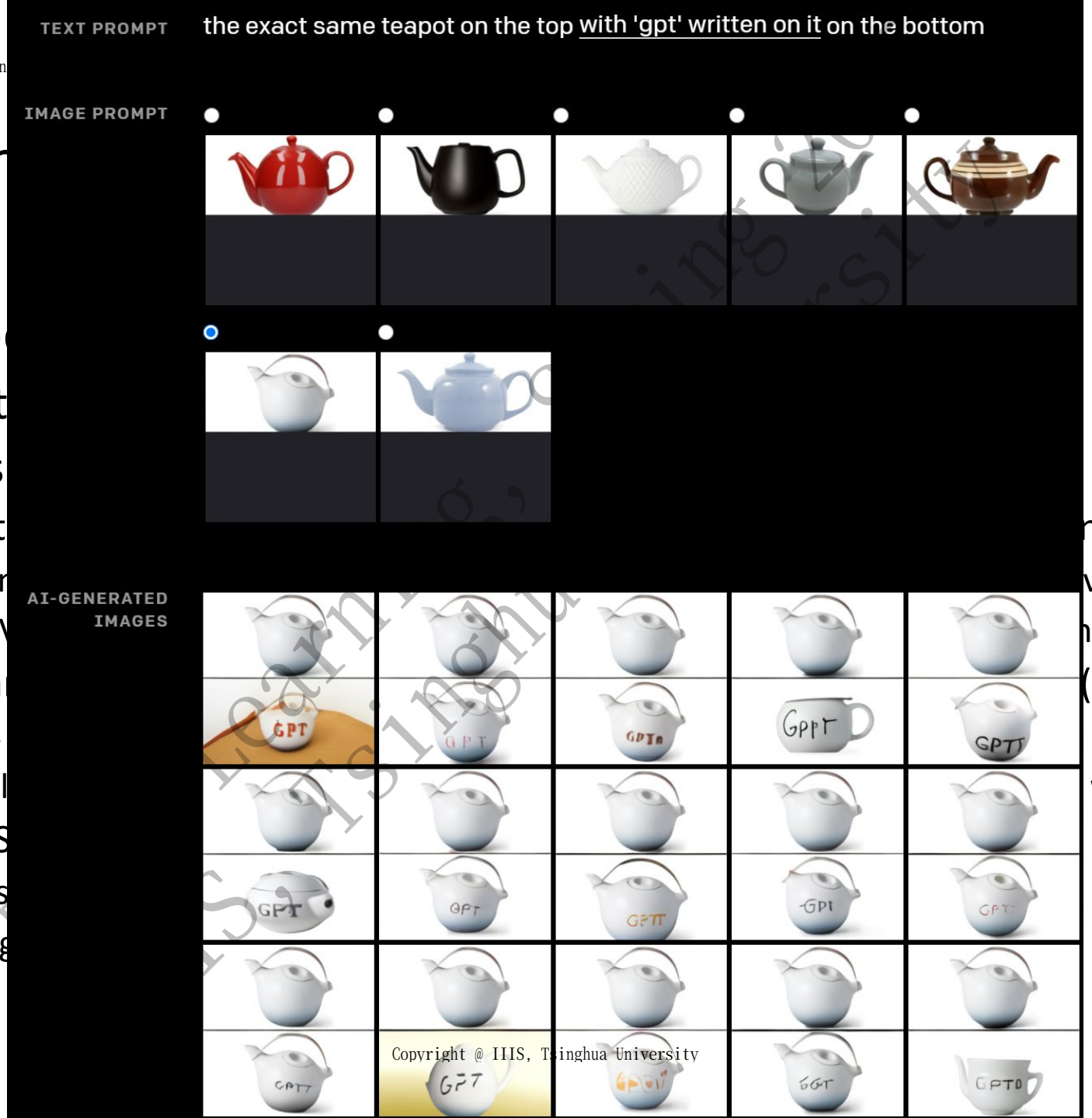
- DALL-E (OpenA
  - Zero-shot tex
  - Key Ideas
    - Discrete VA okens
      - Temper ge over codebook)
      - β-VAE duling
    - 12B Param ata (prior)
      - 64 laye
      - Full-att tion within images
    - Large-Scale
      - Distribu
      - Log-Lap



VAE with

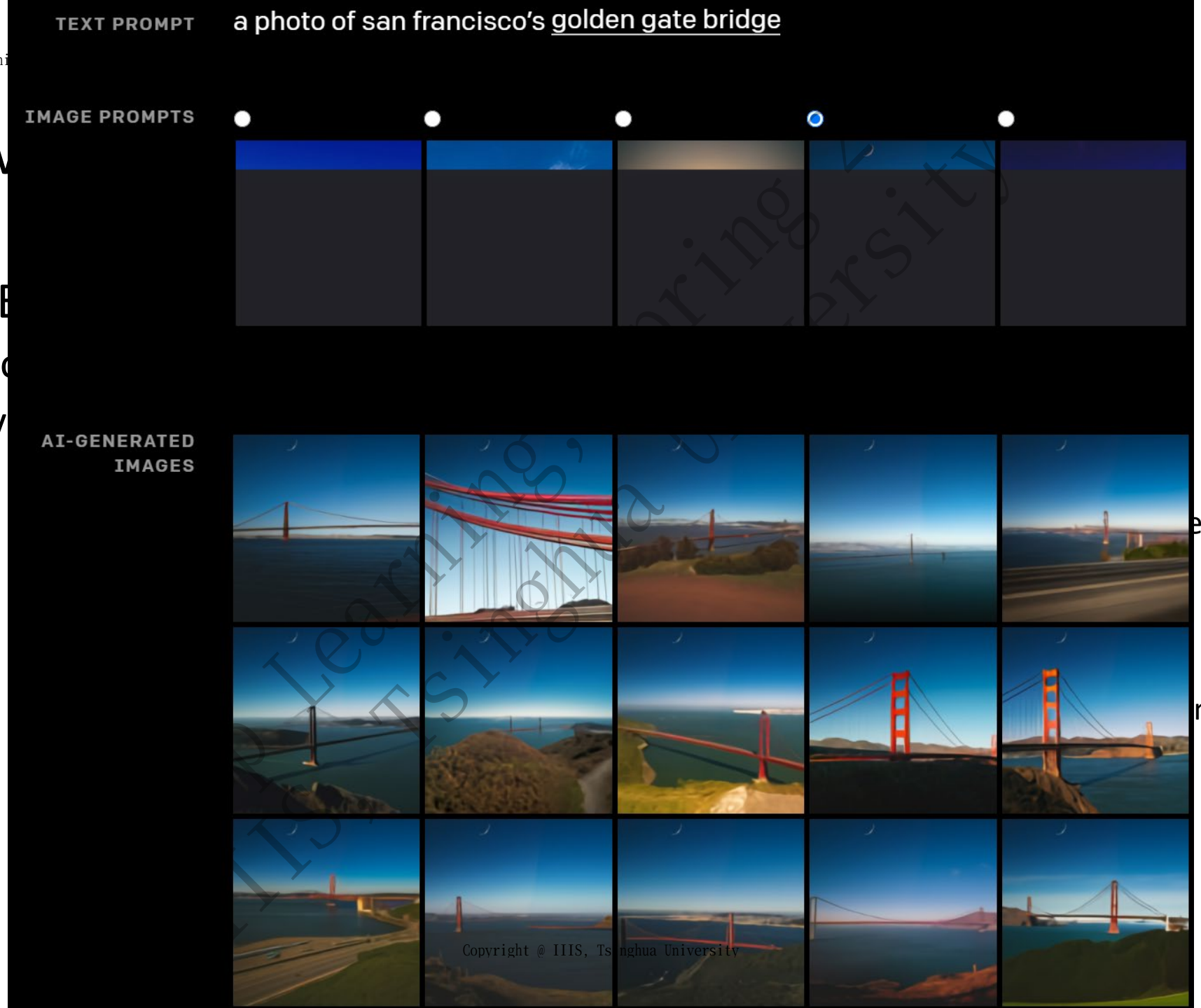
- DALL-E (Op
  - Zero-shot
  - Key Ideas
    - Discret
      - Ter
      - β-V
    - 12B Pa
      - 64
      - Ful
    - Large-S
      - Dis
      - Log

ns
ver codebook)
ng
(prior)
within images



VAE v

- DALL-E
  - Zero
  - Key
    -
    -
    -



ebook)

mages

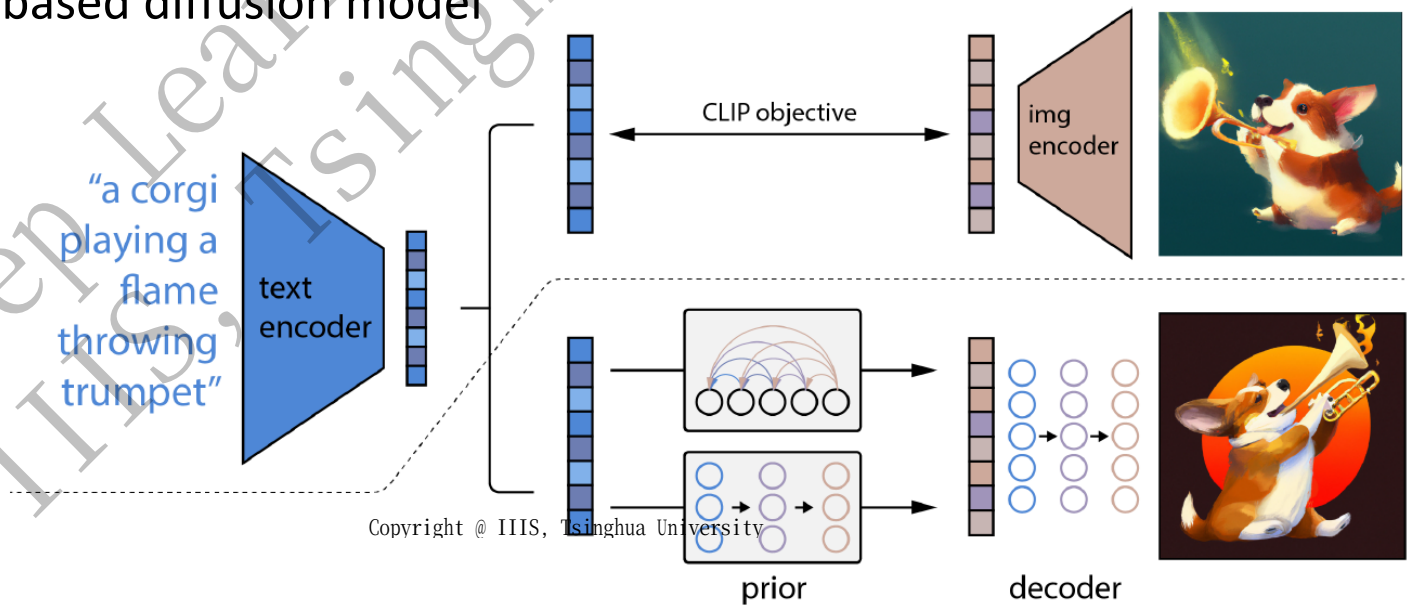
VAE with Discrete Latent

- DALL-E (OpenAI, 2021)
  - Zero-shot text to image generation
  - Key Ideas
    - Discrete VAE using ResNet with 8192 codebook size & 1024 image tokens
      - Temperature Annealed Gumbel-Softmax trick (No VQ; weighted average over codebook)
      - β-VAE with $\beta \to 6.6$; temperature $\tau \to 1/16$; lrate decay; Cosine scheduling
    - 12B Parameter Sparse Transformer model over paired Text-Image data (prior)
      - 64 layers with 62 heads with head size 64
      - Full-attention over texts; cross attention over text-image; sparse attention within images
    - Large-Scale Training (250M Text-Image pairs) with lots of tricks
      - Distributed training + mixed precision training
      - Log-Laplacian for loss function

Can we further scale it up?

Advanced Text2Image Generation

- DALL-E 2 (OpenAI, 2022.4)
  - Pretraining image/text aligned encoding (CLIP, to be covered in lec. 12)
  - Image generation model over image embeddings
    - hierarchical diffusion models; 1024x1024 resolution;
  - Prior model over aligned text/image embeddings
    - Transformer-based diffusion model



State-of-the-Art Text2Image Generation

DALL·E 1

DALL·E 2

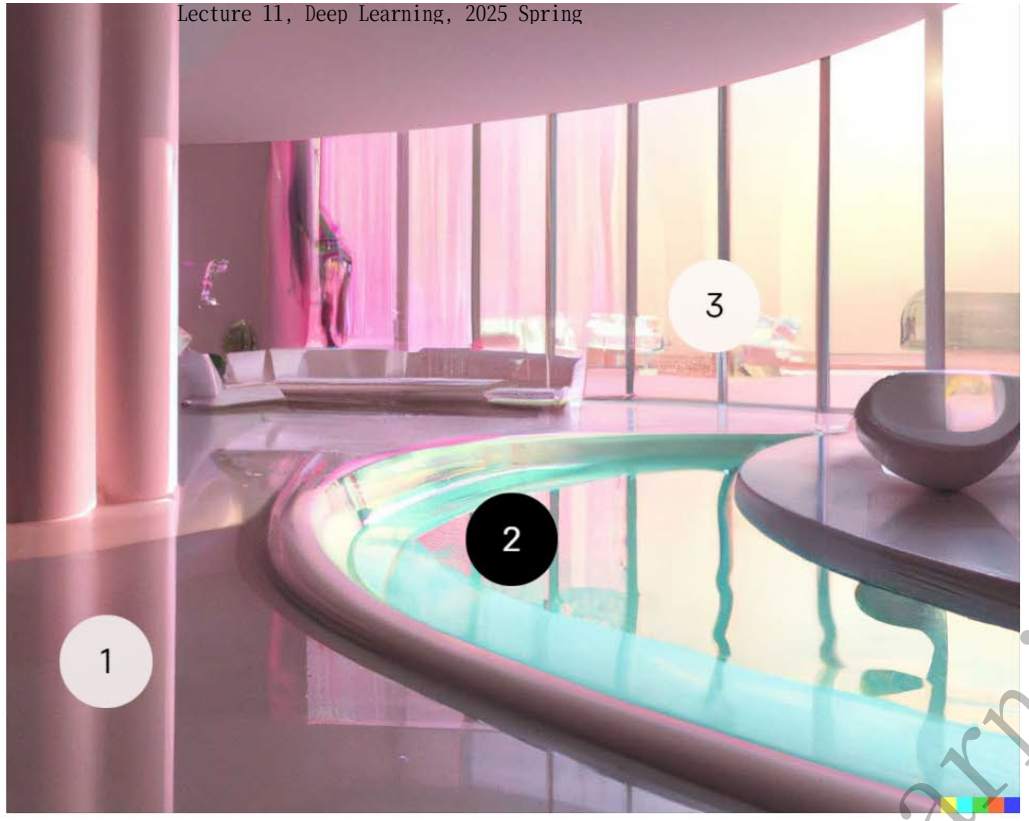
"a painting of a fox sitting in a field at sunrise in the style of Claude Monet"

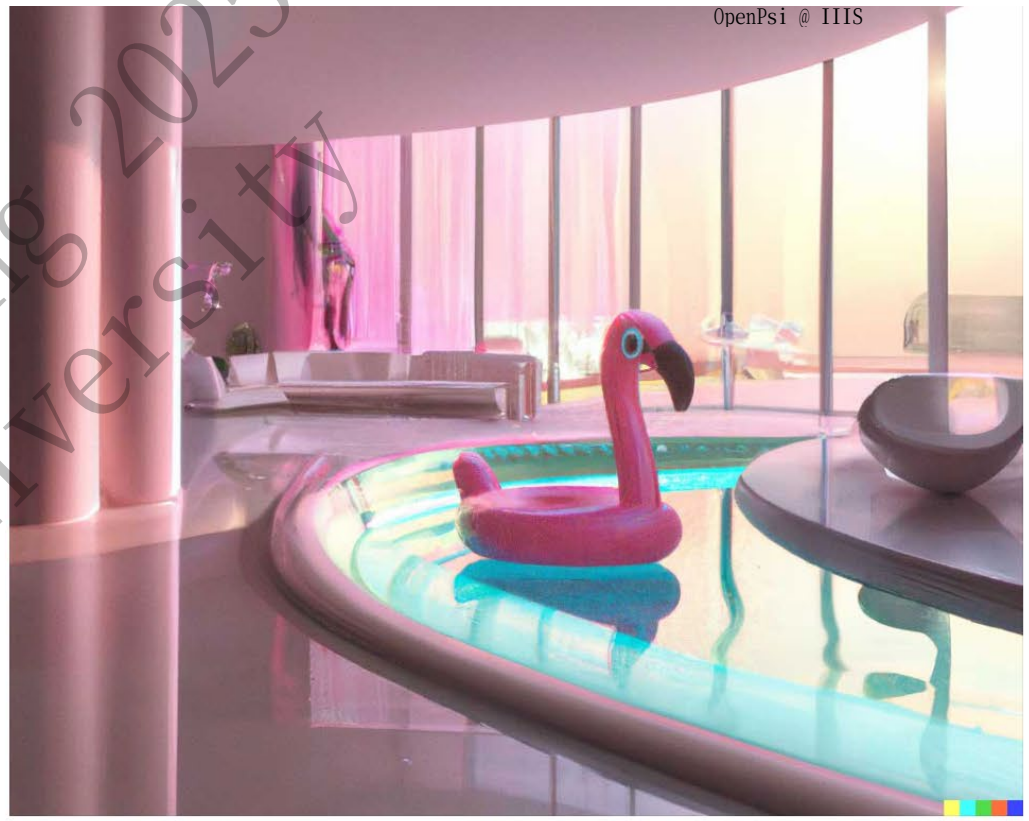
Stat

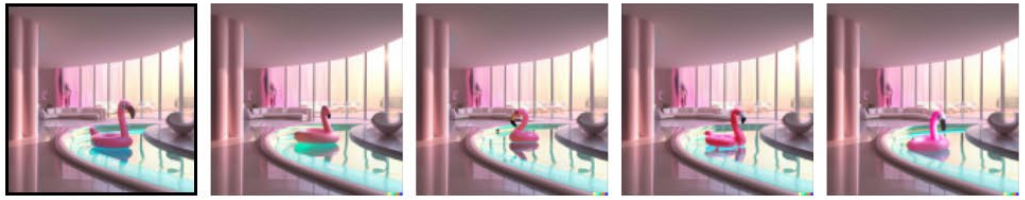
- DAL
  - F
  - I
  - F

n lec 14)

Semantic-Consistent Image Recovery







SELECT LOCATION TO ADD A FLAMINGO

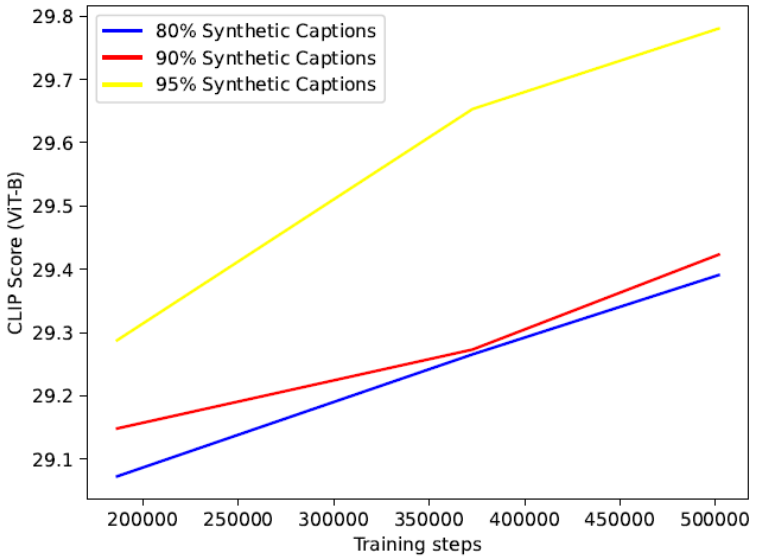
1 2 3

Semantic editing by latent space interpolation

More to find at https://openai.com/dall-e-2/

Advanced Text2Image Generation

- DALL-E 2 (OpenAI, 2022.4)
  - Pretraining image/text aligned encoding (CLIP, to be covered in lec. 12)
  - Image generation model over image embeddings
    - hierarchical diffusion models; 1024x1024 resolution;
  - Prior model over aligned text/image embeddings
    - Transformer-based diffusion model
- DALL-E 3 (OpenAI, 2023.10)
  - Improved training/models/dataset
  - GPT-4V to augment image captions in training data



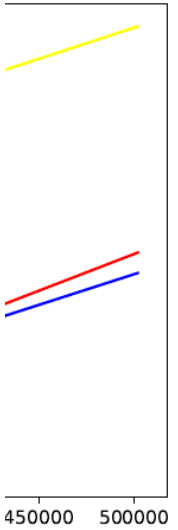
A

- D

- D

Image			
Alt Text	now at victorian plumbing.co.uk	is he finished..just about!	23 (19 of 30) 1200
SSC	a white modern bathtub sits on a wooden floor.	a quilt with an iron on it.	a jar of rhubarb liqueur sitting on a pebble background.
DSC	this luxurious bathroom features a modern freestanding bathtub in a crisp white finish. the tub sits against a wooden accent wall with glass-like panels, creating a serene and relaxing ambiance. three pendant light fixtures hang above the tub, adding a touch of sophistication. a large window with a wooden panel provides natural light, while a potted plant adds a touch of greenery. the freestanding bathtub stands out as a statement piece in this contemporary bathroom.	a quilt is laid out on a ironing board with an iron resting on top. the quilt has a patchwork design with pastel-colored strips of fabric and floral patterns. the iron is turned on and the tip is resting on top of one of the strips. the quilt appears to be in the process of being pressed, as the steam from the iron is visible on the surface. the quilt has a vintage feel and the colors are yellow, blue, and white, giving it an antique look.	rhubarb pieces in a glass jar, waiting to be pickled. the colors of the rhubarb range from bright red to pale green, creating a beautiful contrast. the jar is sitting on a gravel background, giving a rustic feel to the image.

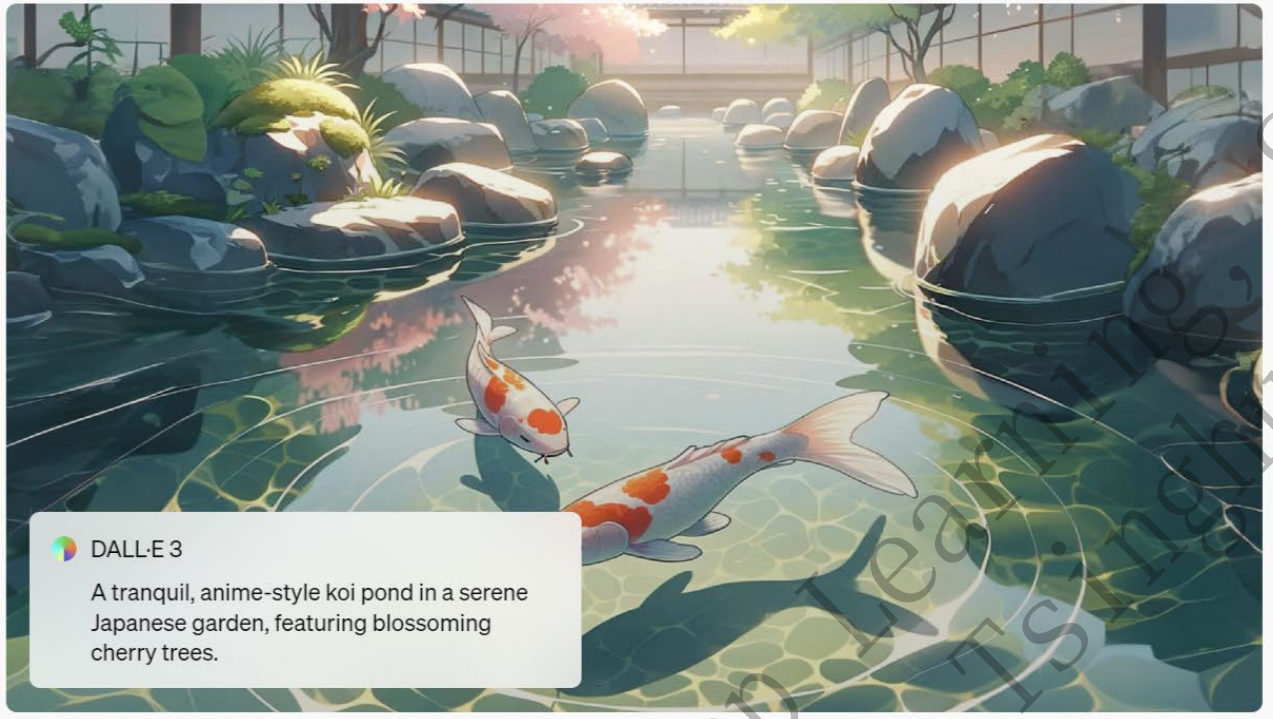
Figure 3 – Examples of alt-text accompanying selected images scraped from the internet, short synthetic captions (SSC), and descriptive synthetic captions (DSC).

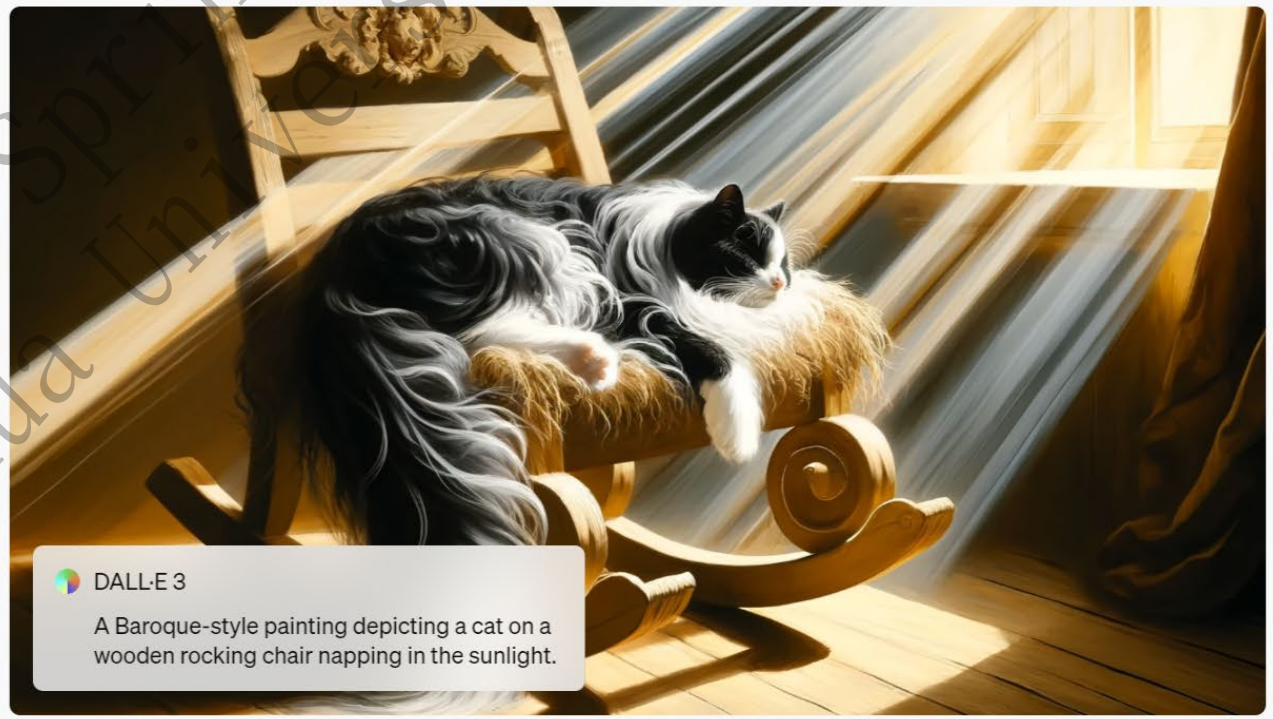


DALL·E 2 · An expressive oil painting of a chocolate chip cookie being dipped in a glass of milk, depicted as an explosion of flavors.

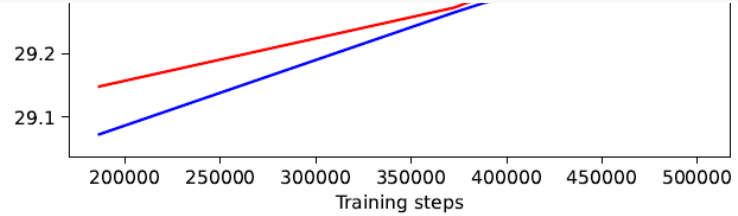
DALL·E 3 · An expressive oil painting of a chocolate chip cookie being dipped in a glass of milk, depicted as an explosion of flavors.

Advanced Text2Image Generation



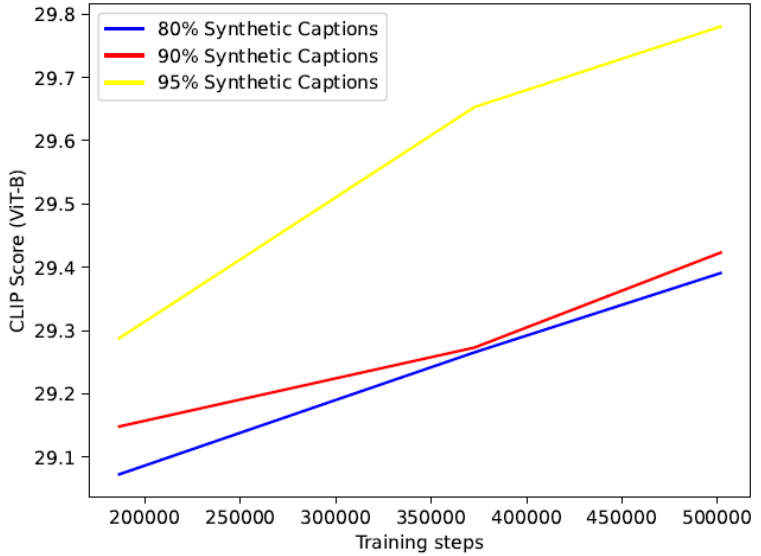


GPT-4V to augment image captions in training data



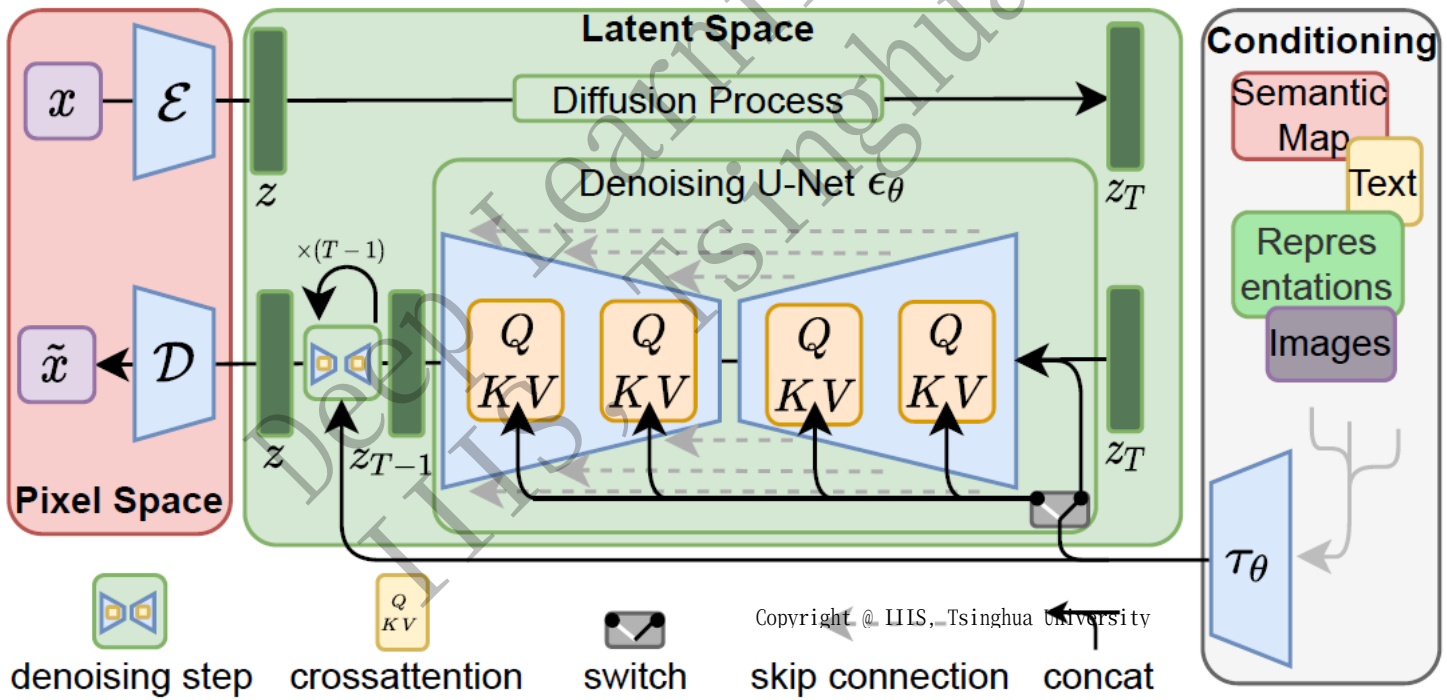
Advanced Text2Image Generation

- DALL-E 2 (OpenAI, 2022.4)
  - Pretraining image/text aligned encoding (CLIP, to be covered in lec. 12)
  - Image generation model over image embeddings
    - hierarchical diffusion models; 1024x1024 resolution;
  - Prior model over aligned text/image embeddings
    - Transformer-based diffusion model
- DALL-E 3 (OpenAI, 2023.10)
  - Improved training/models/dataset
  - GPT-4V to augment image captions in training data
  - You can also combined ChatGPT and DALL-E3!
    - ChatGPT can generate prompts in an interactive way



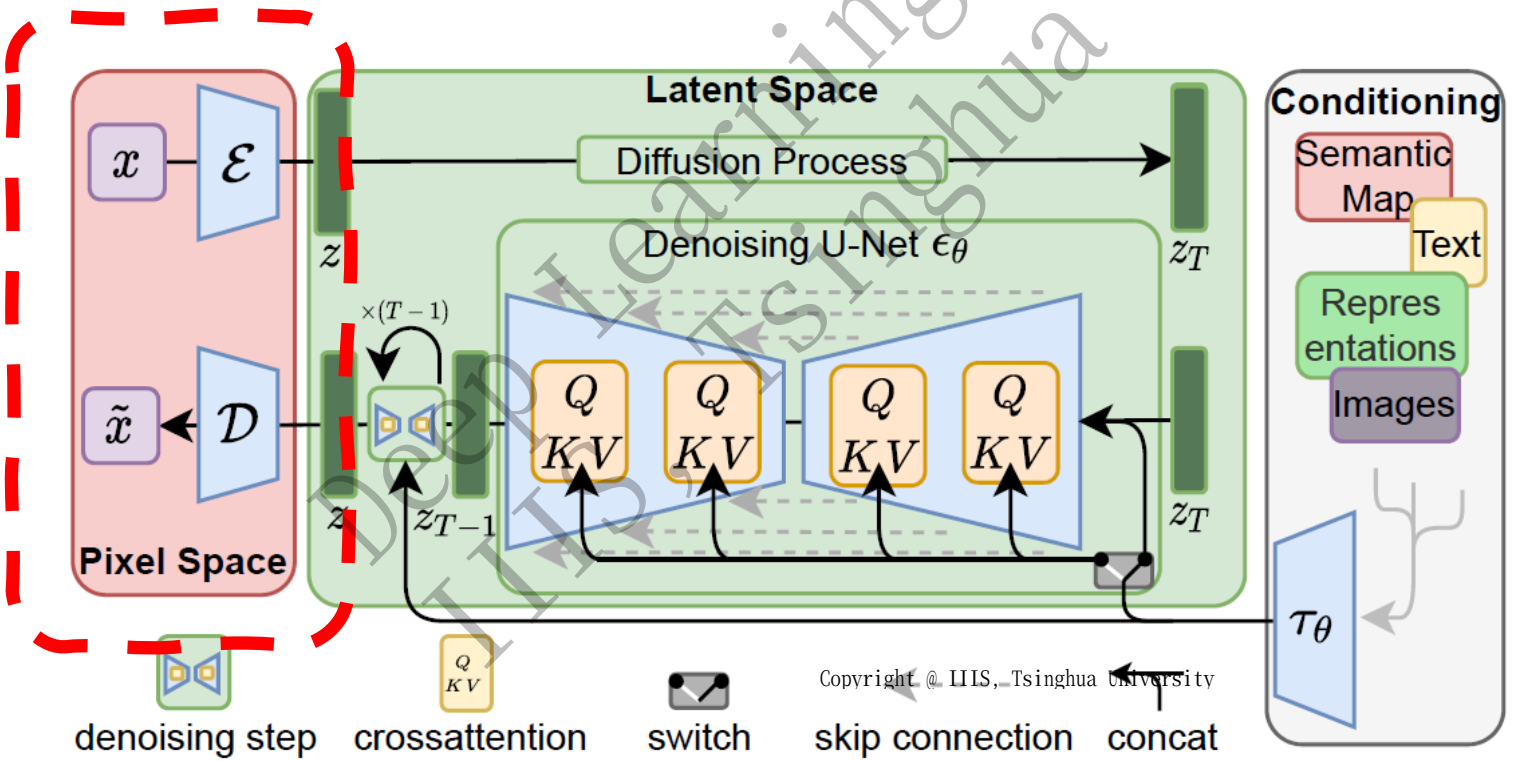
Advanced Text2Image Generation

- We can also adopt pretrained models for latent representations
- Latent Diffusion Model (University of Munich & Runway ML, 2021)



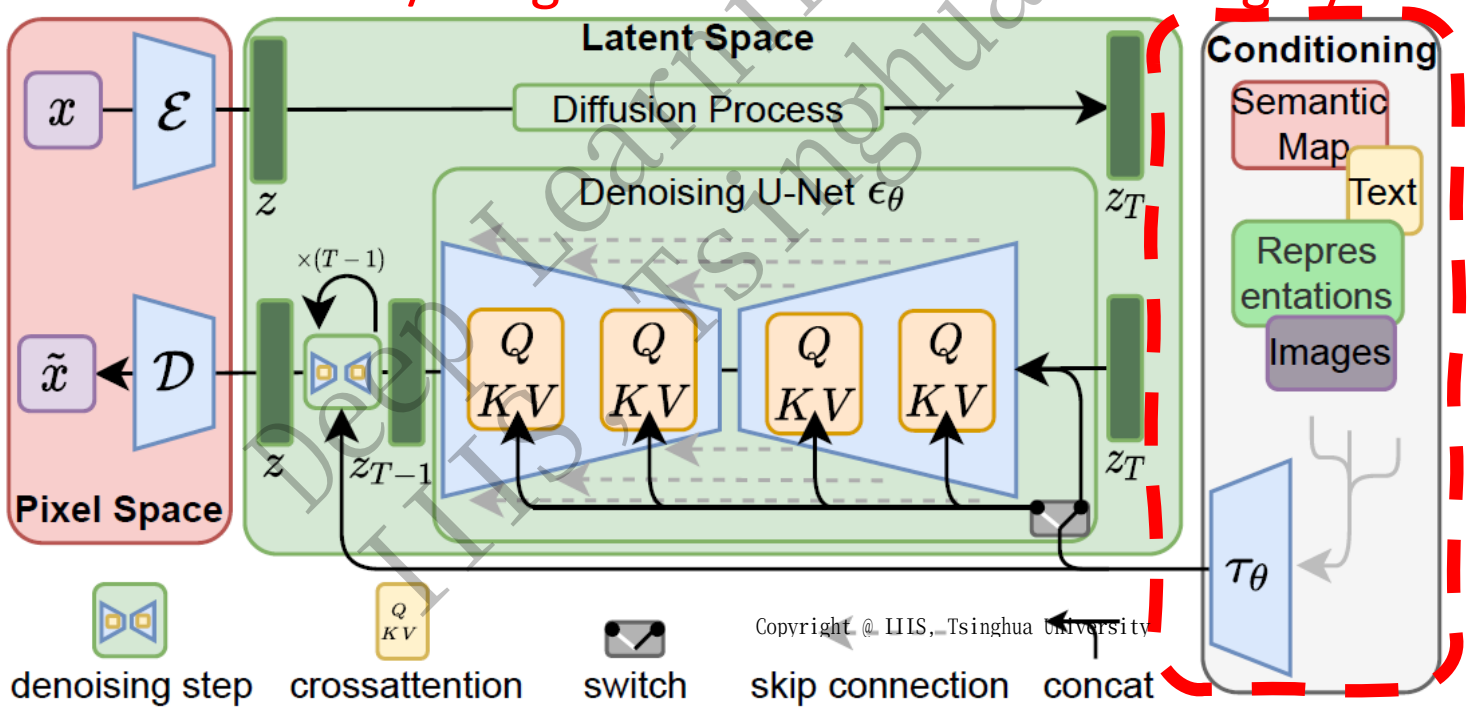
Advanced Text2Image Generation

- We can also adopt pretrained models for latent representations
- Latent Diffusion Model (University of Munich & Runway ML, 2021)
  - A pretrained VAE for image latent space, run diffusion process in latent space



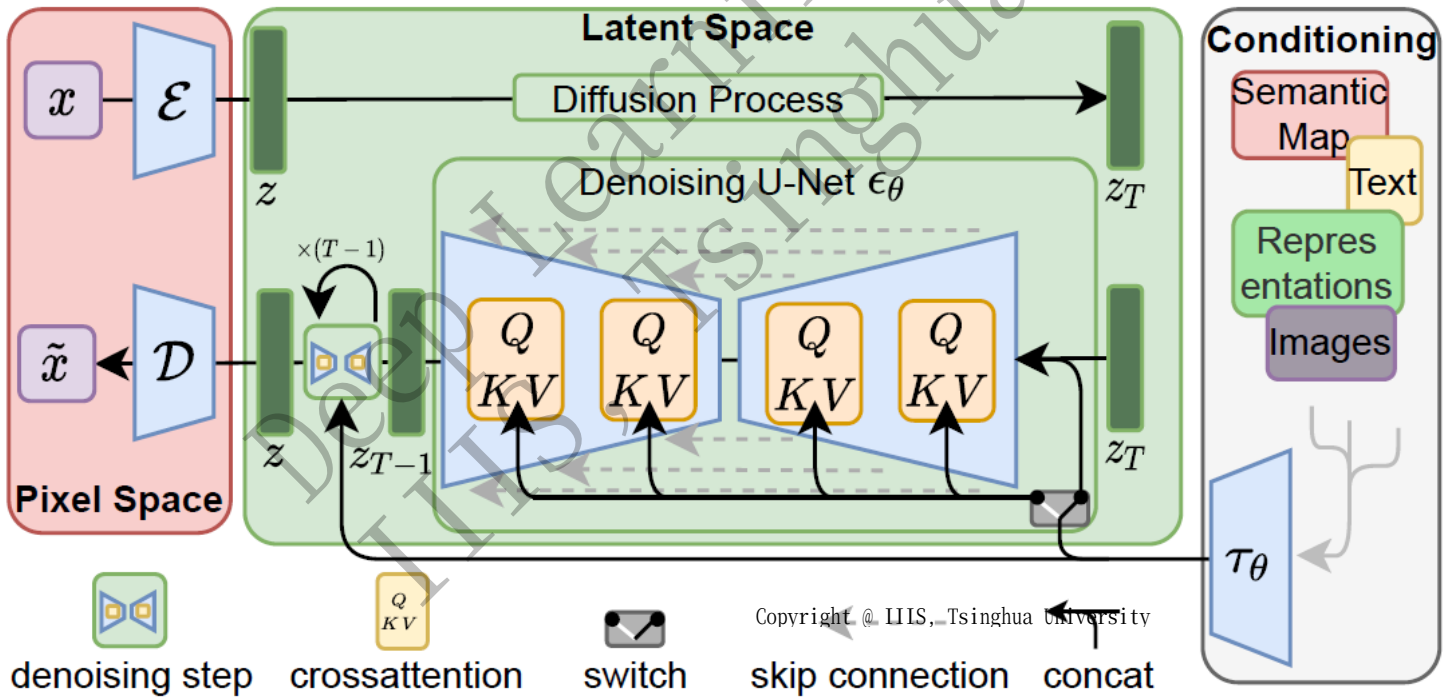
Advanced Text2Image Generation

- We can also adopt pretrained models for latent representations
- Latent Diffusion Model (University of Munich & Runway ML, 2021)
  - A pretrained VAE for image latent space, run diffusion process in latent space
  - Pretrained LLM/Image encoders for conditioning by cross attention



Advanced Text2Image Generation

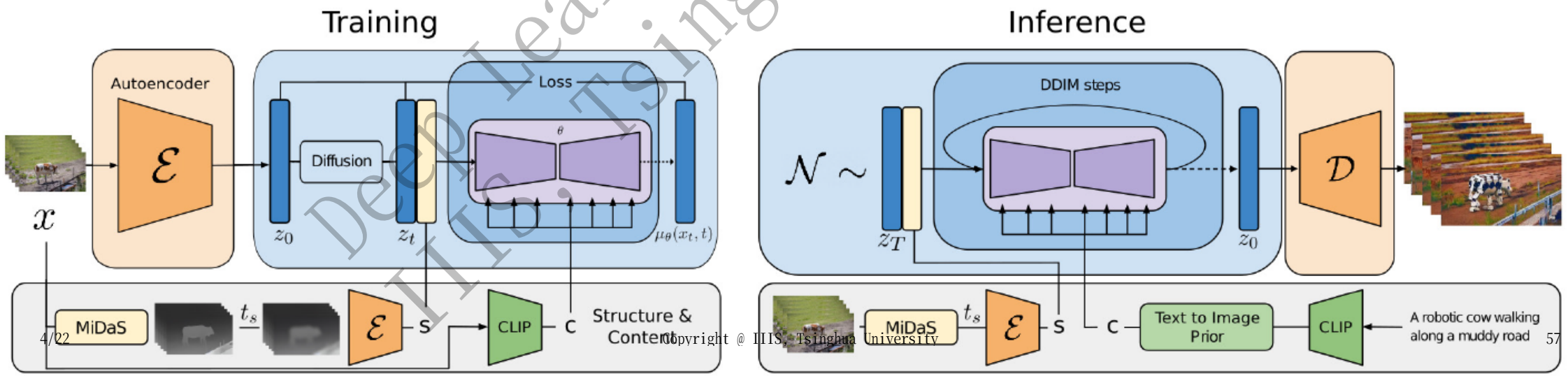
- We can also adopt pretrained models for latent representations
- Latent Diffusion Model (University of Munich & Runway ML, 2021)
  - A pretrained VAE for image latent space, run diffusion process in latent space
  - Pretrained LLM/Image encoders for conditioning by cross attention



This is the technical foundation of Stable Diffusion models

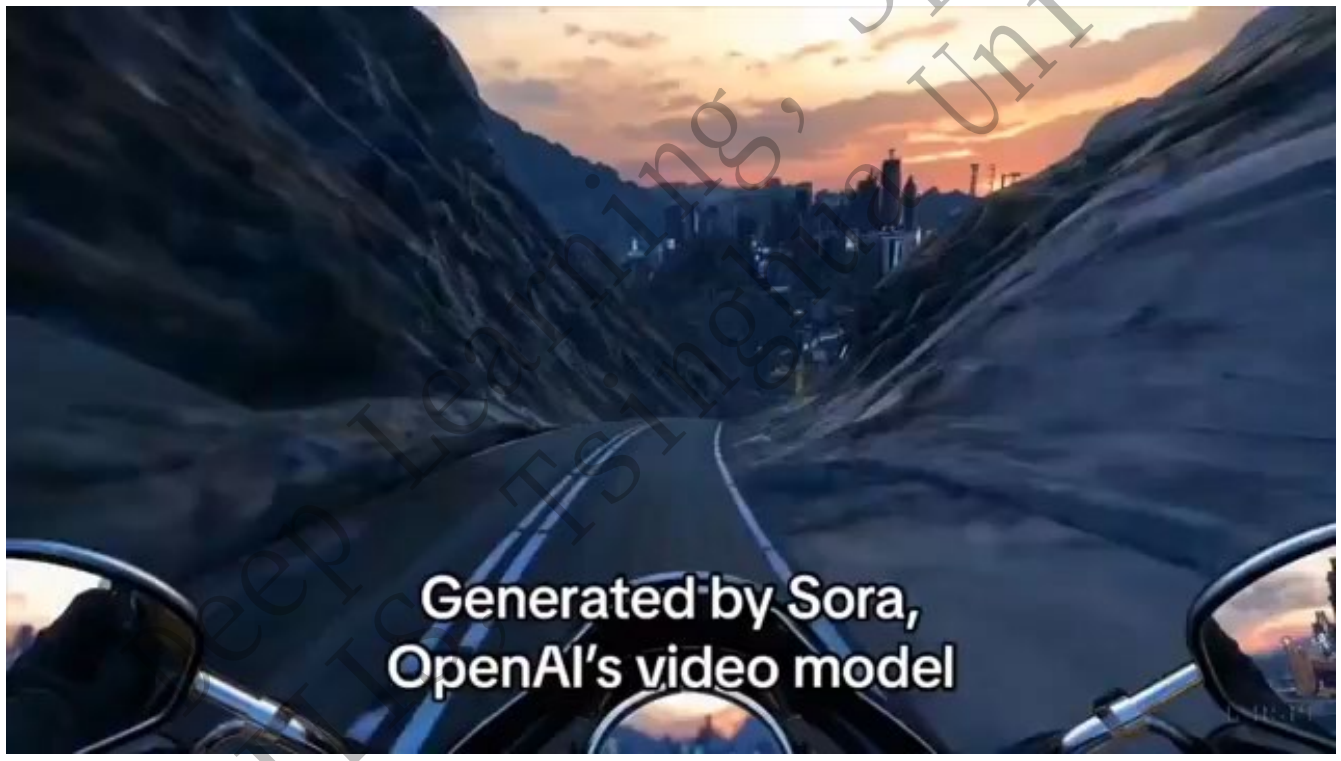
Advanced Text2Image Generation

- We can also adopt pretrained models for latent representations
- Runway Gen1 (Runway ML, 2023)
  - You can also process any structure information (e.g., pose, depth, etc) using pretrained encoders
  - Diffusion model for controller video generation



Advanced Text2Image Generation

- And we recently have a lot of advanced video generation models…



OpenAI Sora

Advanced Text2Image Generation

- And we recently have a lot of advanced video generation models…

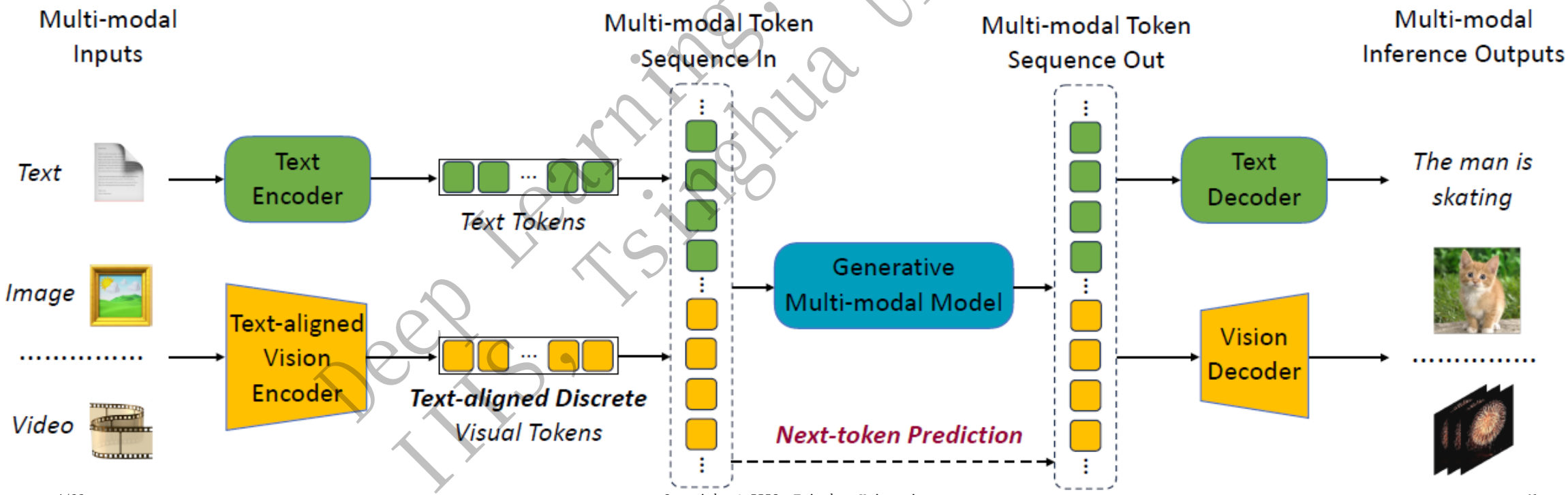
Runway Gen-4

All these models rely on a pretrained conditioning representation…

Can we have a unified model?

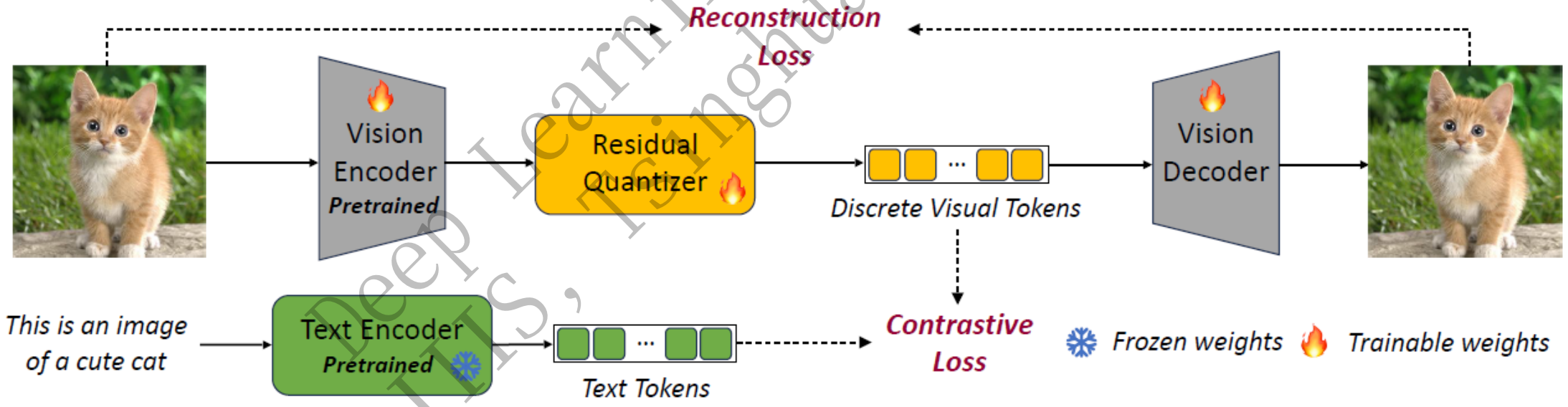
Unified Understanding and Generation

- VILA-U (Nvidia, ICLR 2025)
  - A unified multi-modal model with both understanding and generation



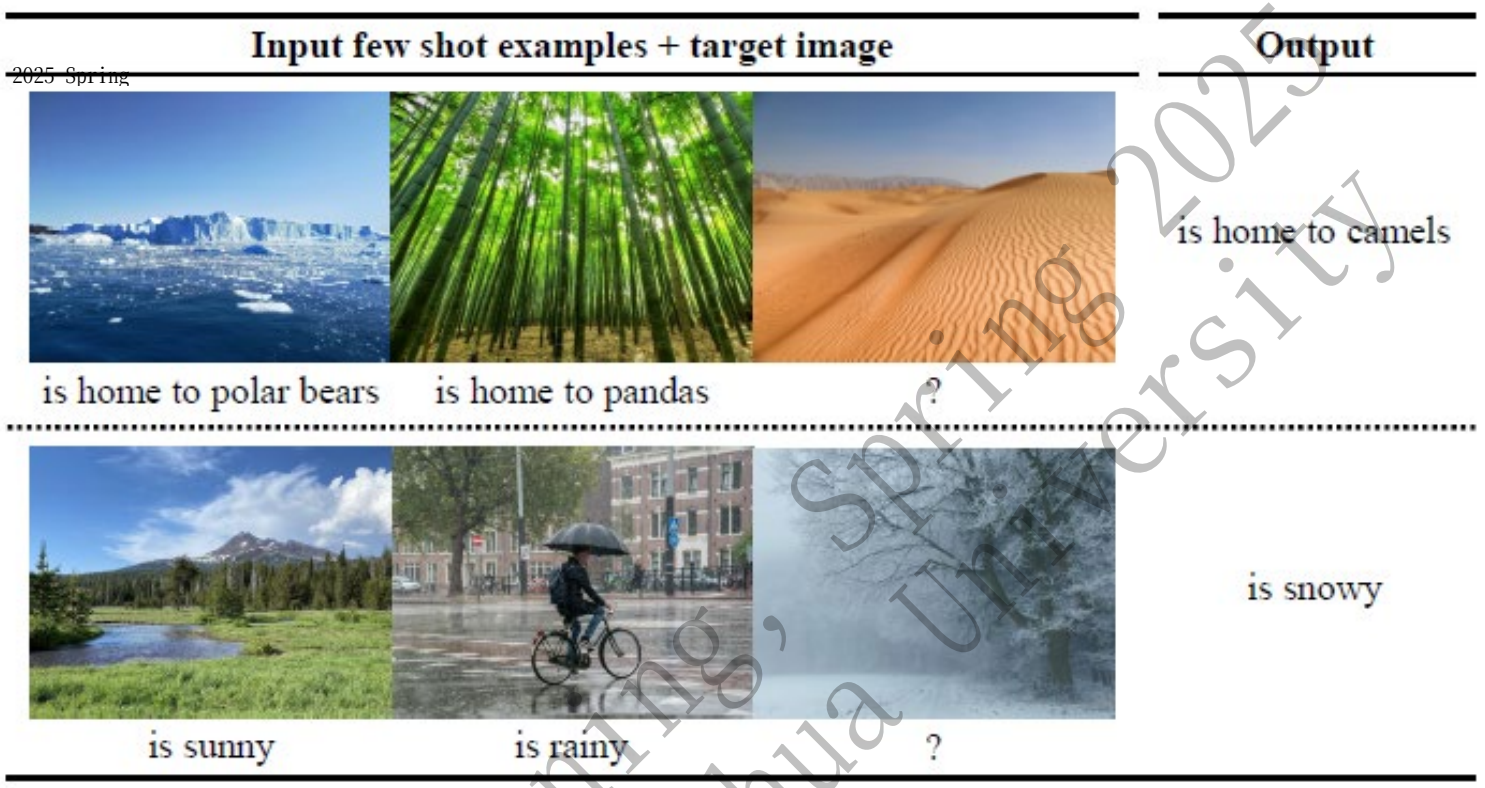
Unified Understanding and Generation

- VILA-U (Nvidia, ICLR 2025)
  - A unified multi-modal model with both understanding and generation
  - Special training process for the visual tower
  - Discrete visual tokens for transformer inputs



Unifie n

- VILA-U
  - A un eration
  - Spec
  - Disc



Fireworks exploding in the sky:

Unified Understanding and Generation

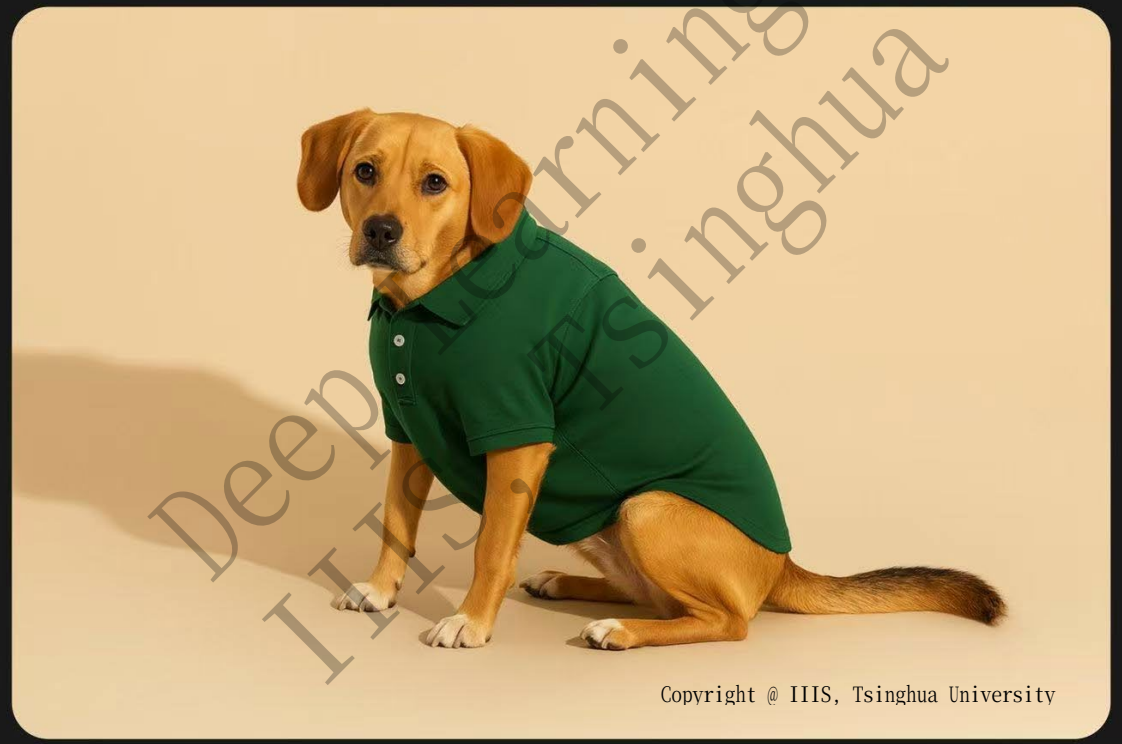
- GPT-4o image generation (OpenAI, 2025.3)
  - A multi-modality model with image-generation capabilities built-in
  - It enables an interactive way of image editing, by language

Unif

- GPT-4
  - A
  - It e

Combine these images

Image created



10:56 5G 93

ChatGPT 4o >

generating image of a miniature of a little penguin wizard in low poly style, refer this picture about the penguin style

图片已创建

给 ChatGPT 发送消息

搜索 深入研究

10:56 5G 93

ChatGPT 4o >

Rotate the miniature and see its back

图片已创建

给 ChatGPT 发送消息

搜索 深入研究

生成一张图片，内容是这10个人的可爱小雕像摆在一张浅色的木桌上。照片从45 度往下拍摄，小雕像要使用 business casual 的着装，并且尽可能地准确还原每个人的长相。应该是大头的全身小雕像

Image created

Summary of Structured Latent Space

- Discrete latent space & Gumbel-Softmax trick
  - Ensure diversity and expressiveness
- VQ-VAE and VQ-VAE 2
  - Vector quantization (clustering) for gradient propagation over discrete latent
  - Hierarchical representation for expressiveness + learned prior for generation
- DALL-E series & latent diffusion model
  - dVAE + Transformer prior over large-scale text-image paired data
  - Learning generative model based on pretrained representations
  - It also enables controlled video generation
- Vila-U and GPT-4o Image Gen
  - Unified foundation model allows a new way of interactive editting

Today's Topic

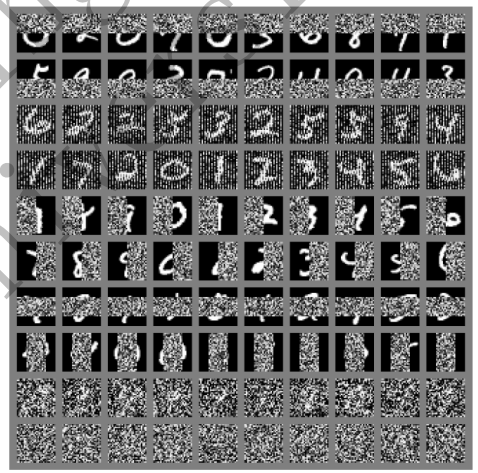
- Structured Priors in Deep Generative Model
  - Discrete/structured latent variables
- Deep Learning for Structured Data
  - Graph neural networks
- Applications

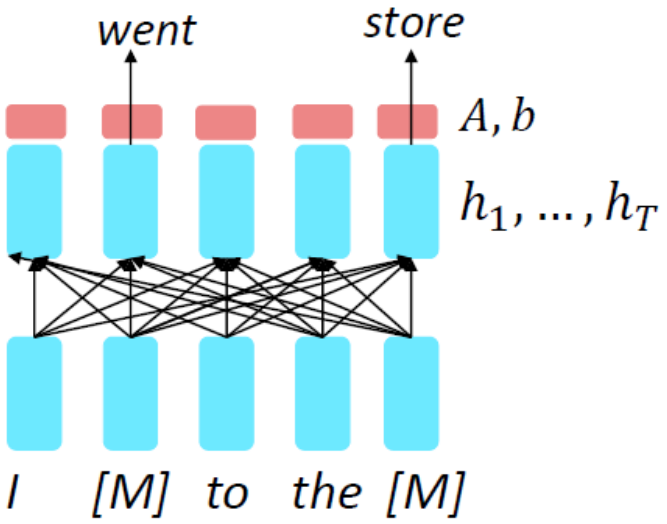
Today's Topic

- Structured Priors in Deep Generative Model
  - Discrete/structured latent variables
- Deep Learning for Structured Data
  - Graph neural networks
- Applications

Recap: Label Prediction Problem

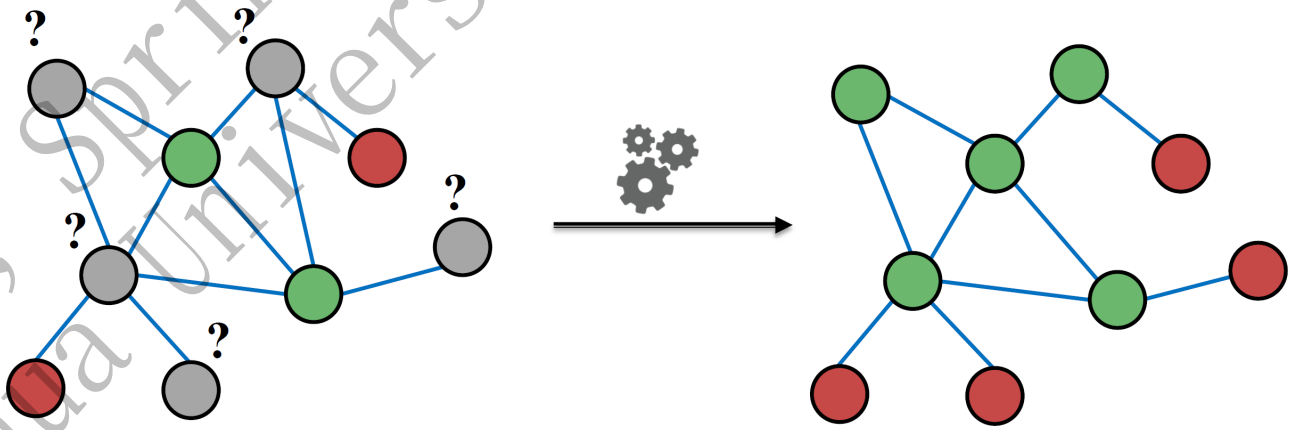
- Image Inpainting (Lecture 4~7)
  - Fill the missing part of the image
  - Conditioned sampling
    - EBM/FLOW/VAE/BiGAN
- Sequence Labeling (Lecture 9)
  - Predict the unknown tokens
  - Masked Language Model (BERT)

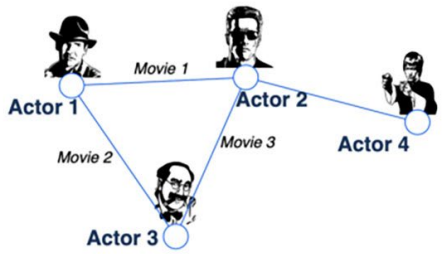


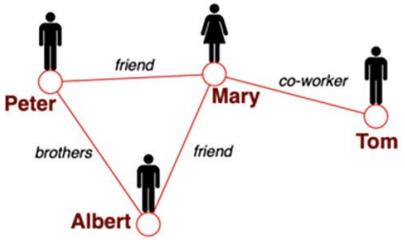


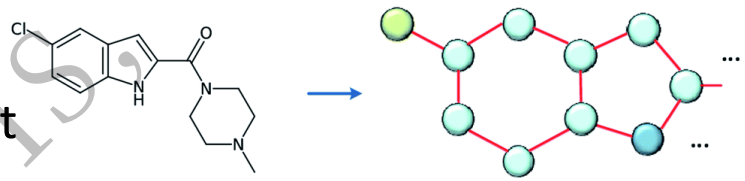
Recap: Label Prediction Problem

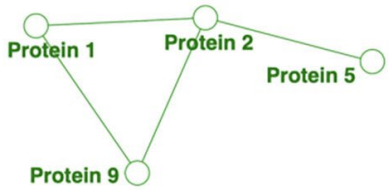
- Image Inpainting (Lecture 4~7)
  - Fill the missing part of the image
  - Conditioned sampling
    - EBM/FLOW/VAE/BiGAN
- Sequence Labeling (Lecture 9)
  - Predict the unknown tokens
  - Masked Language Model (BERT)
- Graph Prediction?
  - Relationships
    - Treatment-patient
    - Friendship
  - Drug discovery

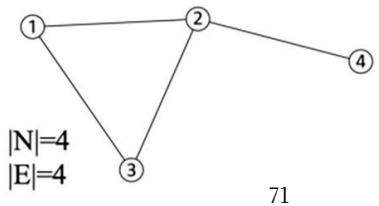






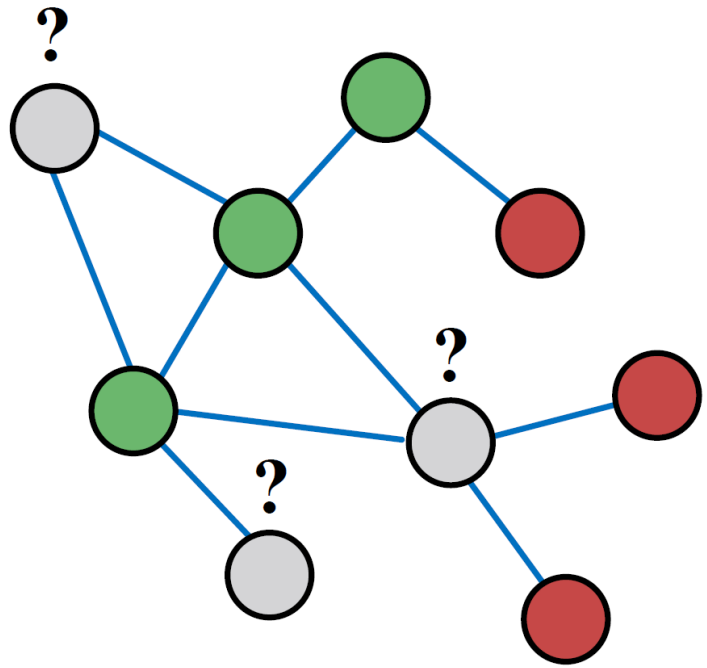






Graph Learning

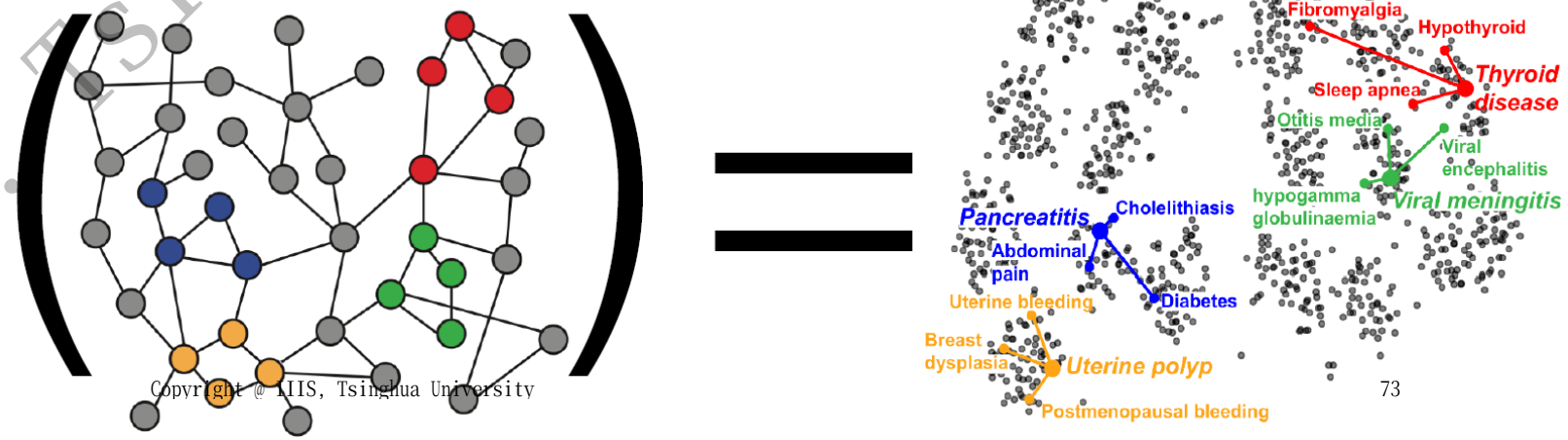
- Graph Labeling
  - A graph $G = (V, N(V))$
    - Adjacency matrix A
    - N_i: neighbors of node i
  - Node features x_i
    - User profile
    - Webpage content
  - Label: $y_i = \{0, 1, ?\}$
  - Task: predict labels for unknown nodes



- Homophily Assumption
  - The tendency of individuals to associate and bond with similar others
  - For edge $< u, v >$, u and v should be "similar" or "highly associated"

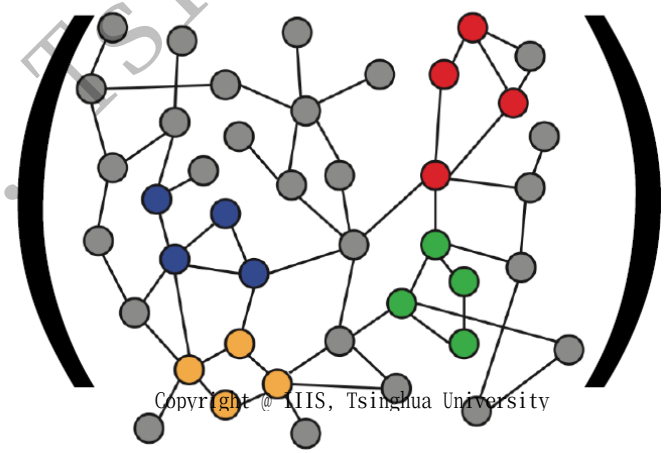
Graph Learning

- Graph Labeling Task
  - Given $G = (V, N(V)), X, Y$ (partial labels)
  - Homophily assumption: associated nodes connect to each other
- Goal: representation learning
  - Learn a embedding z_i for each node, such that similar nodes have similar z_i
    - $Z = f(G, X; \theta)$
  - $y_i = g(z_i; \theta)$

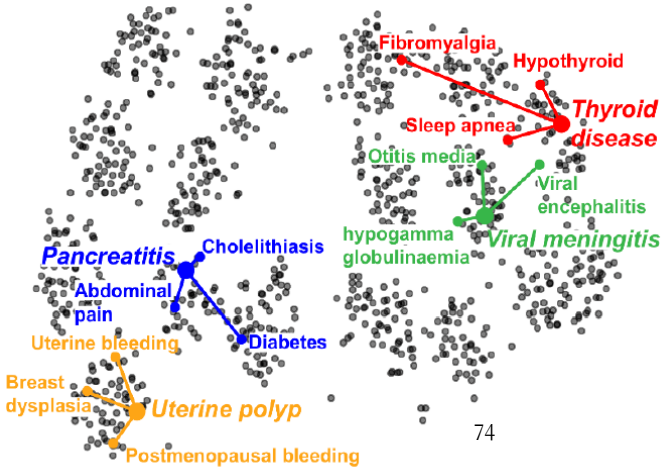


Graph Learning

- Graph Labeling Task
  - Given $G = (V, N(V)), X, Y$ (partial labels)
  - Homophily assumption: associated nodes connect to each other
- Goal: representation learning
  - Learn a embedding z_i for each node, such that similar nodes have similar z_i
    - $Z = f(G, X; \theta)$
  - $y_i = g(z_i; \theta)$
- How to learn f?

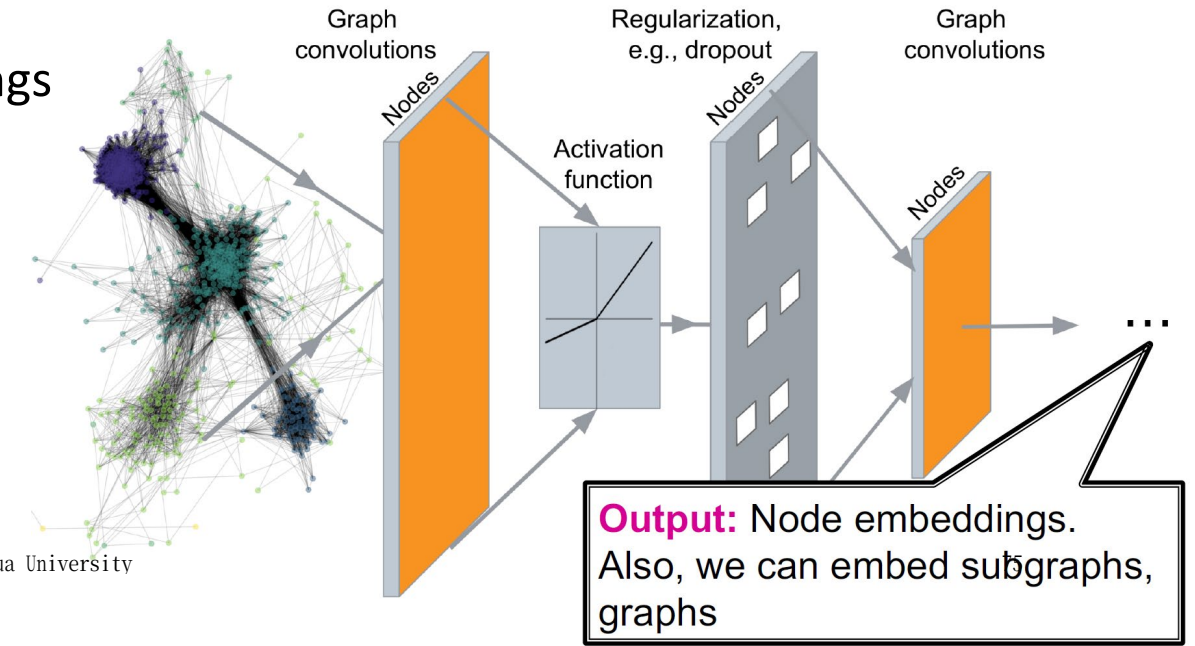


=



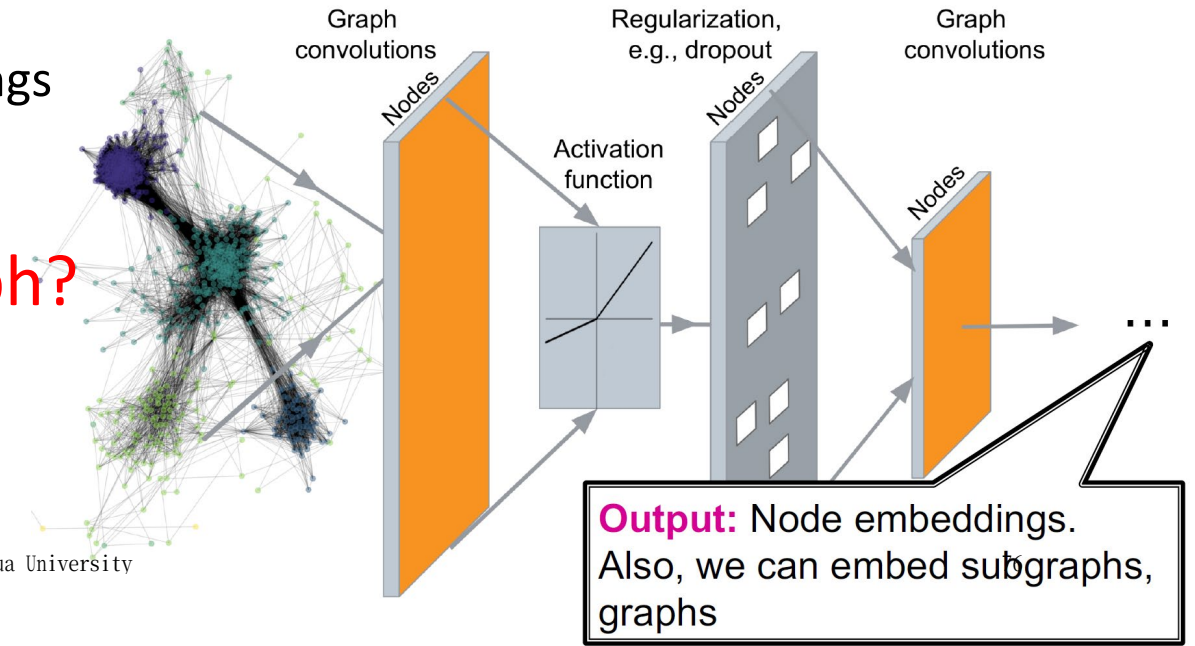
Graph Learning

- Representation Learning
  - Given $G = (V, N(V)), X, Y$ (partial labels)
  - Goal: learn Z where z_i is the representation of node i in G
- Graph Convolutional Networks (GCNs, Kipf & Welling, ICLR 2017)
  - Key idea
    - Convolution on graphs for node embeddings



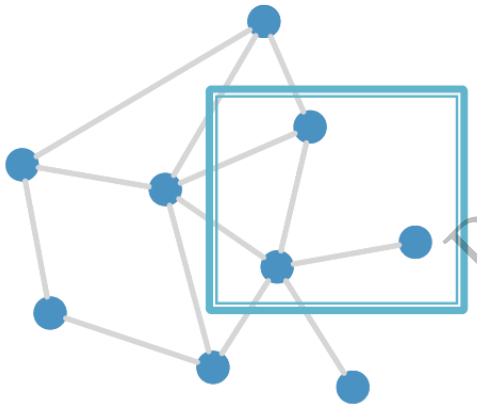
Graph Learning

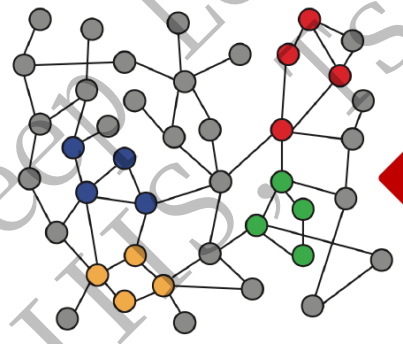
- Representation Learning
  - Given $G = (V, N(V)), X, Y$ (partial labels)
  - Goal: learn Z where z_i is the representation of node i in G
- Graph Convolutional Networks (GCNs, Kipf & Welling, ICLR 2017)
  - Key idea
    - Convolution on graphs for node embeddings
- How to perform convolution on graph?

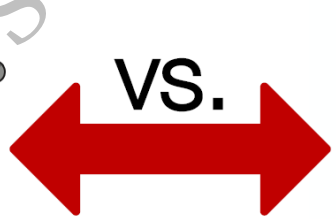


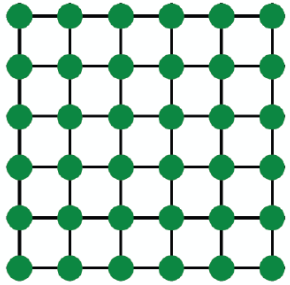
Graph Convolution

- Graph v.s. Image/Sequence
  - Arbitrary size and topological structures (no spatial locality)
  - No fixed node ordering (permutation invariant)
  - Can be dynamic and extremely large
- A convolutional layer operates on locally connected vertices!
  - *Let's convert spatial convolution to graph convolution*



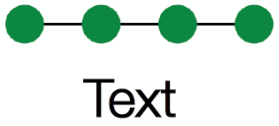


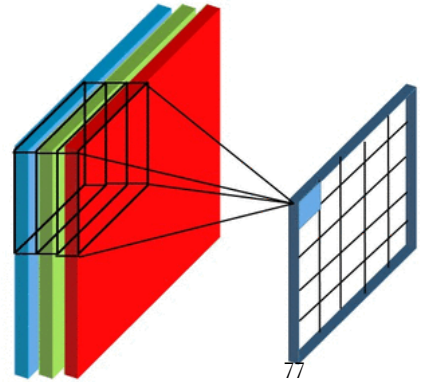




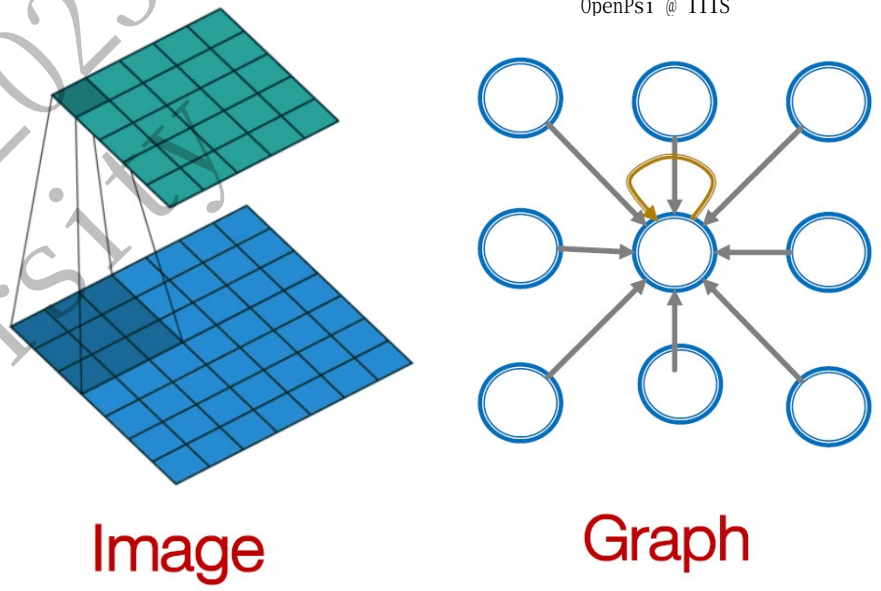
Networks

Images

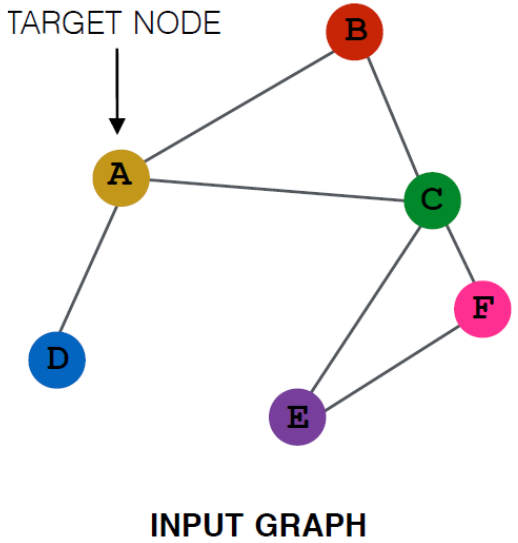




Graph Convolution

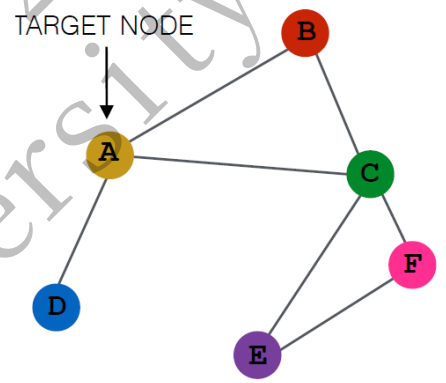


- Spatial Convolution to Graph Convolution
  - Example: 3x3 convolution kernel
  - Spatial Conv: Order sensitive
  - Graph Conv: Order-invariant
- General Solution: *Message Aggregation*
  - Compute a "message" h_j for all the neighbors
    - $h_j = f(x_j)$
  - Aggregate the messages
    - E.g., Average pooling
    - $z_i = \frac{1}{|N(i)|}\sum_{j \in N(i)} h_j$
  - Intuition
    - Propagate information from neighbors

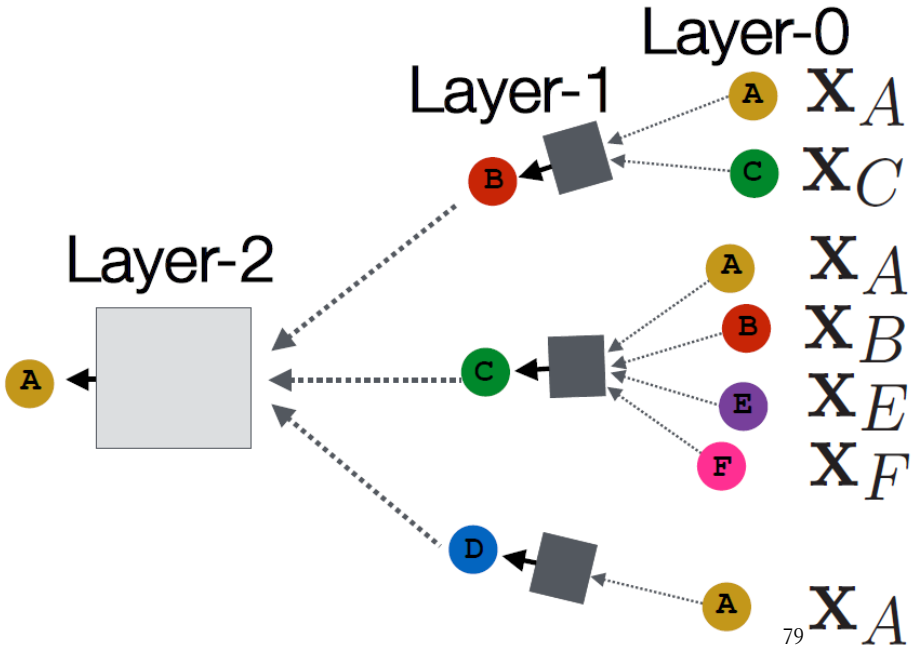


Graph Convolution

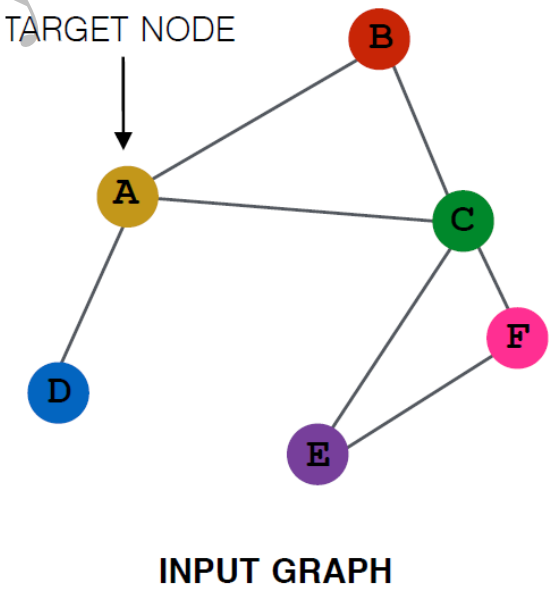
- General Solution: *Message Aggregation*
  - Compute a "message" h_j for all the neighbors
    - $h_j = f(x_j)$
  - Aggregate the messages
    - $z_i = \frac{1}{|N(i)|}\sum_{j \in N(i)} h_j$
  - ***Multi-layer aggregation***
    - Layer-0: embedding of i on x_i
    - Layer-k: embedding from nodes k hops away



INPUT GRAPH

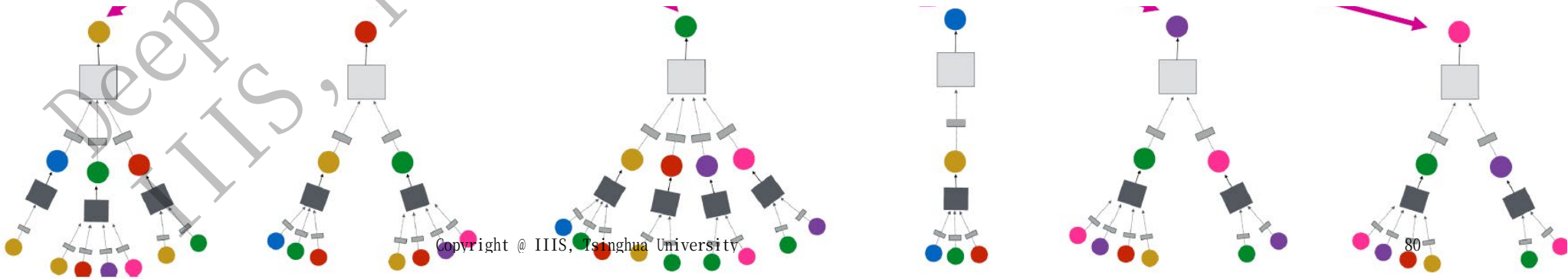


Graph Convolution



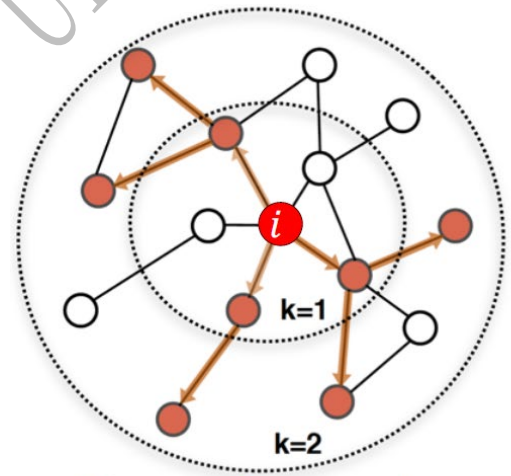
INPUT GRAPH

- General Solution: *Message Aggregation*
  - Compute a "message" h_j for all the neighbors
    - $h_j = f(x_j)$
  - Aggregate the messages
    - $z_i = \frac{1}{|N(i)|}\sum_{j \in N(i)} h_j$
- Perform local message aggregation for every node.
  - *Each node leads to a rooted computation tree*

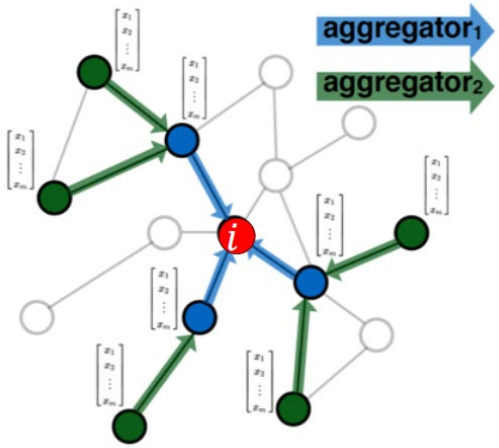


Graph Convolution Network

- General Solution: *Message Aggregation*
  - Compute a "message" h_j for all the neighbors
    - $h_j = f(x_j)$
  - Aggregate the messages
    - $z_i = \frac{1}{|N(i)|}\sum_{j \in N(i)} h_j$
- Graph Convolution Network (GCN)
  - Graph as a computation topology
  - *Message aggregations through edges*



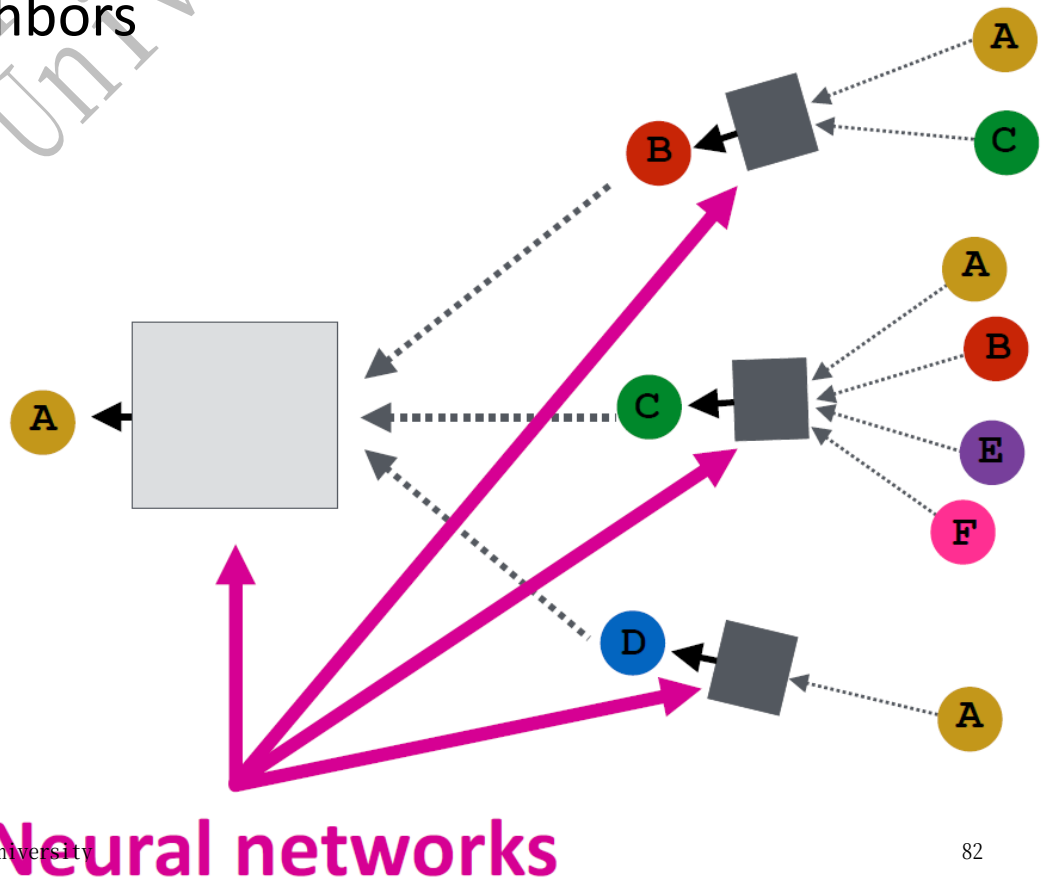
Determine node computation graph



Propagate and transform information

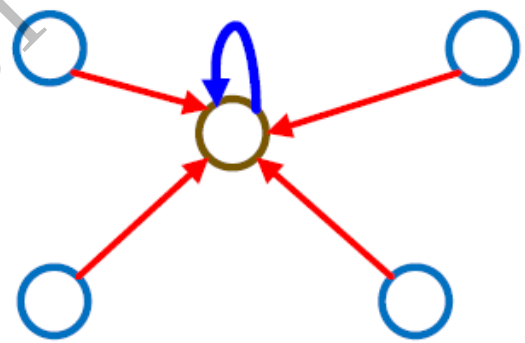
Graph Convolution Network

- General Solution: *Message Aggregation*
  - Compute a "message" h_j for all the neighbors
    - $h_j = f(x_j)$
  - Aggregate the messages
    - $z_i = \frac{1}{|N(i)|}\sum_{j \in N(i)} h_j$
- Graph Convolution Network (GCN)
  - Graph as a computation topology
  - *Message aggregations through edges*
  - *Use neural network for representations!*
    - *Let's mathematically formulate GCN*

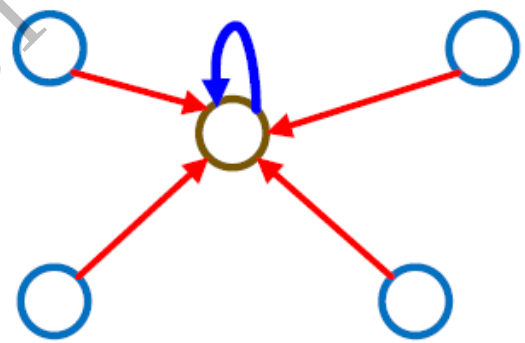


Graph Convolution Network

- Graph Convolution
  - $Z = GC(H, G)$ (G: graph; H: embeddings)
  - $z_i = \sigma(W \sum_{j \in N(i)} \frac{h_j}{|N(i)|} + B \cdot h_i)$

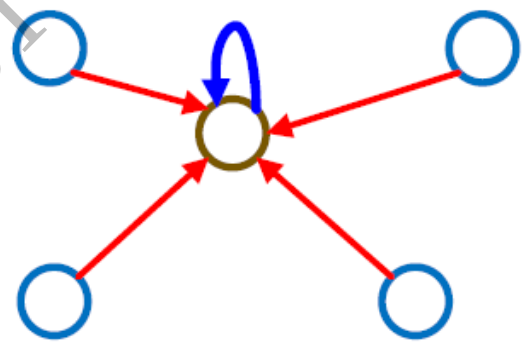


Graph Convolution Network



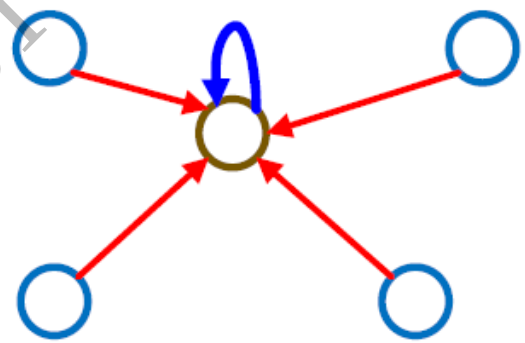
- Graph Convolution
  - $Z = GC(H, G)$ (G: graph; H: embeddings)
  - $z_i = \sigma(W \sum_{j \in N(i)} \frac{h_j}{|N(i)|} + B \cdot h_i)$
  - σ: activation (leaky ReLU or ReLU)
  - Parameters: W for neighbor messages & B for self transformation

Graph Convolution Network

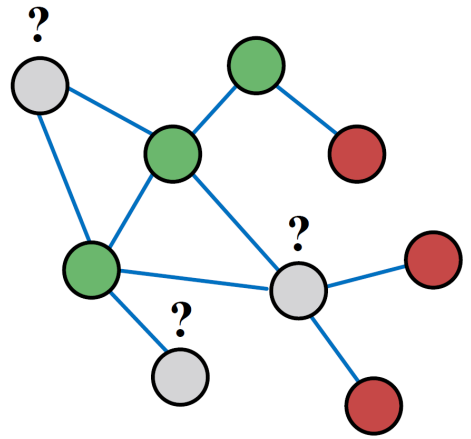


- Graph Convolution
  - $Z = GC(H, G)$ (G: graph; H: embeddings)
  - $z_i = \sigma(W \sum_{j \in N(i)} \frac{h_j}{|N(i)|} + B \cdot h_i)$
  - σ: activation (leaky ReLU or ReLU)
  - Parameters: W for neighbor messages & B for self transformation
- Matrix Form
  - A: adjacency matrix (sparse)
  - Degree matrix: $D = \mathrm{diag}(|N(i)|)$
  - $\tilde{A} = D^{-1}A$
  - $Z = \sigma(\tilde{A}HW^T + HB^T)$

Graph Convolution Network

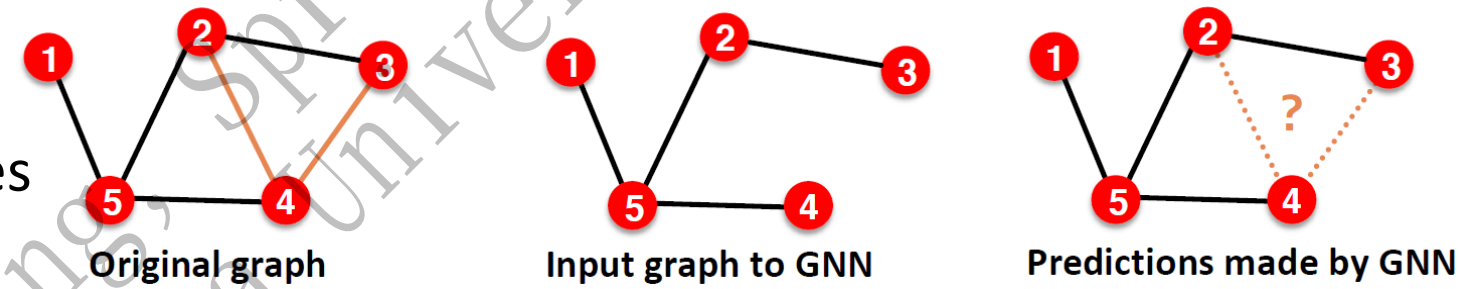


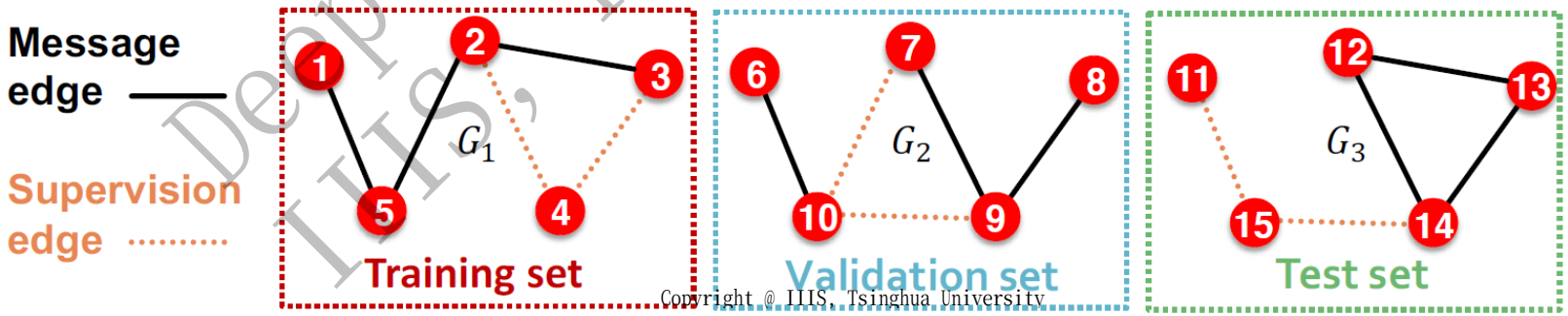
- Graph Convolution
  - $\tilde{A} = D^{-1}A$
  - $Z = GC(G, H) = \sigma\left(\tilde{A}HW^T + HB^T\right)$
  - Parameters: W for neighbor messages & B for self transformation
- **GCN (Kipf & Welling, ICLR 2017)**
  - $H^{(l)} = GC\left(G, H^{(l-1)}\right)$ for $0 < l \leq L$
  - $H^{(0)} = X; \quad Z = H^{(L)}$
  - Learning objectives
    - Node label prediction
    - $L(\theta) = \sum_i CE(g(z_i), y_i)$ for node i with target label y_i
    - *What about edge prediction?*
      - *Dataset partitioning? What's the input to the model? (GCN operates on edges)*



Graph Convolution Network

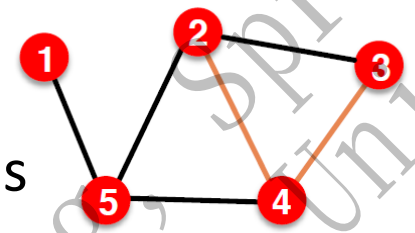
- Edge Prediction Problem
  - Predict missing edges
  - Partition edges into two types
    - Message edges: input to GCN
    - Supervision edges: prediction label *(DO NOT take it as GCN input)*
  - Dataset Creation (Train/Valid/Test)
    - Option 1: Inductive link prediction split
      - Split the graph into disjoint sub-graphs *(cross-graph edges are ignored)*



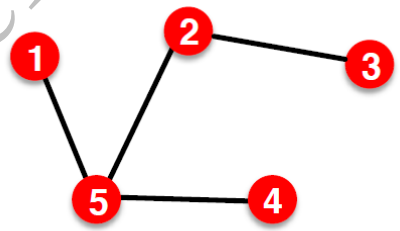


Graph Convolution Network

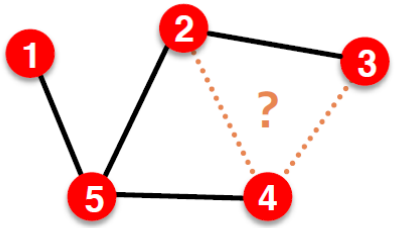
- Edge Prediction Problem
  - Predict missing edges
  - Partition edges into two types
    - Message edges: input to GCN
    - Supervision edges: prediction label *(DO NOT take it as GCN input)*
  - Dataset Creation (Train/Valid/Test)
    - Option 1: Inductive link prediction split
    - Option 2: Transductive link prediction split
      - *Model always observes all the nodes (more popular)*



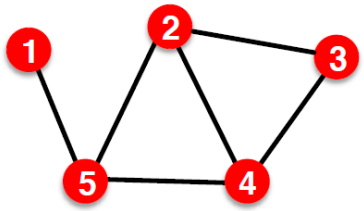
Original graph



Input graph to GNN

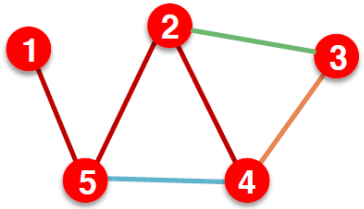


Predictions made by GNN



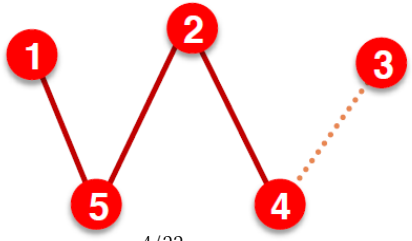
The original graph

Split

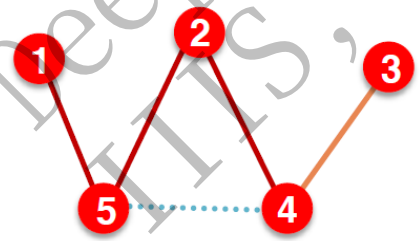


Split Graph with
4 types of edges

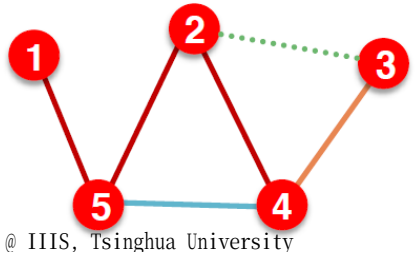
Training message edges
Training supervision edges
Validation edges
Test edges



(1) At training time:



(2) At validation time:

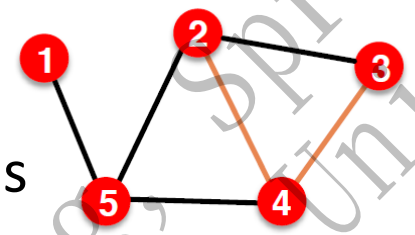


(3) At test time:

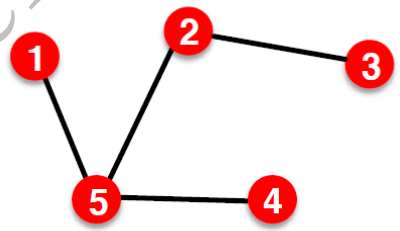
Find more at https://github.com/snap-stanford/GraphGym

Graph Convolution Network

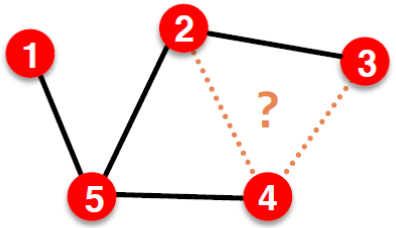
- Edge Prediction Problem
  - Predict missing edges
  - Partition edges into two types
    - Message edges: input to GCN
    - Supervision edges: prediction label *(DO NOT take it as GCN input)*
  - Dataset Creation (Train/Valid/Test)
    - Option 1: Inductive link prediction split
    - Option 2: Transductive link prediction split
    - **You also need negative samples!**
      - Random samples can be effective in many cases
      - Also a lot of heuristics to select challenging negative samples in industrial applications



Original graph



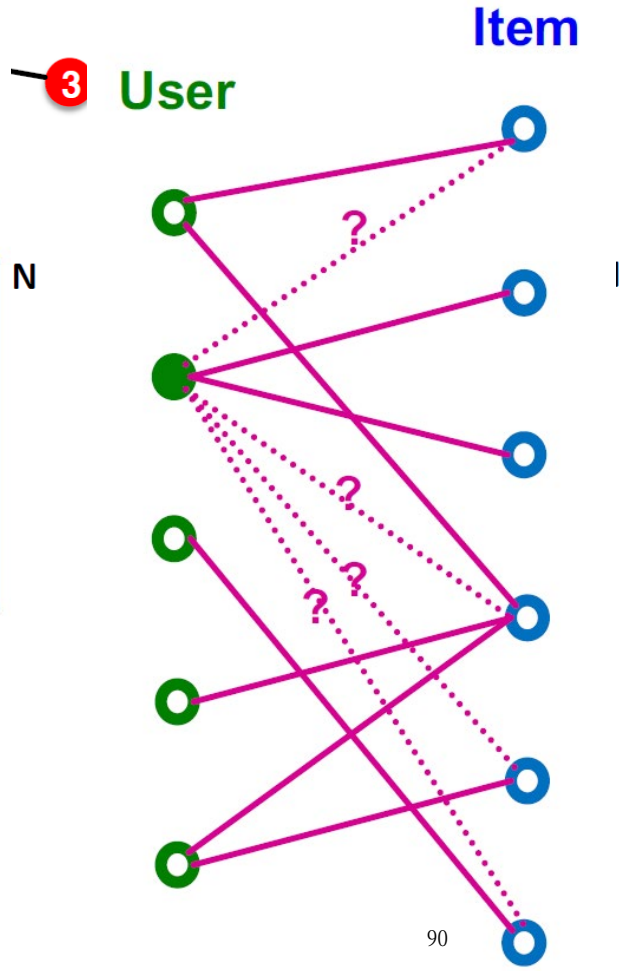
Input graph to GNN



Predictions made by GNN

Graph Convolution Network

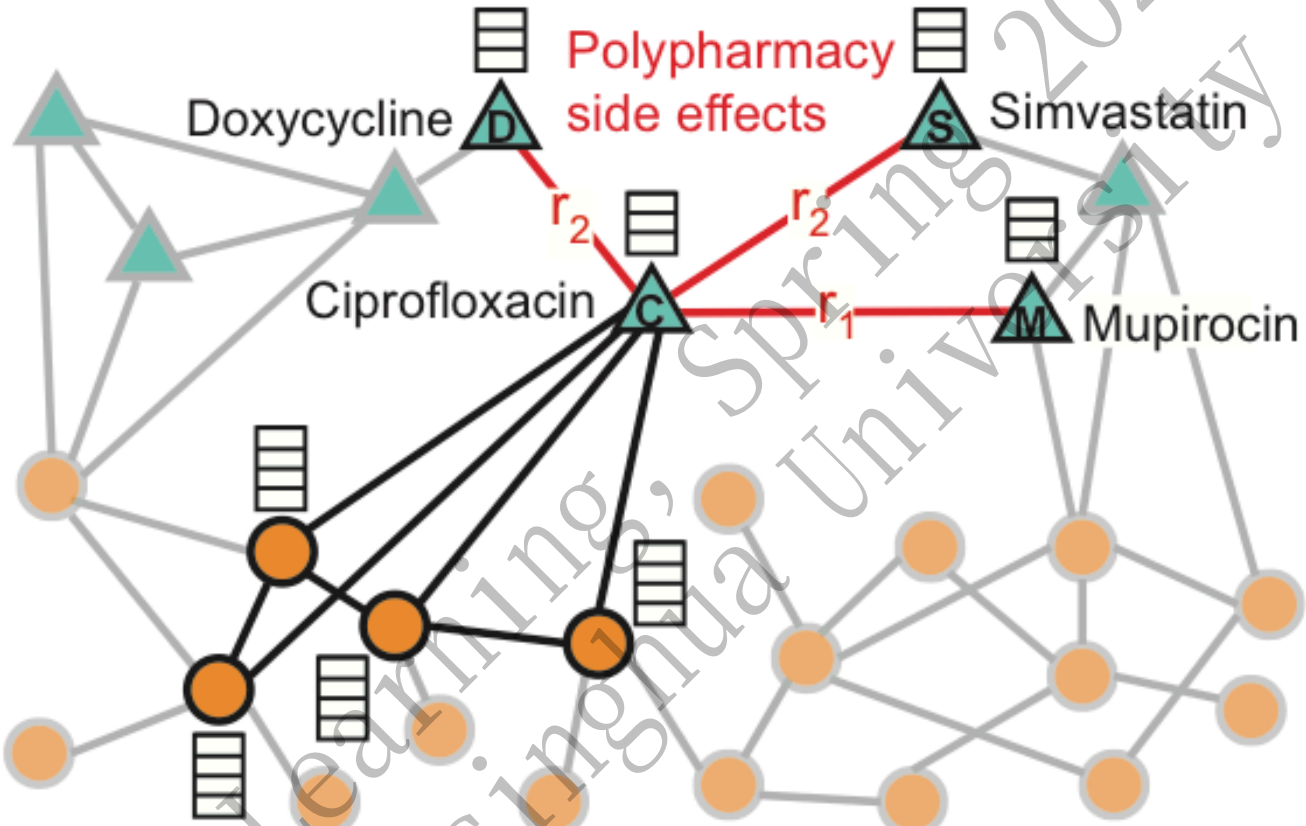
- |

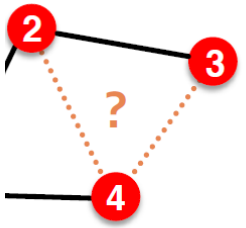


- You also need negative samples!
- Example
  - **Recommendation System**

Gr

- Ec



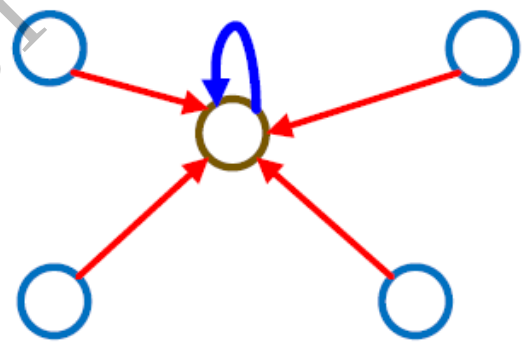


ns made by GNN

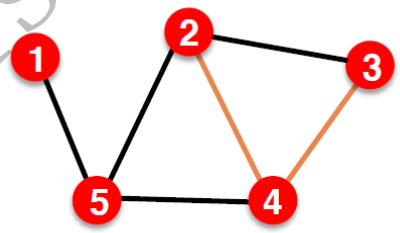
Drug Protein
Node feature vector
r_1 Gastrointestinal bleed side effect
r_2 Bradycardia side effect
Drug-protein interaction
Protein-protein interaction

- **Drug Discovery**

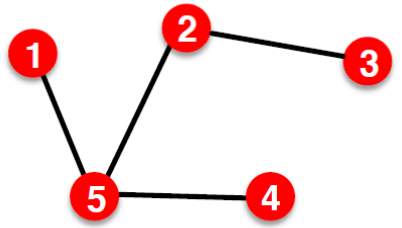
Graph Convolution Network



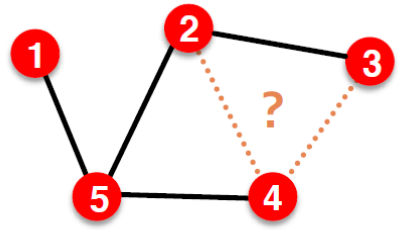
- Graph Convolution
  - $\tilde{A} = D^{-1}A$
  - $Z = GC(G, H) = \sigma\left(\tilde{A}HW^T + HB^T\right)$
  - Parameters: W for neighbor messages & B for self transformation
- GCN (Kipf & Welling, ICLR 2017)
  - $H^{(l)} = GC\left(G, H^{(l-1)}\right)$ for $0 < l \leq L$
  - $H^{(0)} = X; \quad Z = H^{(L)}$
  - Learning objectives
    - Node label prediction
    - *Edge prediction*



Original graph

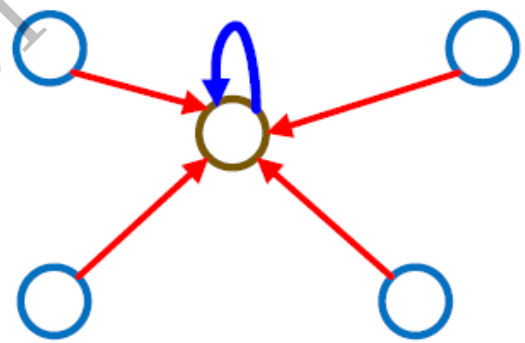


Input graph to GNN

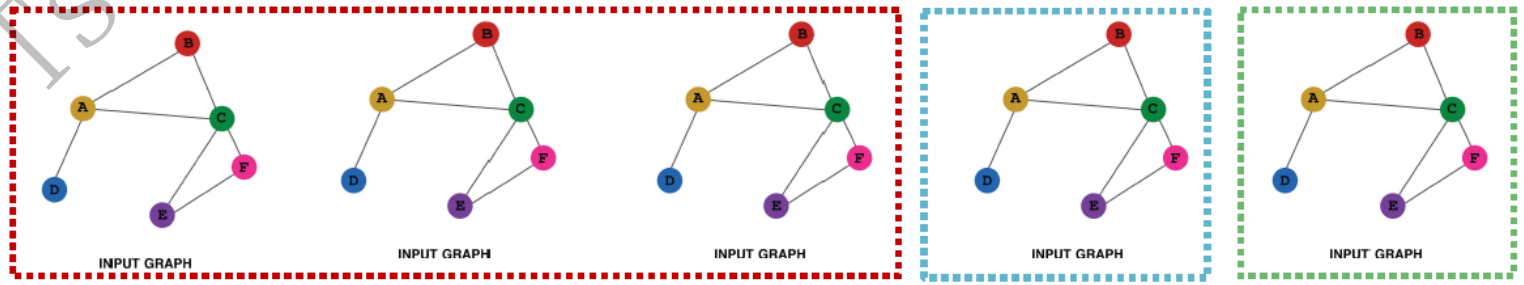


Predictions made by GNN

Graph Convolution Network

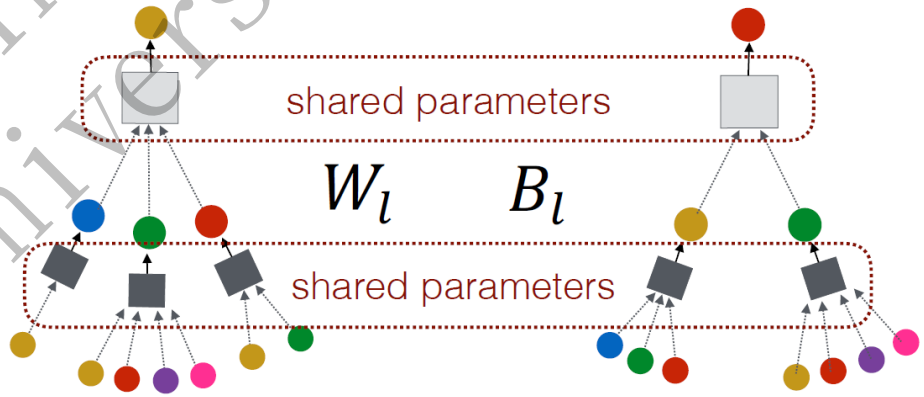


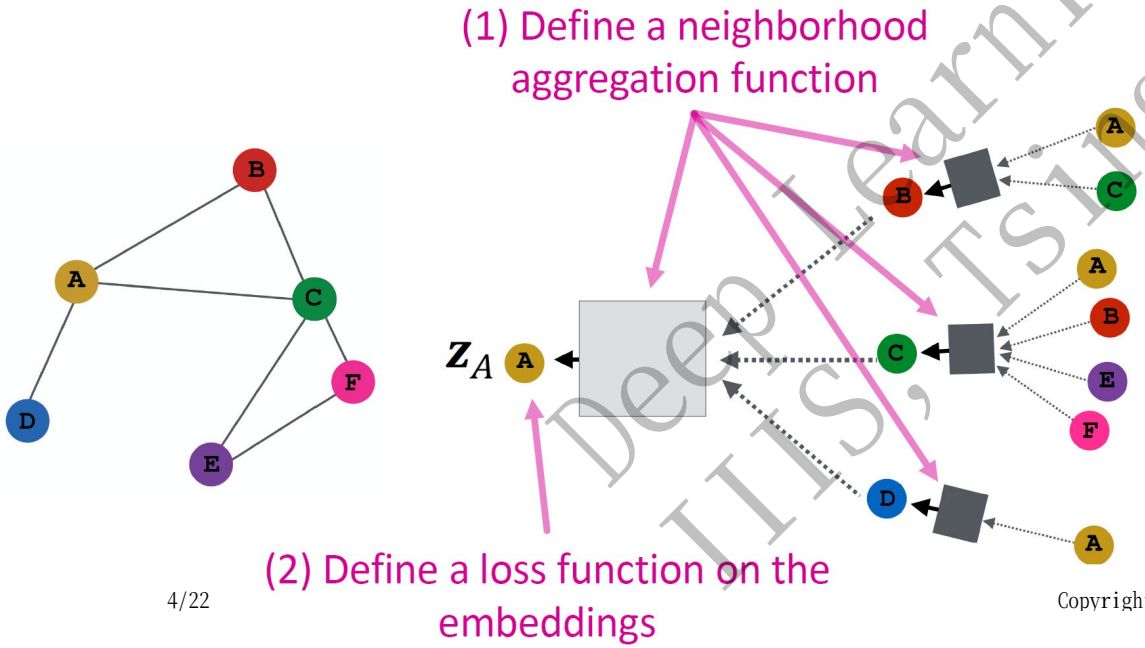
- Graph Convolution
  - $\tilde{A} = D^{-1}A$
  - $Z = GC(G, H) = \sigma\left(\tilde{A}HW^T + HB^T\right)$
  - Parameters: W for neighbor messages & B for self transformation
- GCN (Kipf & Welling, ICLR 2017)
  - $H^{(l)} = GC\left(G, H^{(l-1)}\right)$ for $0 < l \leq L$
  - $H^{(0)} = X; \quad Z = H^{(L)}$
  - Learning objectives
    - Node label prediction
    - Edge prediction
    - *Graph classification*
      - $h_G = AGG(H)$ *(e.g.,* $h_G = \frac{1}{N}\sum h_i$*)*

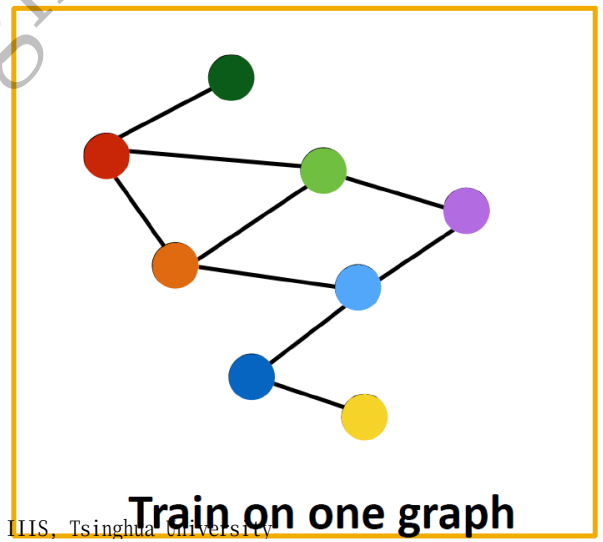


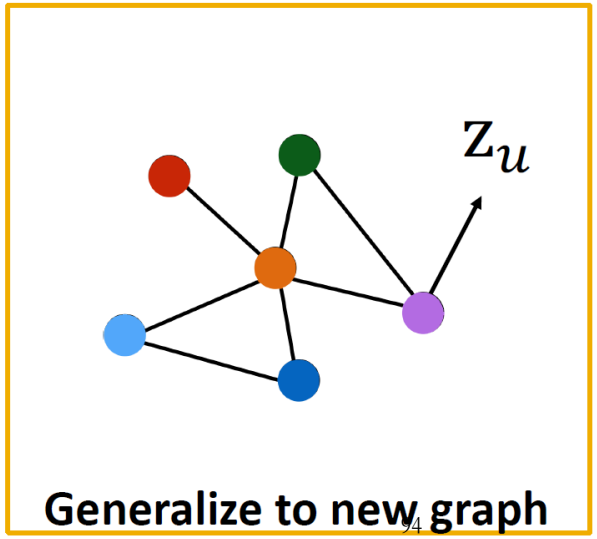
Graph Convolution Network

- GCN Workflow
  - Define aggregation & loss function
  - Train on given graphs w. shared params
  - Generalize to unseen graphs



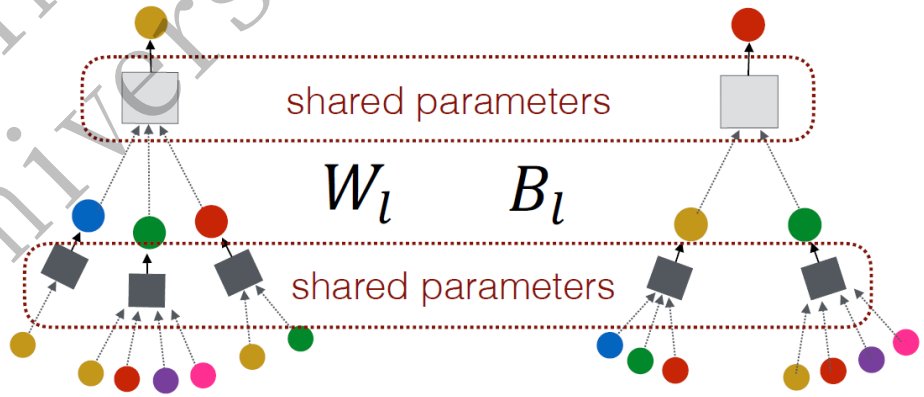


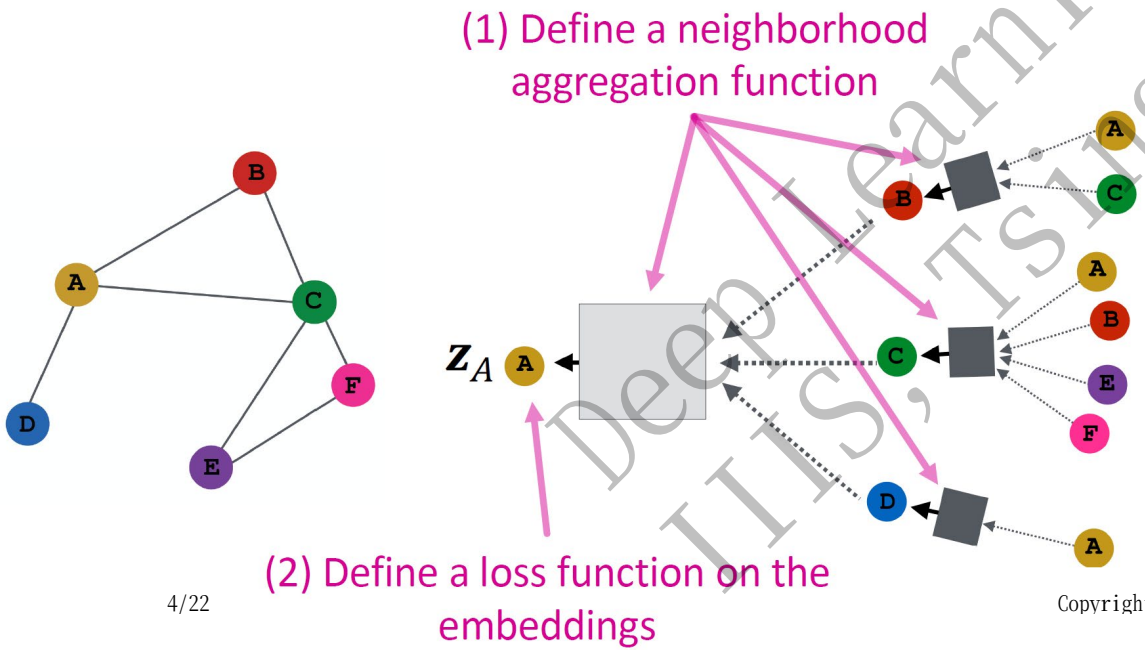


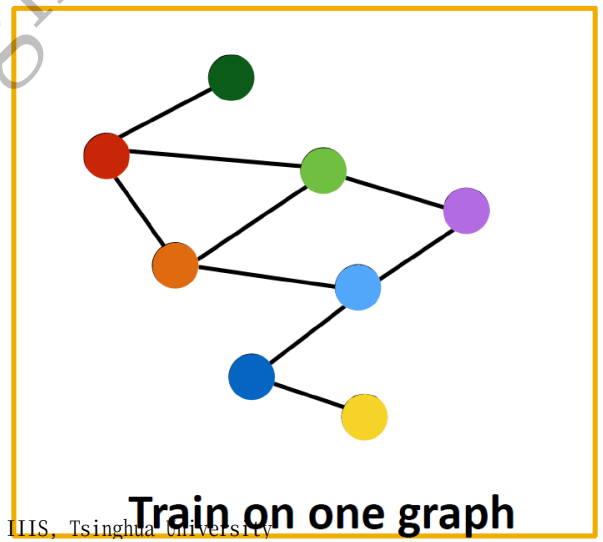


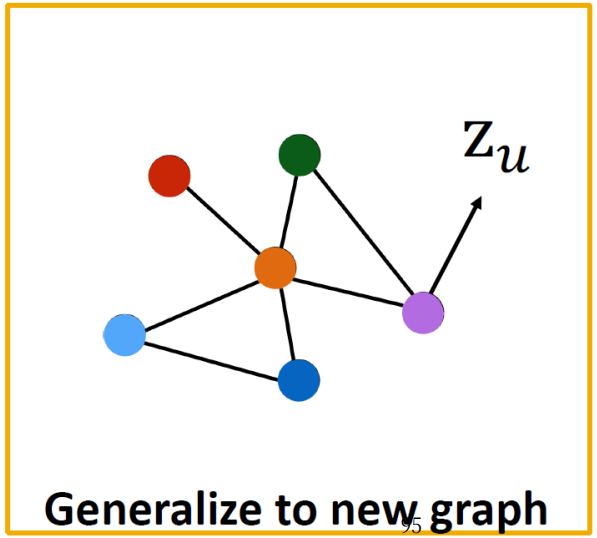
Graph Convolution Network

- GCN Workflow
  - Define ***aggregation*** & loss function
  - Train on given graphs w. shared params
  - Generalize to unseen graphs



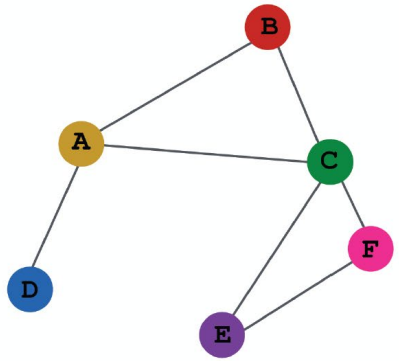


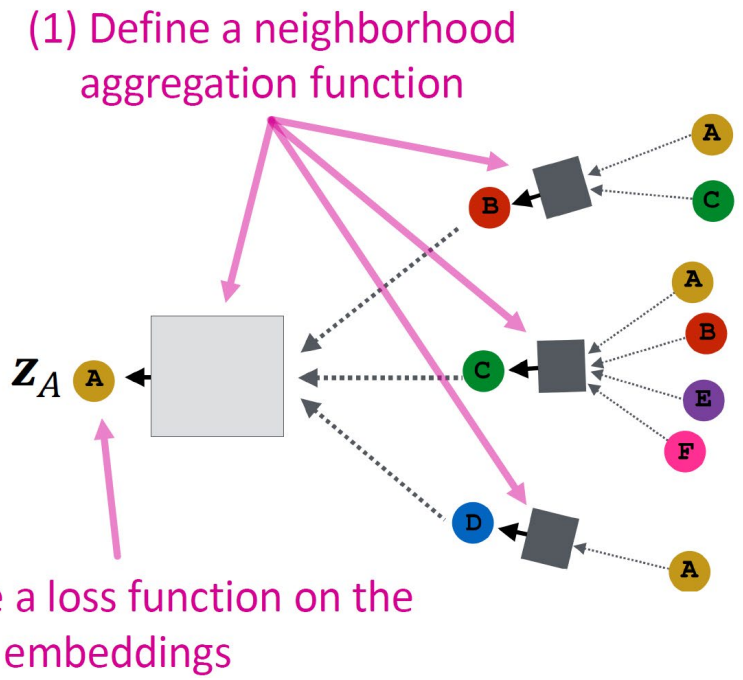




Graph Convolution Network

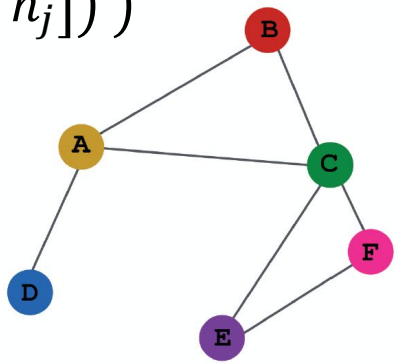
- GraphSage (Stanford, Jure Leskovec group, NIPS 2017)
  - More general aggregation functions than vanilla GCN!
  - $h_i^{(l)} = \sigma\left(W^T\left[h_i^{(l-1)}, AGG\left(\left\{h_j^{(l-1)} \middle| j \in N(i)\right\}\right)\right]\right)$
  - $AGG(\cdot)$: the aggregation function
    - Average pooling (Standard GCN)
    - MLP + Max-Pooling
      - *Default setting*
    - LSTM + random order-permutation
      - *LSTM can be much slower*
- Practical Trick
  - Layer normalization on h_i
  - Feature & Node dropout

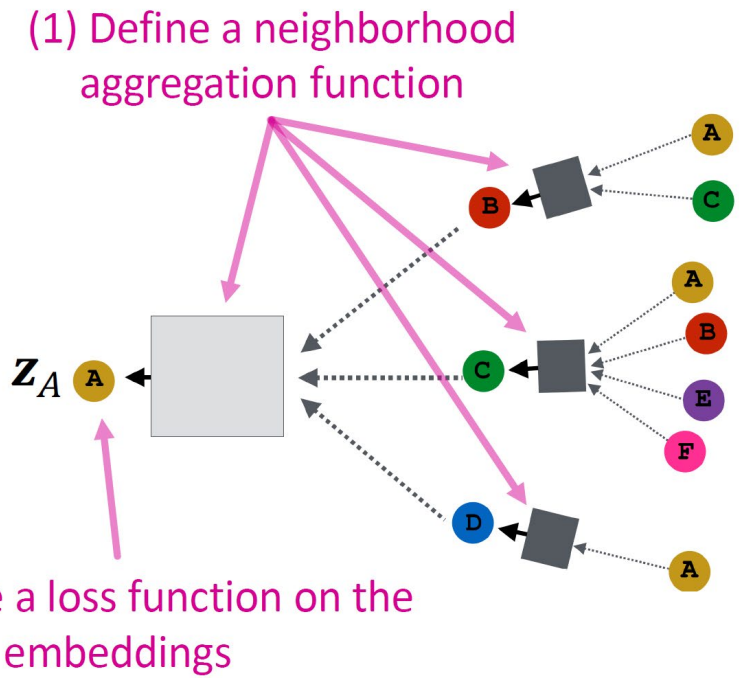




Graph Convolution Network

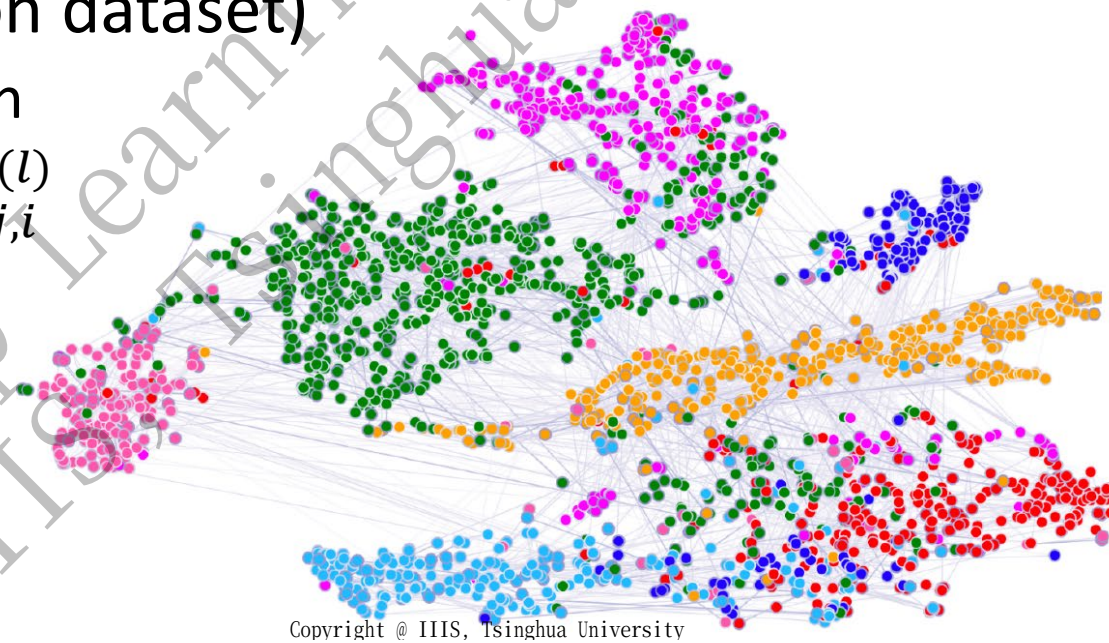
- Graph Attention Network (GAT, Yoshua Bengio group, ICLR 2018)
  - More general aggregation functions than vanilla GCN!
  - $h_i^{(l)} = \sigma\left(W^T\left[ATT\left(h_i^{(l-1)}, \left\{h_j^{(l-1)} \middle| j \in N(i)\right\}\right)\right]\right)$
  - $ATT(\cdot)$: attention over neighbors
    - $\alpha_{ij} = \mathrm{softmax}\left(\mathrm{leakyRELU}\left(q^T\left[W^T h_i, W^T h_j\right]\right)\right)$
    - Also use multi-head attention
- GAT v.s. Transformer
  - Masked attention with A
    - A: adjacency matrix
  - Extremely sparse mask matrix A





Graph Convolution Network

- Graph Attention Network (GAT, Yoshua Bengio group, ICLR 2018)
  - More general aggregation functions than vanilla GCN!
  - $h_i^{(l)} = \sigma\left(W^T\left[ATT\left(h_i^{(l-1)}, \left\{h_j^{(l-1)} \middle| j \in N(i)\right\}\right)\right]\right)$
- Results (Publication dataset)
  - t-SNE visualization
  - Edge: $\sum_l \alpha_{i,j}^{(l)} + \alpha_{j,i}^{(l)}$



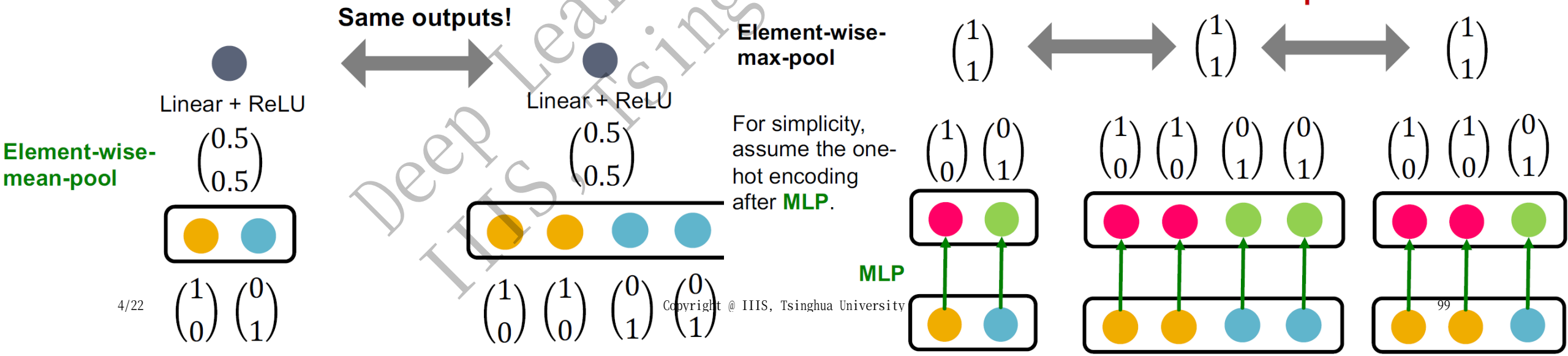
Method	Cora
MLP	55.1%
ManiReg (Belkin et al., 2006)	59.5%
SemiEmb (Weston et al., 2012)	59.0%
LP (Zhu et al., 2003)	68.0%
DeepWalk (Perozzi et al., 2014)	67.2%
ICA (Lu & Getoor, 2003)	75.1%
Planetoid (Yang et al., 2016)	75.7%
Chebyshev (Defferrard et al., 2016)	81.2%
GCN (Kipf & Welling, 2017)	81.5%
GAT	**83.3%**
improvement w.r.t GCN	1.8%

Attention mechanism can be used with many different graph neural network models

In many cases, attention leads to performance gains

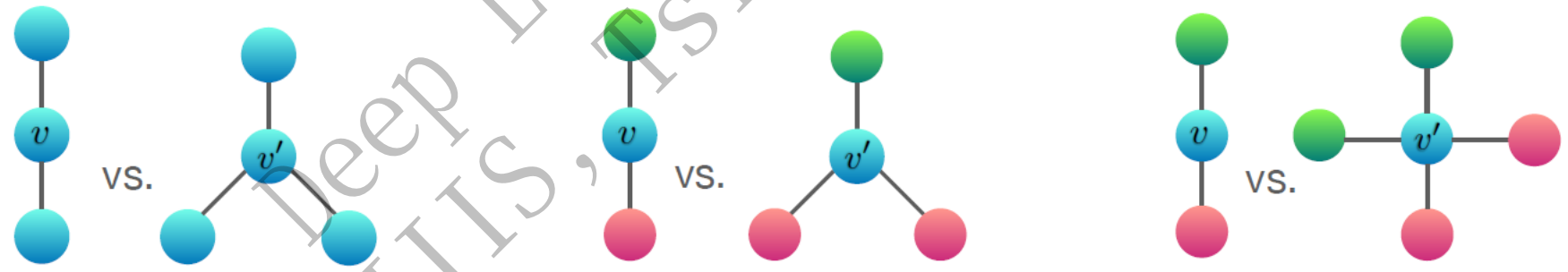
Representation Power

- General Form of GCNs
  - *Pooling* as the aggregation operator
  - Vanilla GCN & GAT: (weighted) average pooling
  - GraphSage: Max pooling
- Expressiveness Issue of Pooling



Representation Power

- Graph Isomorphism Network (GIN, Stanford&MIT, ICLR 2019)
  - *SUM* as the aggregation operator
  - $h_i^{(l)} = MLP^{(l)}\left(\left(1+\epsilon^{(l)}\right)\cdot h_i^{(l-1)} + \sum_{j\in N(i)} h_j^{(l-1)}\right)$
  - Theorem: GIN has the same expressiveness of distinguishing graphs as Weisfeiler-Lehman (WL) graph isomorphism test.
    - Further reading: a general toolbox for analyzing the expressive power of any GNN using tensor language
    - ICLR 2022 outstanding paper: https://openreview.net/forum?id=wIzUeM3TAU



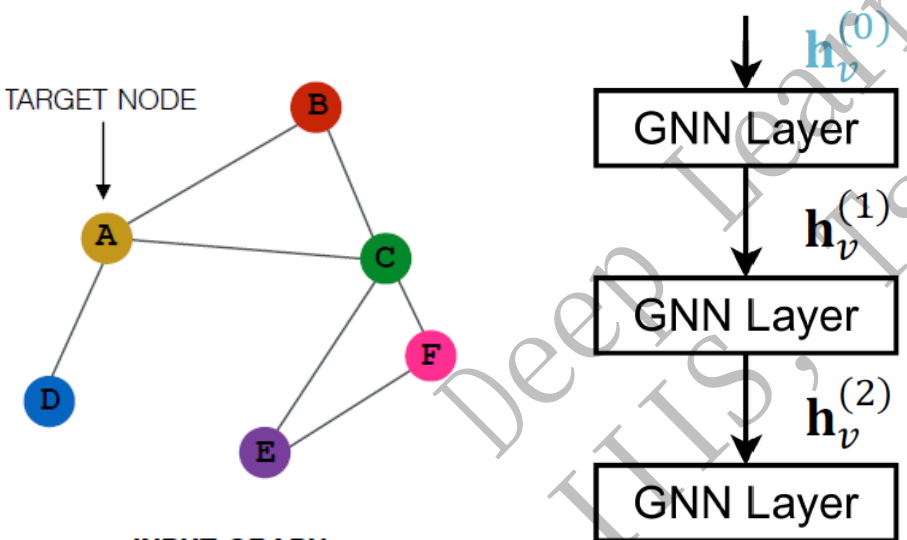
(a) Mean and Max both fail (b) Max fails (c) Mean and Max both fail

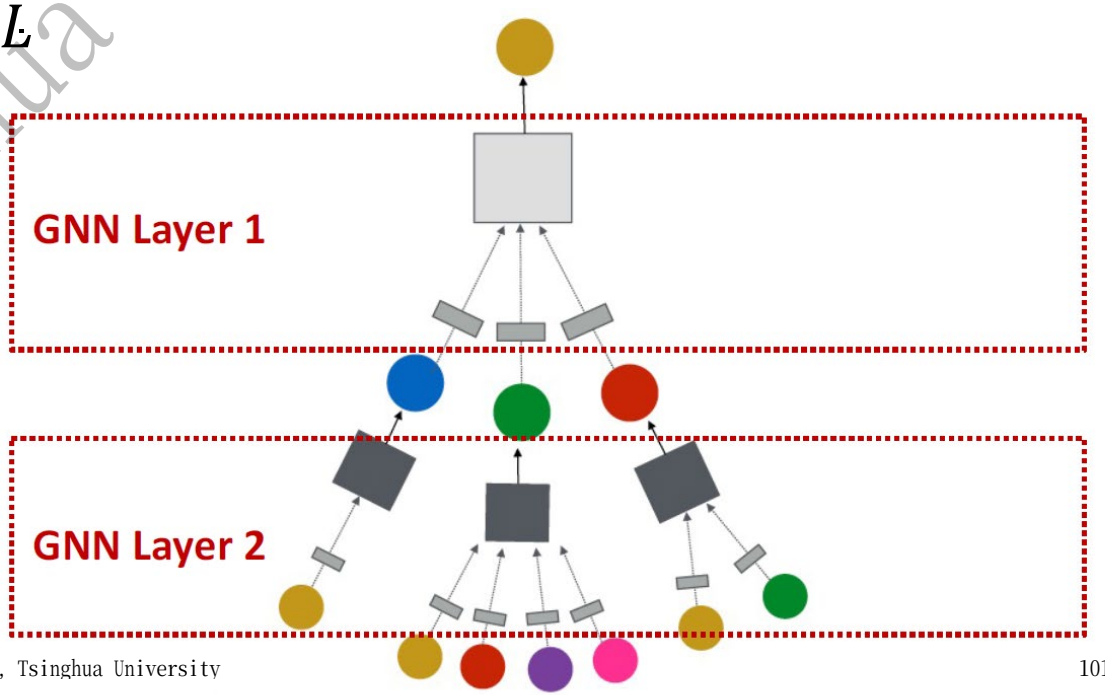
Figure 3: **Examples of graph structures that mean and max aggregators fail to distinguish.**

Representation Power

- Graph Convolution Layer
  - Aggregation function → representation capability
- GCN: stacking multiple graph convolution layers
  - $h^{(l)} = GC^{(l)}\left(h^{(l-1)}; \theta^{(l)}\right)\ 1 \le l \le L$

Q: can we make a really **powerful** GCN by stacking ***a lot of*** GC layers?



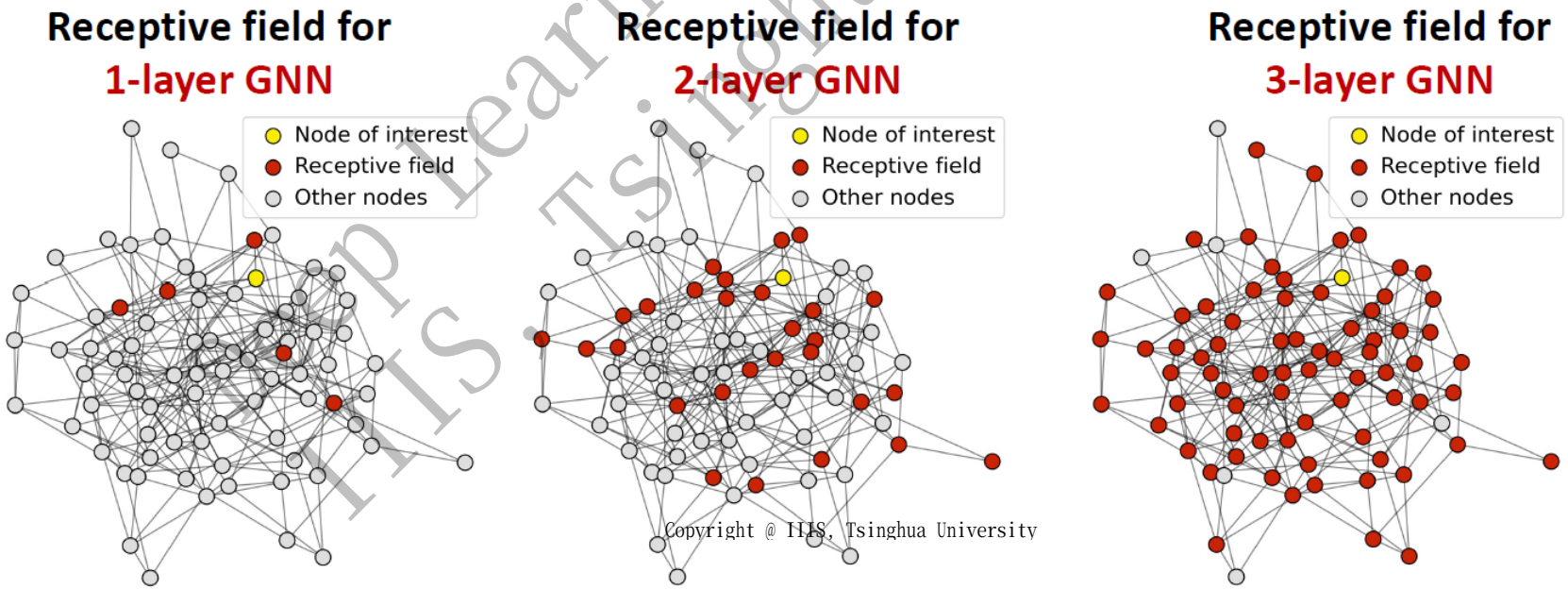


Representation Power

- Graph Convolution Layer
  - Aggregation function → representation capability
- GCN: stacking multiple graph convolution layers
  - $h^{(l)} = GC^{(l)}\left(h^{(l-1)}; \theta^{(l)}\right)\ 1 \leq l \leq L$
  - GCN will suffer from the ***over-smoothing*** issue when L is too large
    - Each node tends to have the same embedding $\forall i, h_i^{(K)} \approx \tilde{h}$
    - *Why?*

Representation Power

- GCN: stacking multiple graph convolution layers
  - $h^{(l)} = GC^{(l)}\left(h^{(l-1)}; \theta^{(l)}\right)\ 1 \leq l \leq L$
  - GCN will suffer from the ***over-smoothing*** issue when L is too large
    - The output node embedding is decided by its *receptive field*
    - When l grows, the shared neighbors quickly grows! → similar output embeddings

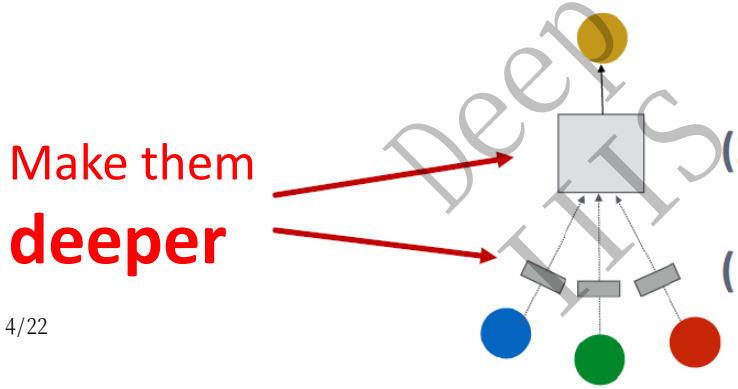


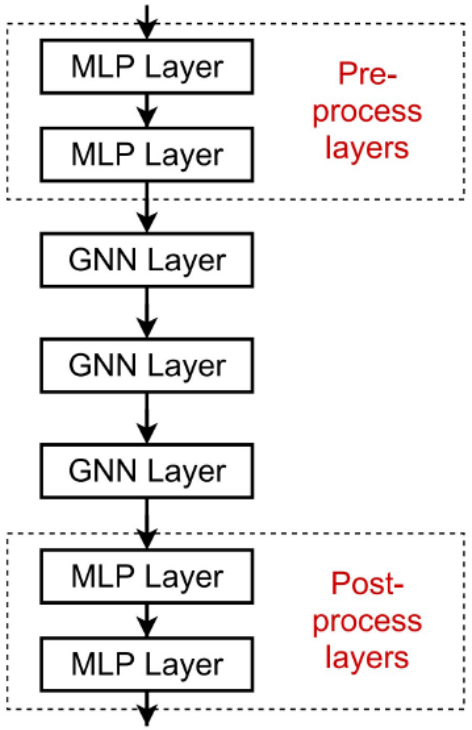
Representation Power

- GCN: stacking multiple graph convolution layers
  - $h^{(l)} = GC^{(l)}\left(h^{(l-1)}; \theta^{(l)}\right)\ 1 \leq l \leq L$
  - GCN will suffer from the ***over-smoothing*** issue when L is too large
  - Practical remarks
    - Tip #1: be cautious when adding more GC layers!
      - *But what if we want to increase the representation power of GCN?*

Representation Power

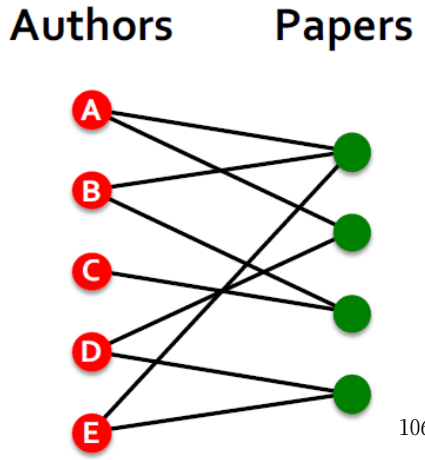
- GCN: stacking multiple graph convolution layers
  - $h^{(l)} = GC^{(l)}\left(h^{(l-1)}; \theta^{(l)}\right) \ 1 \leq l \leq L$
  - GCN will suffer from the ***over-smoothing*** issue when L is too large
  - Practical remarks
    - Tip #1: be cautious when adding more GC layers!
    - Tip #2: increase the capability ***within*** each GC layer;
    - or add pre/post processing layers





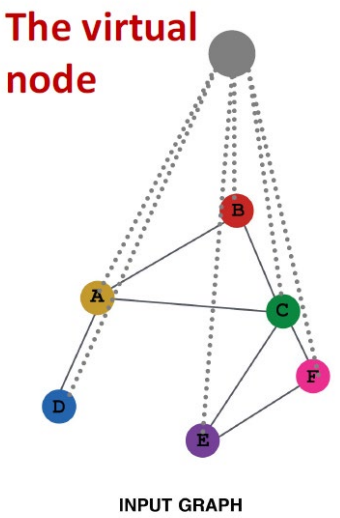
Representation Power

- GCN: stacking multiple graph convolution layers
  - $h^{(l)} = GC^{(l)}\left(h^{(l-1)}; \theta^{(l)}\right)\ 1 \leq l \leq L$
  - GCN will suffer from the ***over-smoothing*** issue when L is too large
  - Practical remarks
    - Tip #1: be cautious when adding more GC layers!
    - Tip #2: increase the capability ***within*** each GC layer or add pre/post processing layers
    - Tip #3: add virtual edges/nodes
      - *So that a few GC layers can be sufficiently powerful*
      - Edge augmentation: $A \to A + A^2$
        - Example: bi-partite graph → add author-to-author interactions



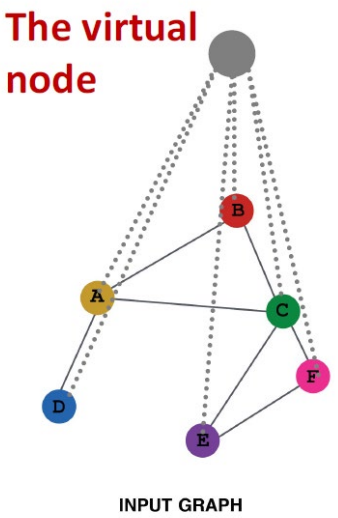
Representation Power

- GCN: stacking multiple graph convolution layers
  - $h^{(l)} = GC^{(l)}\left(h^{(l-1)}; \theta^{(l)}\right)\ 1 \leq l \leq L$
  - GCN will suffer from the ***over-smoothing*** issue when L is too large
  - Practical remarks
    - Tip #1: be cautious when adding more GC layers!
    - Tip #2: increase the capability ***within*** each GC layer or add pre/post processing layers
    - Tip #3: add virtual edges/nodes
      - *So that a few GC layers can be sufficiently powerful*
      - Edge augmentation: $A \to A + A^2$
      - Node augmentation
        - Add a fully connected virtual node
        - This improves message passing for sparse graphs



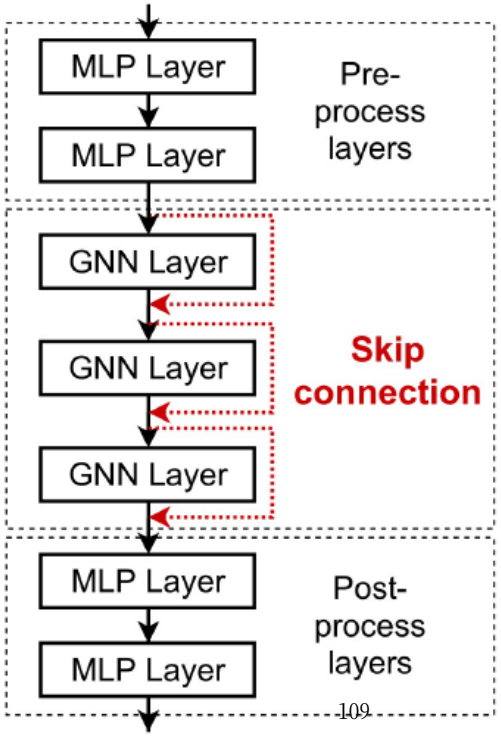
Representation Power

- GCN: stacking multiple graph convolution layers
  - $h^{(l)} = GC^{(l)}\left(h^{(l-1)}; \theta^{(l)}\right)\ 1 \leq l \leq L$
  - GCN will suffer from the ***over-smoothing*** issue when L is too large
  - Practical remarks
    - Tip #1: be cautious when adding more GC layers!
    - Tip #2: increase the capability ***within*** each GC layer or add pre/post processing layers
    - Tip #3: add virtual edges/nodes
      - *So that a few GC layers can be sufficiently powerful*
      - Edge augmentation: $A \to A + A^2$
      - Node augmentation
        - Add a fully connected virtual node
        - This improves message passing for sparse graphs
      - *What if we really want a lot of GC layers?*



Representation Power

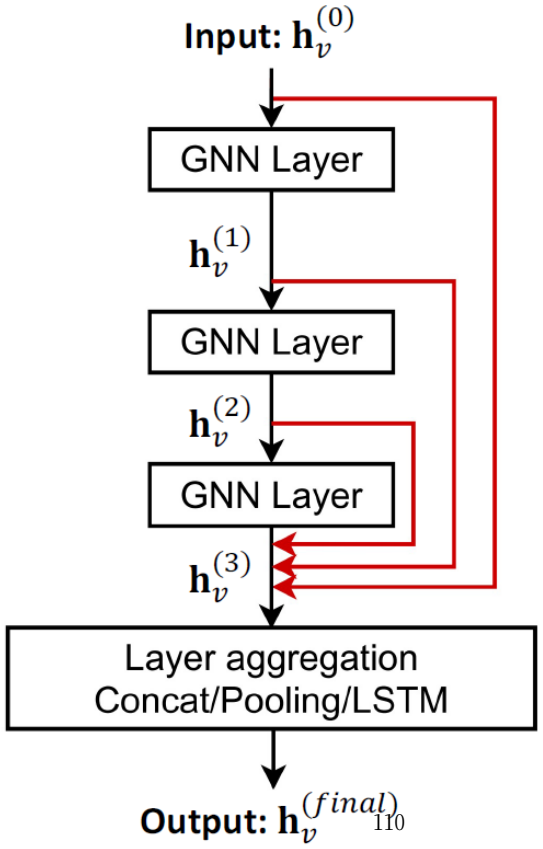
- GCN: stacking multiple graph convolution layers
  - $h^{(l)} = GC^{(l)}\left(h^{(l-1)}; \theta^{(l)}\right)\ 1 \leq l \leq L$
  - GCN will suffer from the ***over-smoothing*** issue when L is too large
  - Practical remarks
    - Tip #1: be cautious when adding more GC layers!
    - Tip #2: increase the capability ***within*** each GC layer or add pre/post pro
    - Tip #3: add virtual edges/nodes
    - Tip #4: add skip connection to leverage features from shallow layers!
      - Option #1: residual connection $h^{(l)} = GC\left(h^{(l-1)}\right) + h^{(l-1)}$



Representation Power

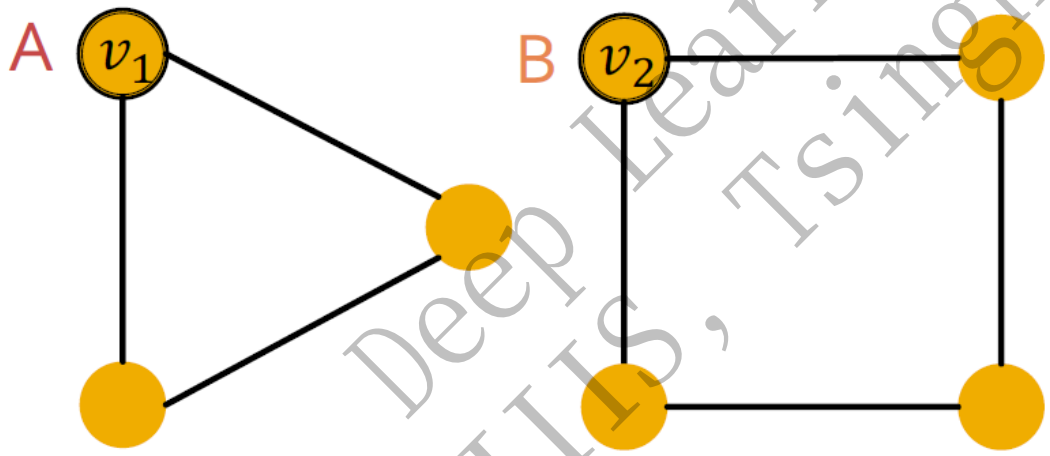
- GCN: stacking multiple graph convolution layers
  - $h^{(l)} = GC^{(l)}\left(h^{(l-1)}; \theta^{(l)}\right)\ 1 \le l \le L$
  - GCN will suffer from the ***over-smoothing*** issue when L is too l
  - Practical remarks
    - Tip #1: be cautious when adding more GC layers!
    - Tip #2: increase the capability ***within*** each GC layer or add pre/post pr
    - Tip #3: add virtual edges/nodes
    - Tip #4: add skip connection to leverage features from shallow layers!
      - Option #1: residual connection $h^{(l)} = GC\left(h^{(l-1)}\right) + h^{(l-1)}$
      - Option #2: direct connection to output

The end of story?



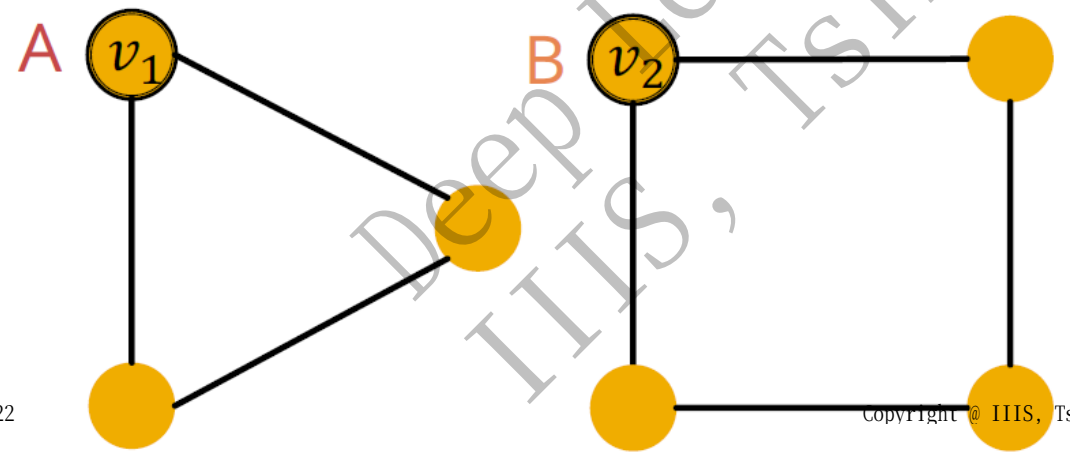
Representation Power

- GCN: stacking multiple graph convolution layers
  - $h^{(l)} = GC^{(l)}\left(h^{(l-1)}; \theta^{(l)}\right)\ 1 \leq l \leq L$
- *Can GCN distinguish which graph the selected node locates on?*

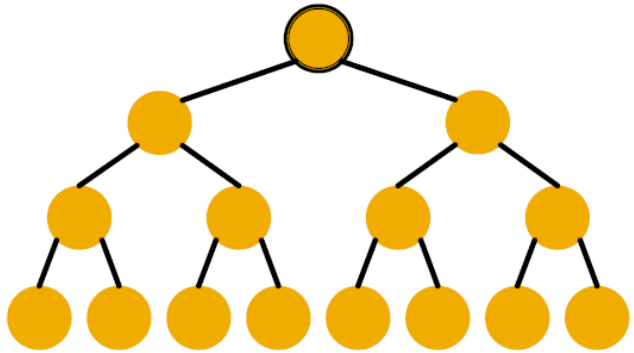


Representation Power

- GCN: stacking multiple graph convolution layers
  - $h^{(l)} = GC^{(l)}\left(h^{(l-1)}; \theta^{(l)}\right)\ 1 \leq l \leq L$
- *Can GCN distinguish which graph the selected node locates on?*
  - **NO!** They have the same computation graph!
  - *Can we fix it?*

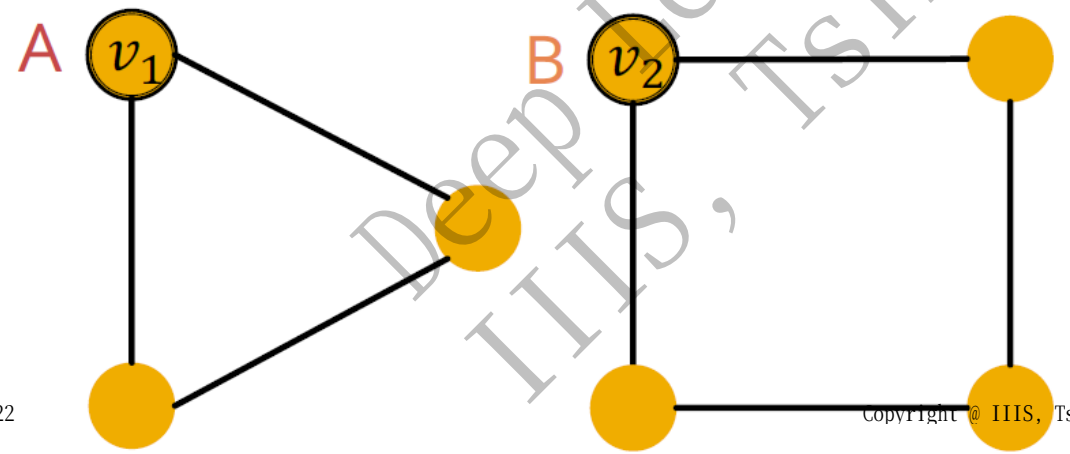


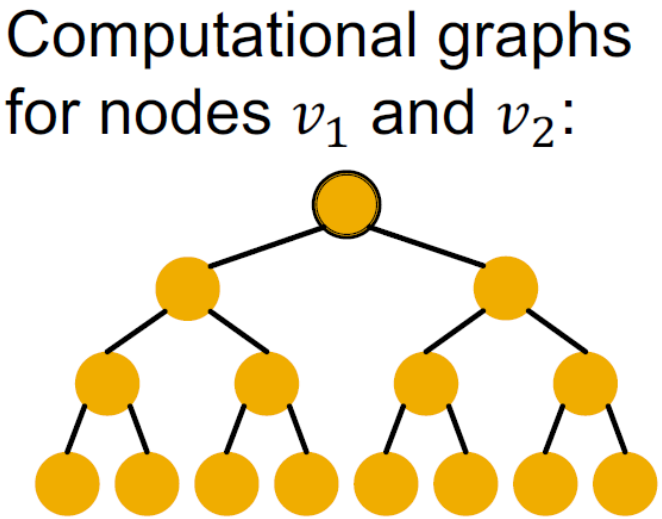
Computational graphs for nodes v_1 and v_2:



Representation Power

- GCN: stacking multiple graph convolution layers
  - $h^{(l)} = GC^{(l)}\left(h^{(l-1)}; \theta^{(l)}\right)\ 1 \leq l \leq L$
- *Can GCN distinguish which graph the selected node locates on?*
  - **NO!** They have the same computation graph!
  - Your homework ☺
    - Reference: position-aware GNN (https://arxiv.org/abs/1906.04817); identity-aware GNN (https://arxiv.org/abs/2101.10320)





Representation Power

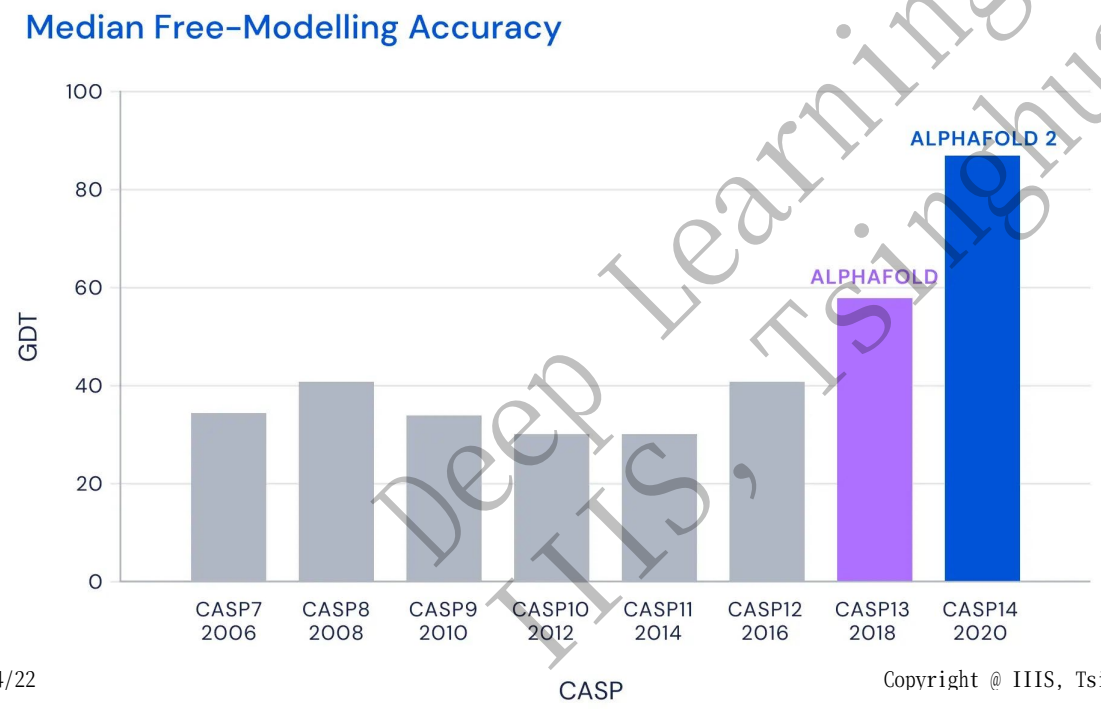
- Graph Convolution Layer
  - More powerful Aggregation function → stronger representation capability
- GCN: stacking multiple graph convolution layers
  - $h^{(l)} = GC^{(l)}\left(h^{(l-1)}; \theta^{(l)}\right)\ 1 \leq l \leq L$
  - Feature learning for each nodes with practical remarks
- Can we generate graphs?
  - Learning a generative model over graphs
  - Key insight: convert a graph into an image/matrix or a sequence
  - Your homework ☺
    - Reference
      - GraphVAE: https://arxiv.org/abs/1611.07308
      - GraphRNN: https://arxiv.org/abs/1802.08773
      - GraphAttentionRNN: https://arxiv.org/abs/1910.00760

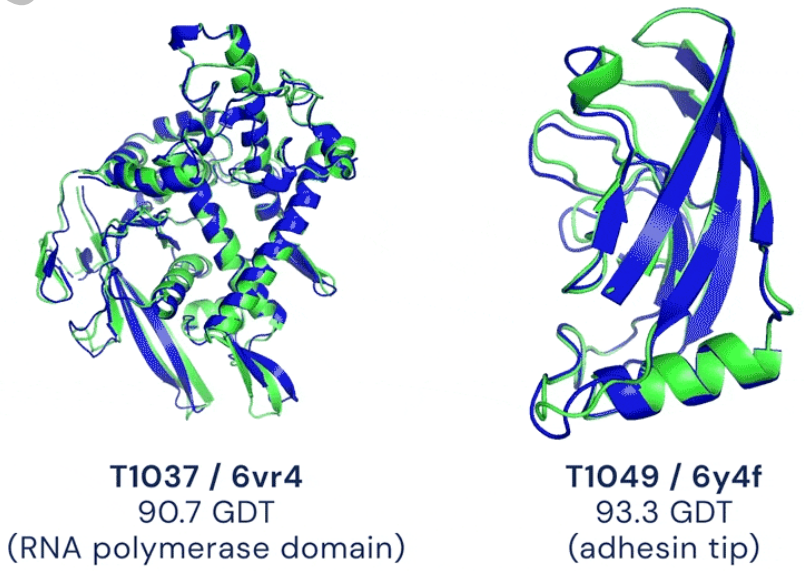
Representation Power

- Graph Convolution Layer
  - More powerful Aggregation function → stronger representation capability
- GCN: stacking multiple graph convolution layers
  - $h^{(l)} = GC^{(l)}\left(h^{(l-1)}; \theta^{(l)}\right)\ 1 \leq l \leq L$
  - Feature learning for each nodes with practical remarks
- Can we generate graphs?
  - Learning a generative model over graphs
  - Key insight: convert a graph into an image/matrix or a sequence
- *Can we generate something more complex?*

Protein Structure Prediction

- AlphaFold (DeepMind; v1.0 2020; v2.0 2021)
  - Amino Acid Sequence → Atomic Coordinates
  - End-to-end training; Transformer-based model; (https://github.com/deepmind/alphafold)



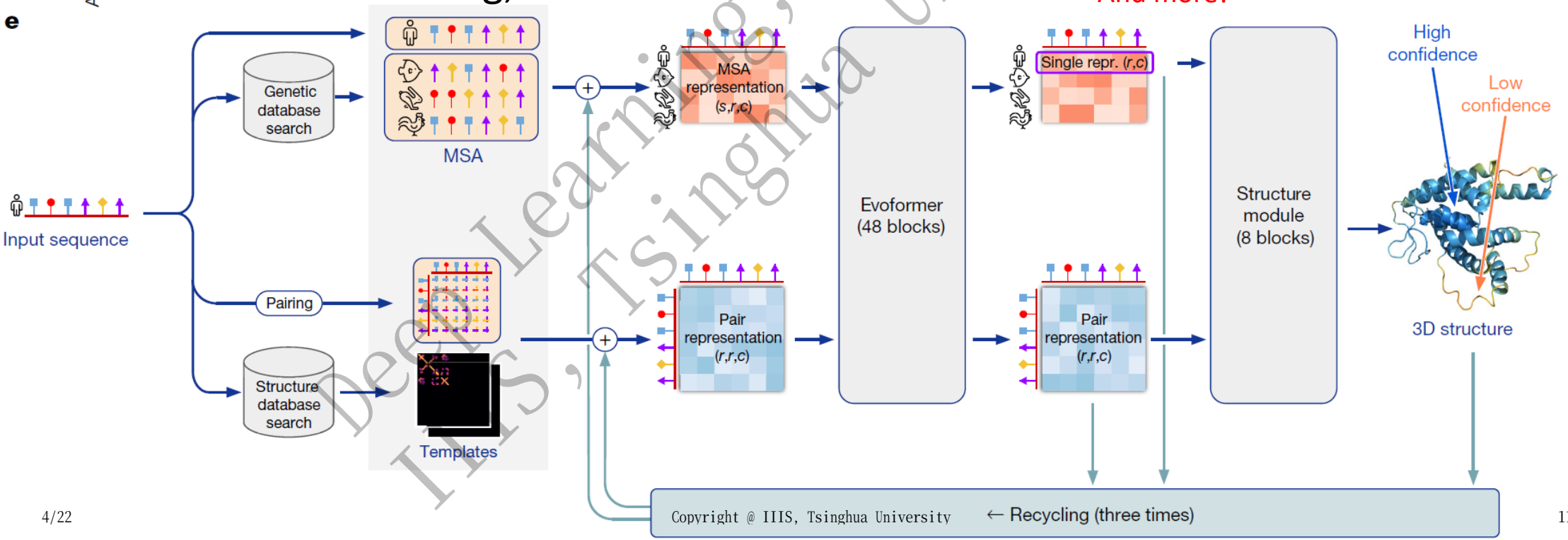


Protein Structure Prediction

- AlphaFold (DeepMind; v1.0 2020; v2.0 2021)
  - Amino Acid Sequence → Atomic Coordinates
  - End-to-end training; Transformer-based model

A lot of tricks and engineering!

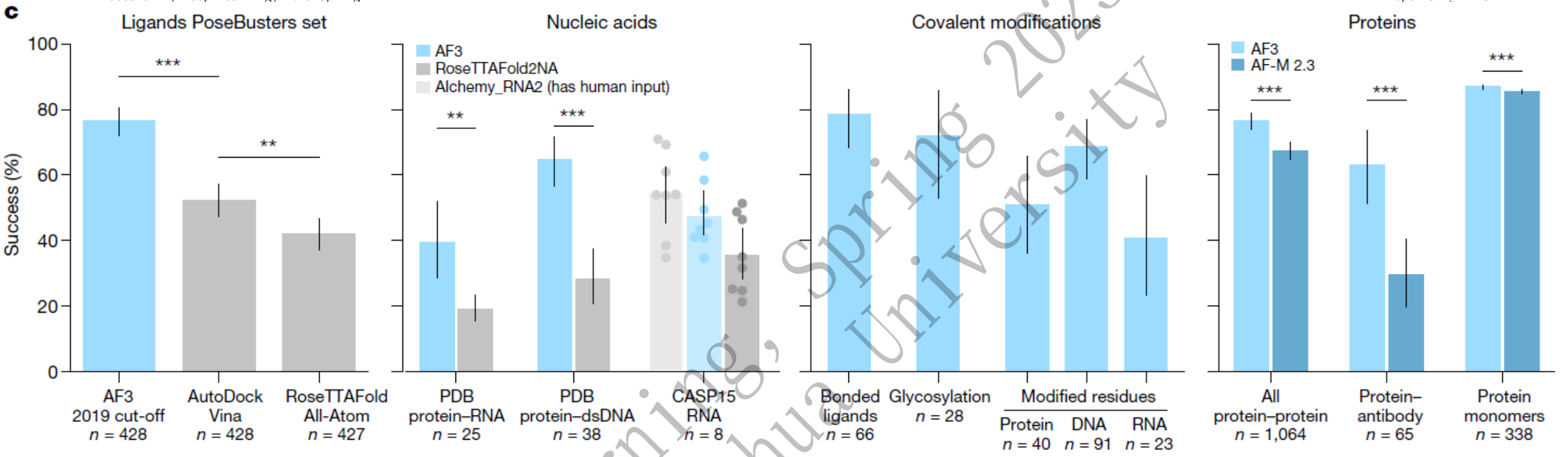
- Pre-processed MSA and templates
- Various new attention module
- Geometry-aware structure output
- Lots of loss functions
- Iterative refinement and self-distillation
- And more!

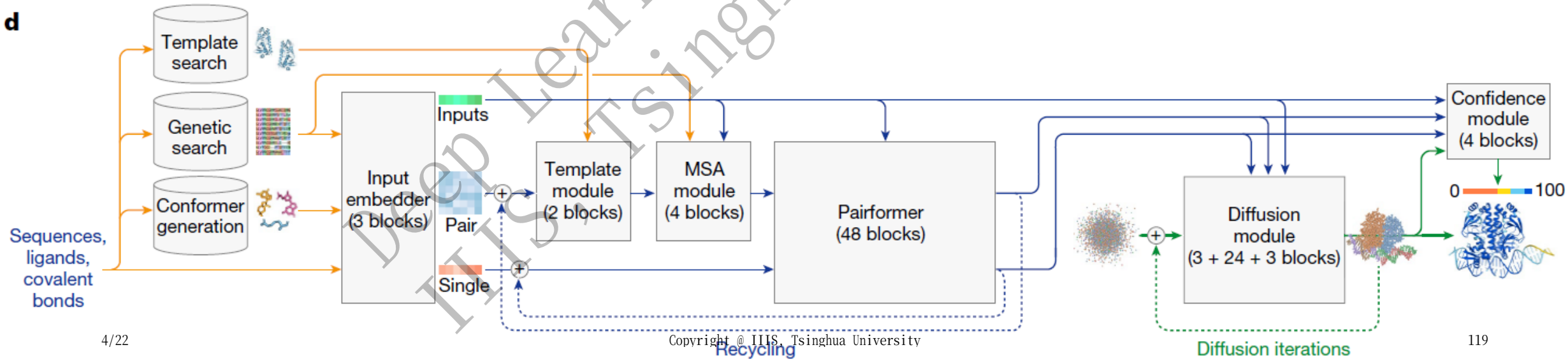


A good reading: https://www.blopig.com/blog/2021/07/alphafold-2-is-here-whats-behind-the-structure-prediction-miracle/

Protein Structure Prediction

- AlphaFold (DeepMind; v1.0 2020; v2.0 2021)
  - Amino Acid Sequence → Atomic Coordinates
  - End-to-end training; Transformer-based model
- AlphaFold-3 (Deepmind/Isomorphic Labs., 2024.5)
  - A much larger dataset (>200M protein structures)
  - A Diffusion-based architecture for end-to-end generation
  - Open-sourced server and dataset
    - https://deepmind.google/technologies/alphafold/





Protein Structure Prediction

THE NOBEL PRIZE

“for computational protein design”

© Nobel Prize Outreach. Photo: Clément Morin

“for protein structure prediction”

© Nobel Prize Outreach. Photo: Clément Morin

“for protein structure prediction”

© Nobel Prize Outreach. Photo: Clément Morin

Graph Convolution Network

- GCN is a tool for learning graph representations for graphs
  - Key idea: local message aggregation for each node
  - Applications
    - node labeling
    - edge prediction:
    - graph classification
- On the expressiveness power of GCN
  - Capability of aggregation operator
  - Practical representation enhancements
    - shallow layers; model/graph/feature augmentation
- Generative modeling

Summary

- Structured Priors in Deep Generative Model
  - Gumbel-Softmax: Reparameterization for Softmax
  - VQ-VAE & auto-regressive prior learning

- Deep Learning for Structured Data
  - GCN variants
  - Representation power and practical techniques

- Applications
  - (Conditioned) Structured Data Generation
  - Interaction/Graph Prediction (RecSys, Drug Discovery)

Thanks!

- Good luck with your final project proposal & happy holiday!