Deep learning for probabilistic models

Why Toward tractable inference for more expressive probabilistic models

- Tractable inference for intractable distributions (unnormalized density)

Posterior distribution	$p(\theta \mid \mathbf{X}, \alpha) = \frac{p(\mathbf{X} \mid \theta) p(\theta \mid \alpha)}{p(\mathbf{X} \mid \alpha)} \propto p(\mathbf{X} \mid \theta) p(\theta \mid \alpha)$
Energy-based models	$P(x) = \frac{1}{Z} \exp f(x)$

- Guide the design of deep learning models
- Complex generative tasks

Similar to deep learning, inference method are often gradient based

- Variational inference
- MCMC (e.g. Hamiltonian Monte Carlo uses gradient to speed up sampling)

http://arogozhnikov.github.io/2016/12/19/markov_chain_monte_carlo.html

Deep learning for probabilistic models

Why Toward tractable inference for more expressive probabilistic models

- Tractable inference for intractable distributions (unnormalized density)

Posterior distribution	$p(\theta \mid \mathbf{X}, \alpha) = \dfrac{p(\mathbf{X} \mid \theta) p(\theta \mid \alpha)}{p(\mathbf{X} \mid \alpha)} \propto p(\mathbf{X} \mid \theta) p(\theta \mid \alpha)$
Energy-based models	$P(x) = \dfrac{1}{Z} \exp f(x)$

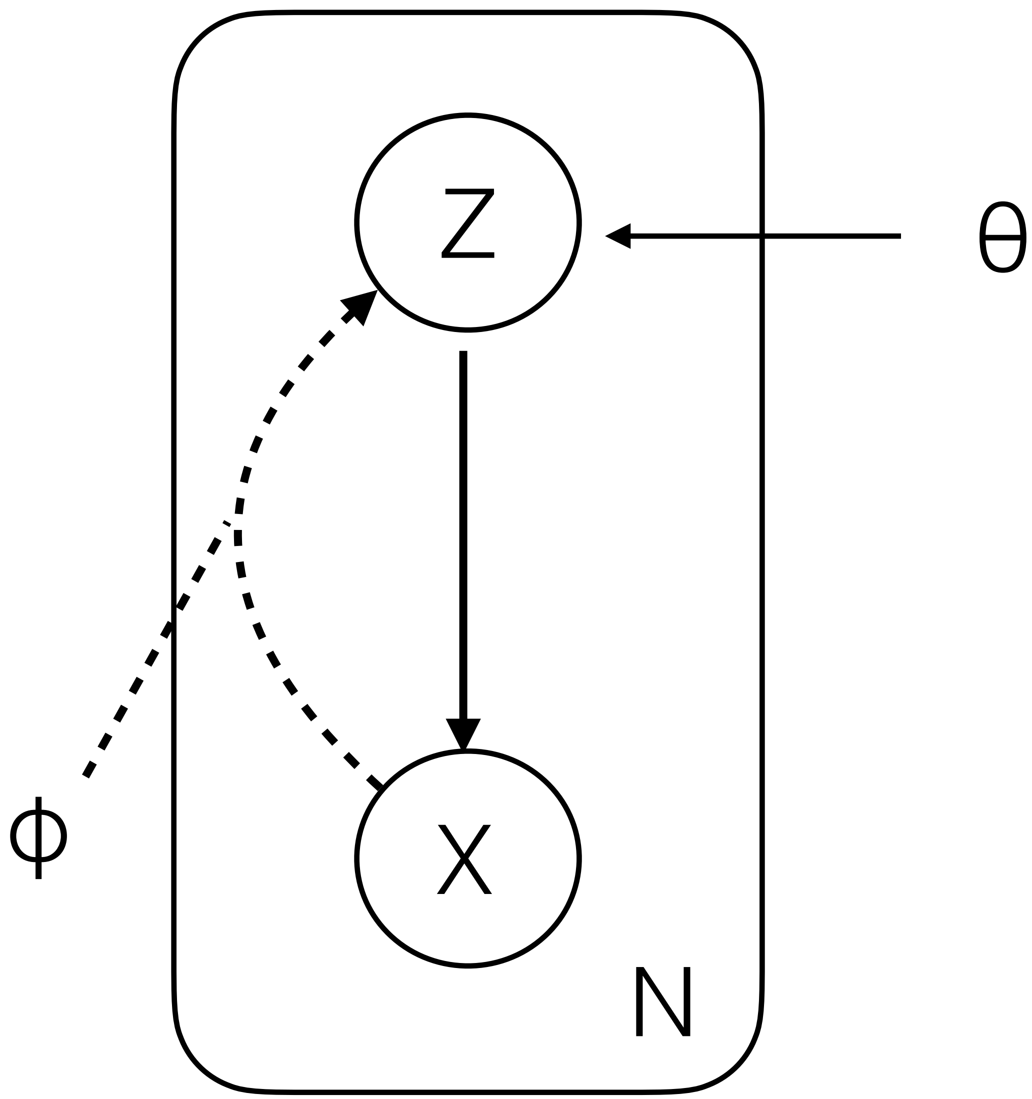
Potential approaches for NN-assisted inference

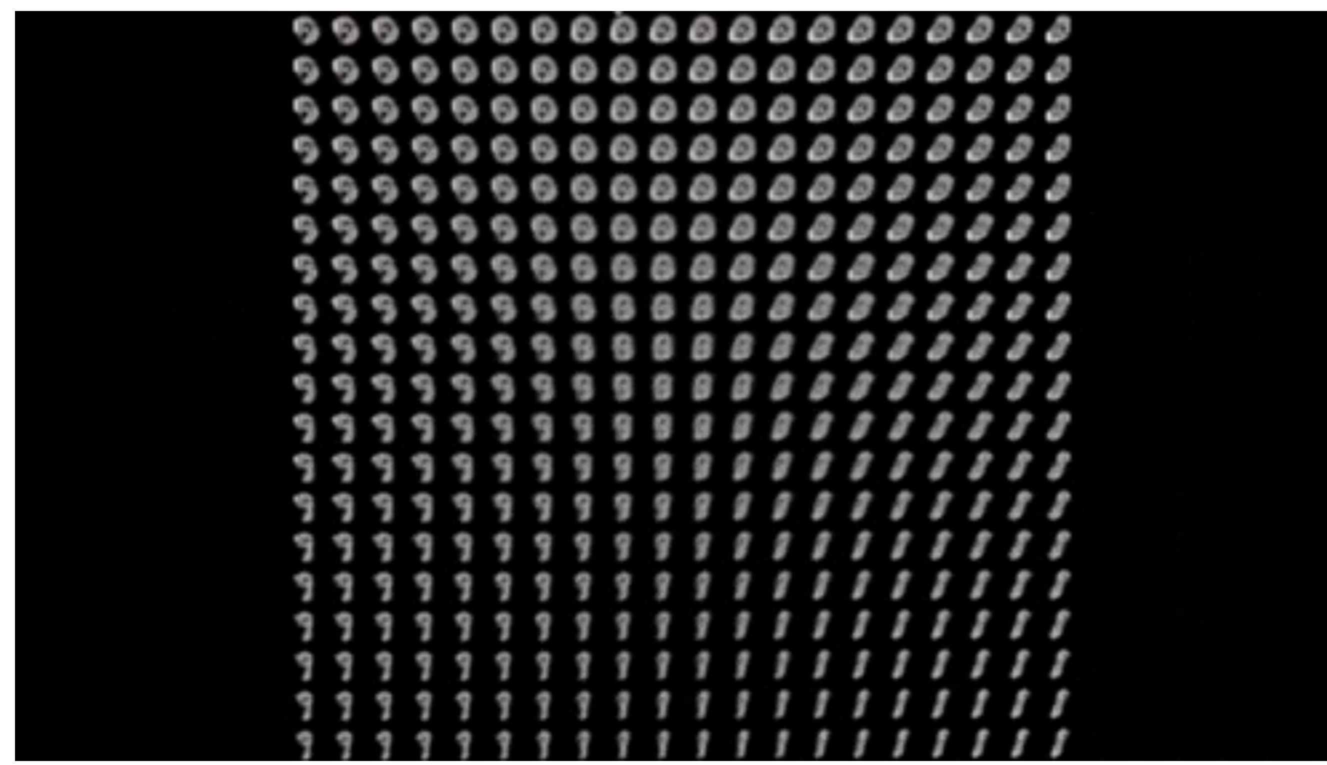
  - Neural variational inference (variational autoencoder, diffusion probability model*)
  - Neural MCMC sampler

- Design probability model with tractable & flexible distribution
  - Neural autoregressive model (e.g. transformer language model)
  - Normalizing flow
  - Neural ODE (continuous normalizing flow)
- Implicit probability model with sampling capability
  - Generative adversarial network*
  - Diffusion probability models*

Neural variational inference

Use neural network for describing P(X|Z) or Q(Z|X)

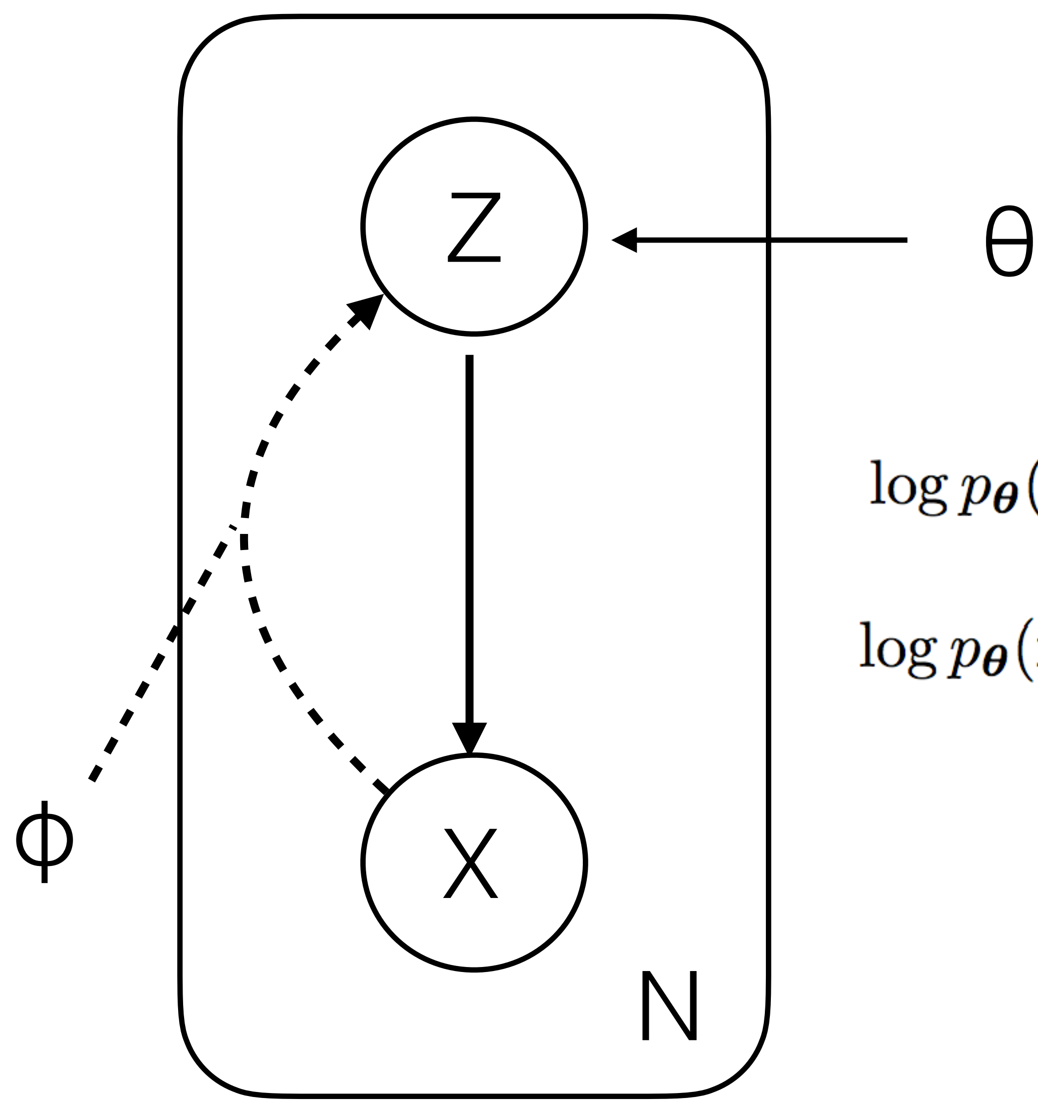




Kingma and Welling, 2014 Auto-Encoding Variational Bayes

Neural variational inference

Use neural network for describing P(X|Z) or Q(Z|X)



$$\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) = D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^{(i)})||p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}^{(i)})) + \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}^{(i)})$$

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) \geq \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}^{(i)}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[-\log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) + \log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})\right]$$

$$= \boxed{-D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^{(i)})||p_{\boldsymbol{\theta}}(\mathbf{z})) + \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^{(i)})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}|\mathbf{z})\right]}$$

The variational objective

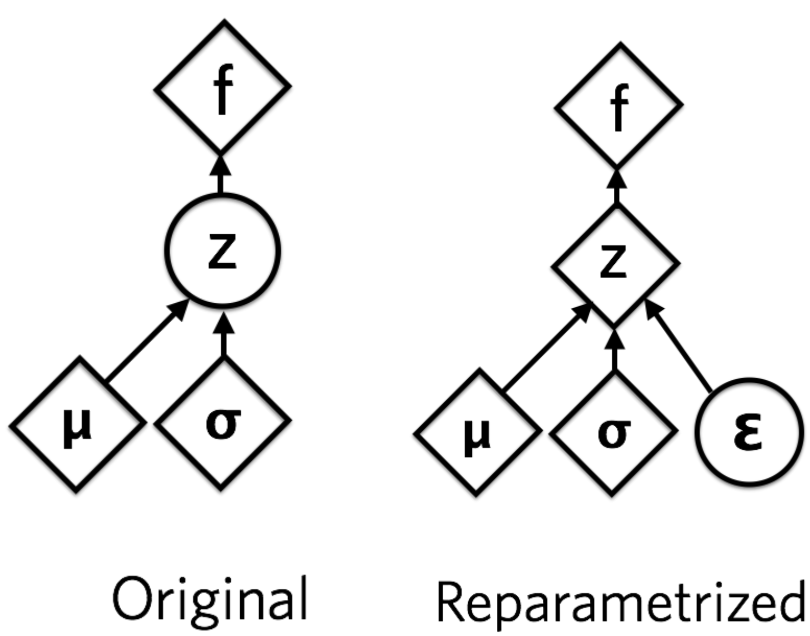
Kingma and Welling, 2014 Auto-Encoding Variational Bayes

Backpropagation over stochastic units:
Reparametrization trick

How to compute good gradient estimate of

$$-D_{KL}(q_{\phi}(\mathbf{z}|\mathbf{x}^{(i)})||p_{\theta}(\mathbf{z})) + \boxed{\mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x}^{(i)})}\left[\log p_{\theta}(\mathbf{x}^{(i)}|\mathbf{z})\right]}$$

Gradient of expectation -> expectation of stochastic gradient



$$\nabla_{\mu,\sigma} E_{z\sim\mu,\sigma}[f(z)] = E_{z\sim\mu,\sigma}[f(z)\nabla_{\mu,\sigma}\log\left(p(z|\mu,\sigma)\right)]$$

$$\nabla_{\mu,\sigma} E_{\epsilon\sim p(\epsilon)}[f(z)] = E_{\epsilon\sim p(\epsilon)}[\nabla_{\mu,\sigma} f\left(g(\mu,\sigma,\epsilon)\right)]$$

Williams, 1992 “REINFORCE” estimator

Kingma , et al. 2014. Auto-Encoding Variational Bayes

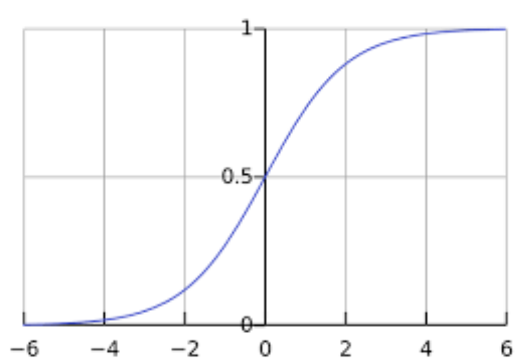
Backpropagation over stochastic units:
Reparametrization trick for discrete variables

The Gumbel trick for sampling from discrete distributions $P(X = k) \propto \alpha_k$

$$G = -\log(-\log(U)) \text{ with } U \sim \mathrm{Unif}[0, 1]$$

$$X = \arg\max_k \left(\log \alpha_k + G_k\right).$$

Softmax function for approximating the max operation with a differentiable function



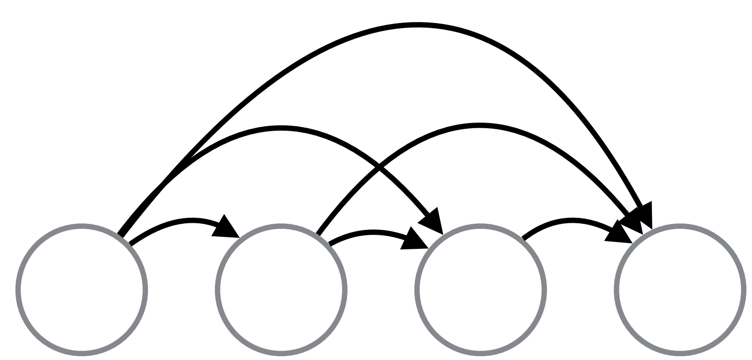
$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

Discrete variables can always be represented by binary vectors

Toward flexible and normalized density models

$$-D_{KL}(q_{\phi}(\mathbf{z}|\mathbf{x}^{(i)})||p_{\theta}(\mathbf{z})) + \mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x}^{(i)})}\left[\log p_{\theta}(\mathbf{x}^{(i)}|\mathbf{z})\right]$$

1.Fully factorized models

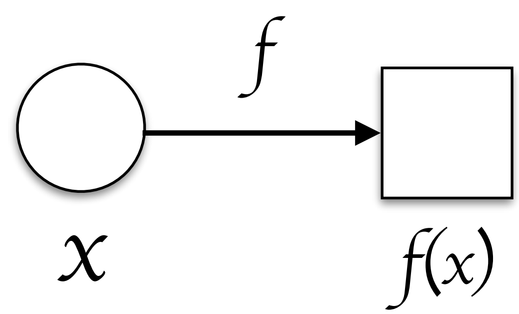


Neural autoregressive models

Probability function is fully factorized
However, it has to commit to a certain order

Toward flexible and normalized density models

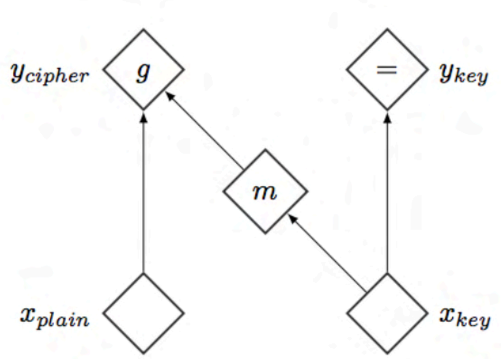
2. Invertible transformations (Flow models)



$$p_X(x) = p_H(f(x))|\det \frac{\partial f(x)}{\partial x}|.$$

Examples:

NICE



determinant fixed

Dinh 2015, NICE: NON-LINEAR INDEPENDENT COMPONENTS ESTIMATION

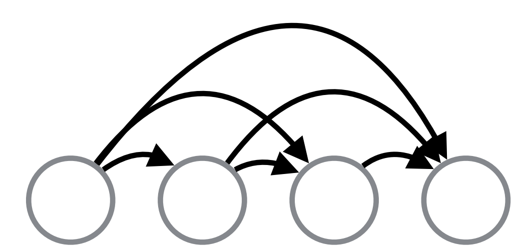
Normalizing flow

$$f(\mathbf{z}) = \mathbf{z} + \mathbf{u}h(\mathbf{w}^\top \mathbf{z} + b)$$

determinant O(D) time

Rezende 2016. Variational Inference with Normalizing Flows

Invertible autoregressive flow



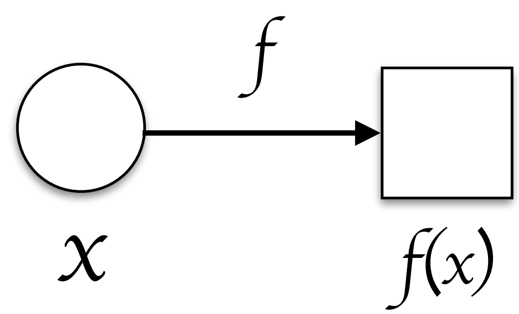
determinant O(D) time

Kingma, 2017 Invertible autoregressive flow

Hidden variables are equal in dimensionality.

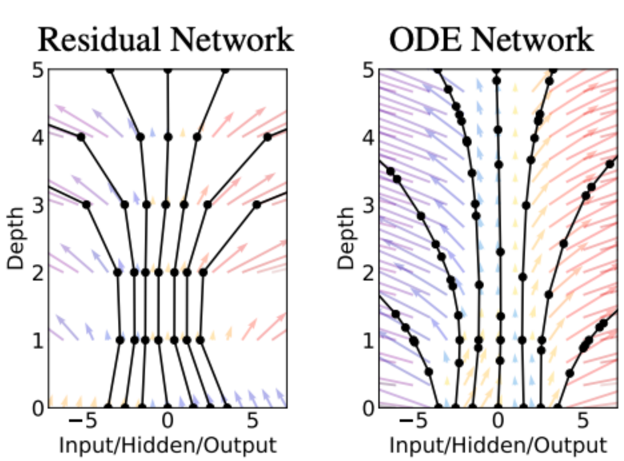
Toward flexible and normalized density models

2. Invertible transformations (Flow models)



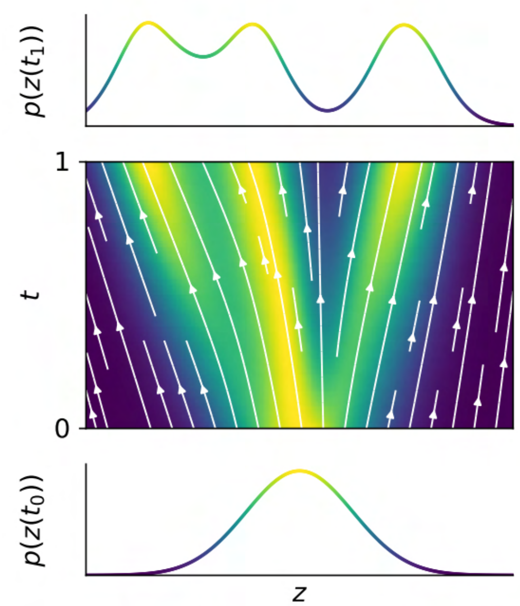
$$p_X(x) = p_H(f(x))|\det \frac{\partial f(x)}{\partial x}|.$$

More Examples:



Chen 2018, Neural ODE

ODE



determinant fixed

Grathwohl 2019: FFJORD

Continuous change of variable formula

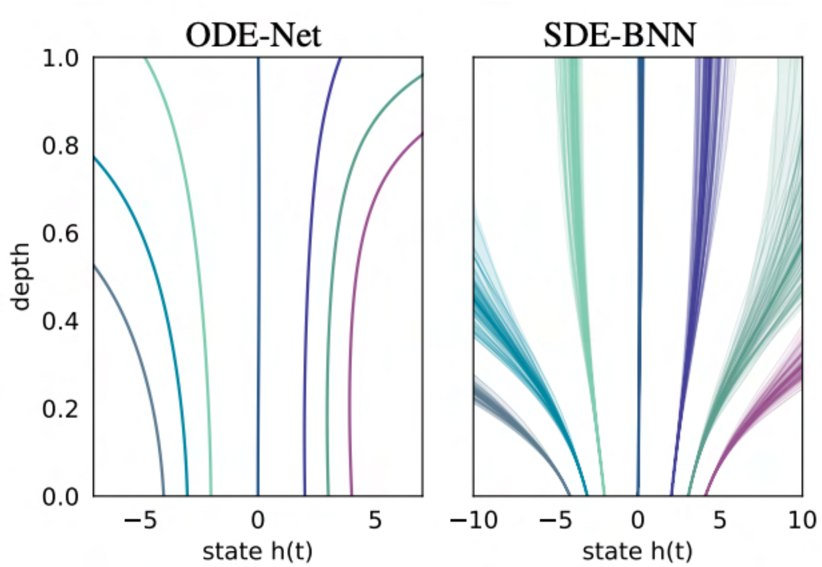
Log likelihood

$$\log p(\mathbf{z}(t_1)) = \log p(\mathbf{z}(t_0)) - \int_{t_0}^{t_1} \mathrm{Tr}\left(\frac{\partial f}{\partial \mathbf{z}(t)}\right) dt.$$

FFJORD: Unbiased estimate of $\mathrm{Tr}(\frac{\partial f}{\partial \mathbf{z}})$ with $\boldsymbol{\epsilon}^T \frac{\partial f}{\partial \mathbf{z}} \boldsymbol{\epsilon}$

Hidden variables are equal in dimensionality.

Probabilistic inference for trajectories using SDE-BNN



Xu 2021, Infinitely Deep Bayesian Neural Networks with Stochastic Differential Equations

Drift function

Diffusion function

$$\mathrm{d}w_t = f(w_t, t)\ \mathrm{d}t + g(w_t, t)\ \mathrm{d}B_t,$$

Infinite dimensional ELBO

$$\mathcal{L}_{\mathrm{ELBO}_\infty}(\phi) = \mathbb{E}_{q_\phi(w)}\left[\log p(\mathcal{D}|w) - \int_0^1 \tfrac{1}{2}\left\|u(w_t, t, \phi)\right\|_2^2\ \mathrm{d}t\right]$$

$$u(t, \phi) = g_\theta(w_t, t)^{-1}\left[f_\theta(w_t, t) - f_\phi(w_t, t)\right]$$

Prior drift function

Variational approximate posterior drift function

Score-matching for partition function-free generative model fitting

Energy-based models $P(x) = \frac{1}{Z} \exp f(x)$

score := gradient of log probability wrt x

$$\mathbf{s}_\theta(\mathbf{x}) = \nabla_\mathbf{x} \log p_\theta(\mathbf{x}) = -\nabla_\mathbf{x} f_\theta(\mathbf{x}) + \underbrace{\nabla_\mathbf{x} \log Z_\theta}_{=0} = -\nabla_\mathbf{x} f_\theta(\mathbf{x})$$

Minimize Fisher divergence $\mathbb{E}_{p(\mathbf{x})}[\|\nabla_\mathbf{x} \log p(\mathbf{x}) - \mathbf{s}_\theta(\mathbf{x})\|_2^2]$

Equivalent to $\mathbb{E}_{p_d}\left[\operatorname{tr}(\nabla_\mathbf{x} \mathbf{s}_m(\mathbf{x};\boldsymbol{\theta})) + \frac{1}{2}\|\mathbf{s}_m(\mathbf{x};\boldsymbol{\theta})\|_2^2\right]$ Hyvärinen (2005)

$$J(Q) = \int_{\mathbb{R}^m} p(x)\left[\|\nabla \log q(x) - \nabla \log p(x)\|_2^2\right] dx \quad <=> \quad J(Q) = \int_{\mathbb{R}^m} p(x)\left[\Delta \log q(x) + \frac{1}{2}\|\nabla \log q(x)\|_2^2\right] dx + \text{const},$$

However, trace of Hessian is usually intractable / too slow

Sliced score-matching (random projection)

$$\mathbb{E}_{p_\mathbf{v}}\mathbb{E}_{p_{\text{data}}}\left[\mathbf{v}^\intercal \nabla_\mathbf{x} \mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x})\mathbf{v} + \frac{1}{2}\|\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x})\|_2^2\right]$$

Denoised score-matching

$$\frac{1}{2}\mathbb{E}_{q_\sigma(\tilde{\mathbf{x}}|\mathbf{x})p_{\text{data}}(\mathbf{x})}[\|\mathbf{s}_{\boldsymbol{\theta}}(\tilde{\mathbf{x}}) - \nabla_{\tilde{\mathbf{x}}} \log q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x})\|_2^2].$$

http://yang-song.github.io/blog/2021/score/

Sample from p(X) using its gradient: Langevin dynamics

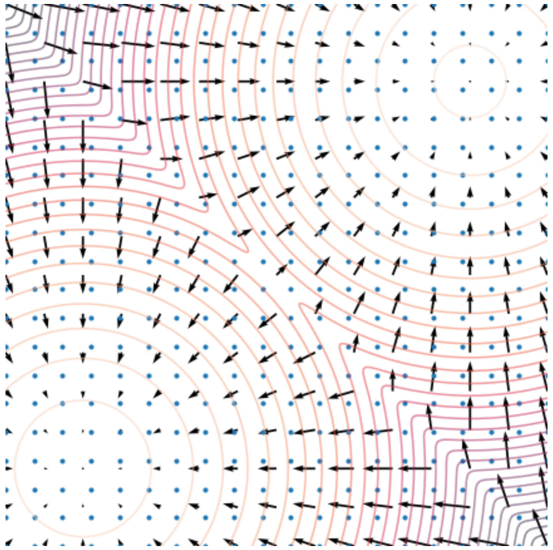
Initialize x from arbitrary distribution

$$\mathbf{x}_{i+1} \leftarrow \mathbf{x}_i + \epsilon \nabla_{\mathbf{x}} \log p(\mathbf{x}) + \sqrt{2\epsilon}\, \mathbf{z}_i, \quad i = 0, 1, \cdots, K,$$

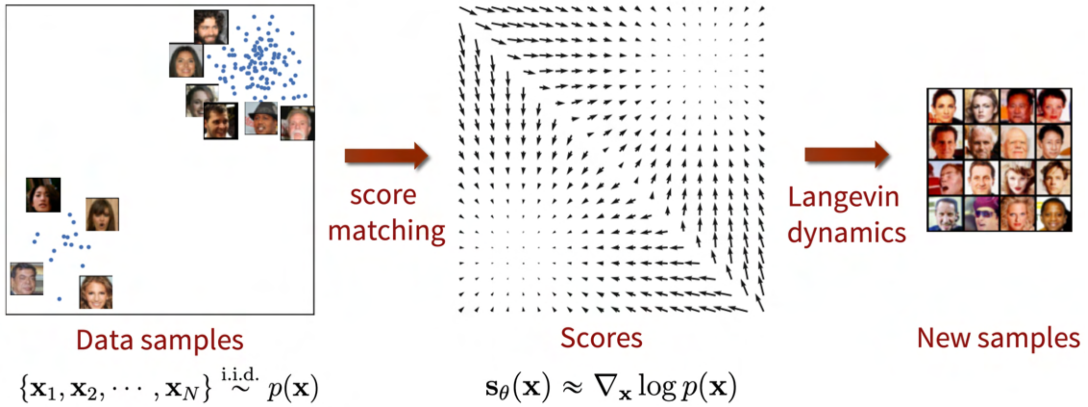
$\mathbf{z}_i \sim \mathcal{N}(0, I)$.

$\epsilon \to 0$

$K \to \infty$



i.e. once we learned the score function, we can sample from p(X),



Sample from p(X) using its gradient: Langevin dynamics

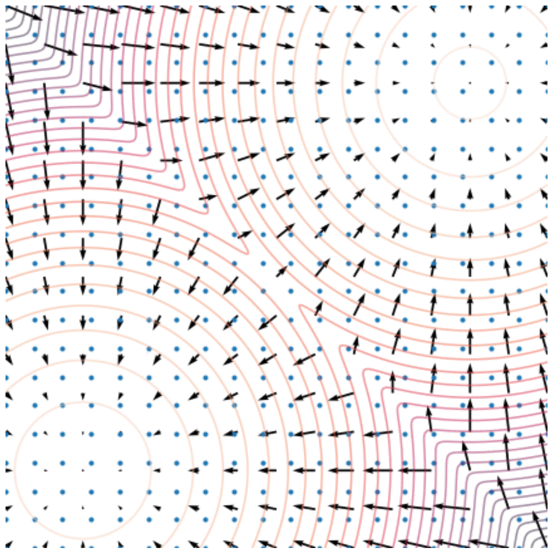
Initialize x from arbitrary distribution

$$\mathbf{x}_{i+1} \leftarrow \mathbf{x}_i + \epsilon \nabla_{\mathbf{x}} \log p(\mathbf{x}) + \sqrt{2\epsilon}\, \mathbf{z}_i, \quad i = 0, 1, \cdots, K,$$

$\mathbf{z}_i \sim \mathcal{N}(0, I)$.

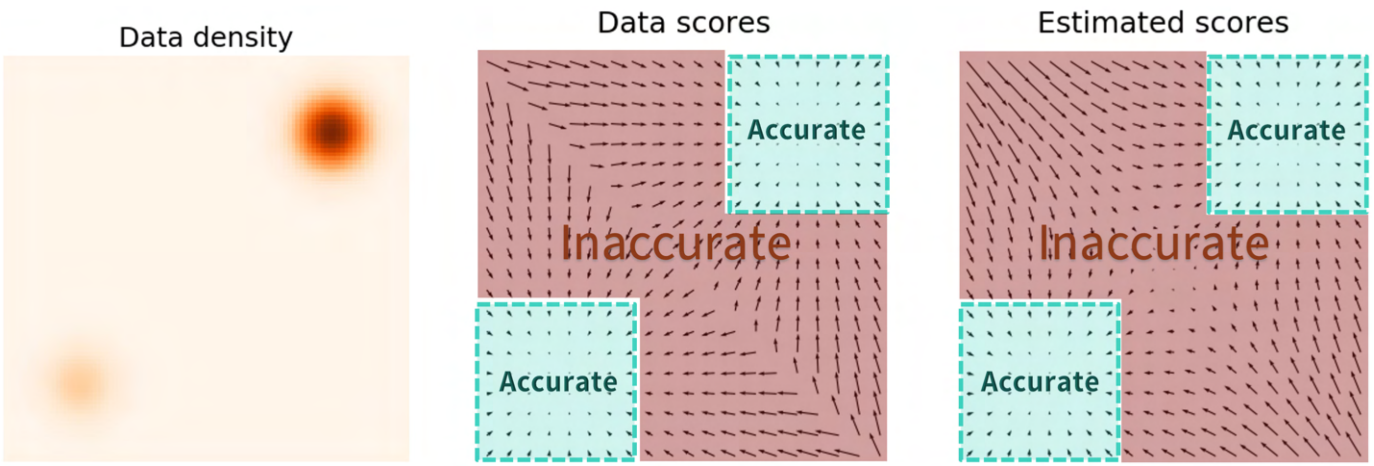
$\epsilon \to 0$

$K \to \infty$

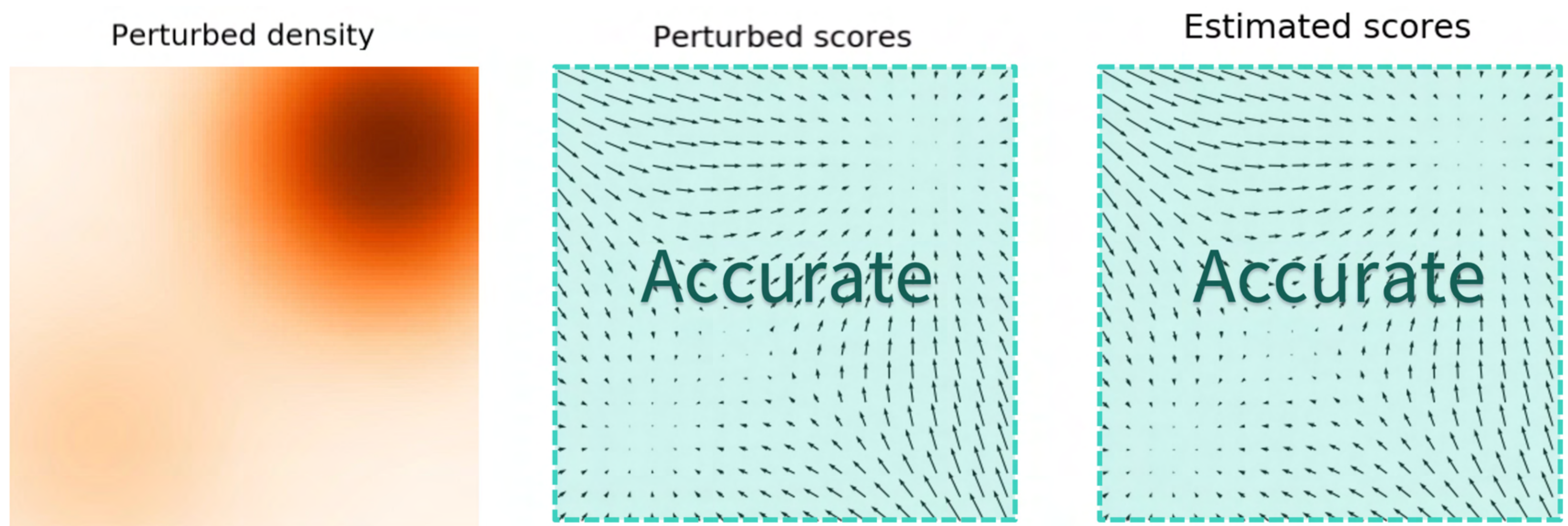


i.e. once we learned the score function, we can sample from p(X),

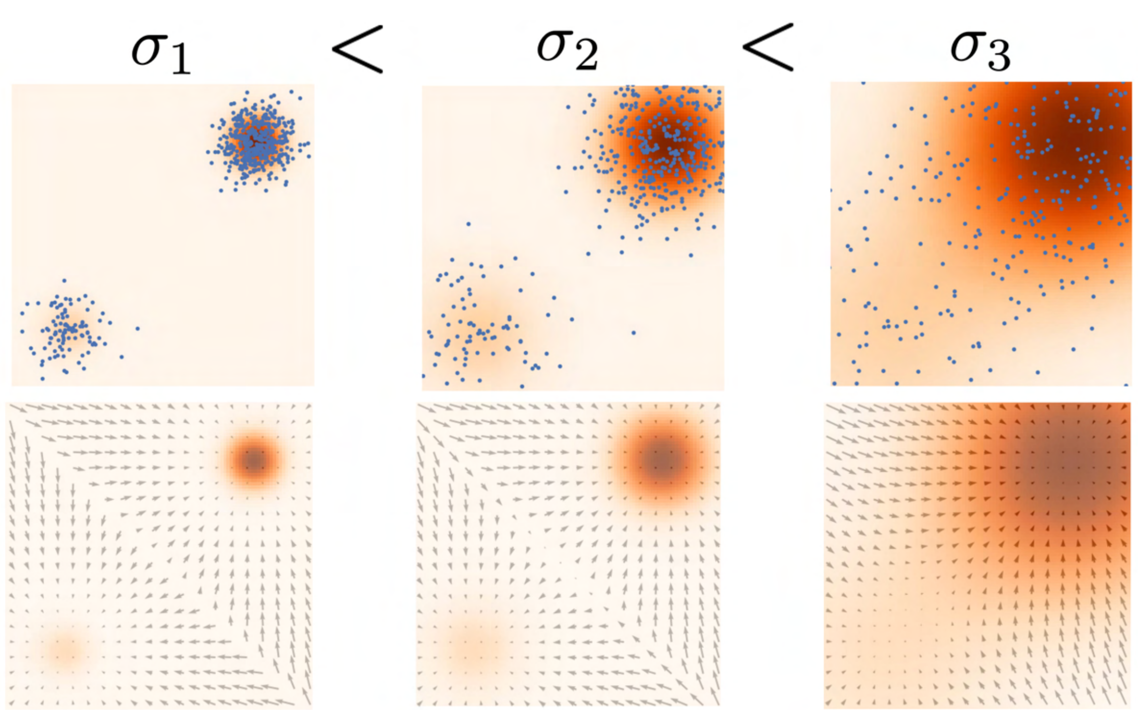
however…



Learning the score function with data + noise



What noise level? Use multiple!



Annealed Langevin dynamics

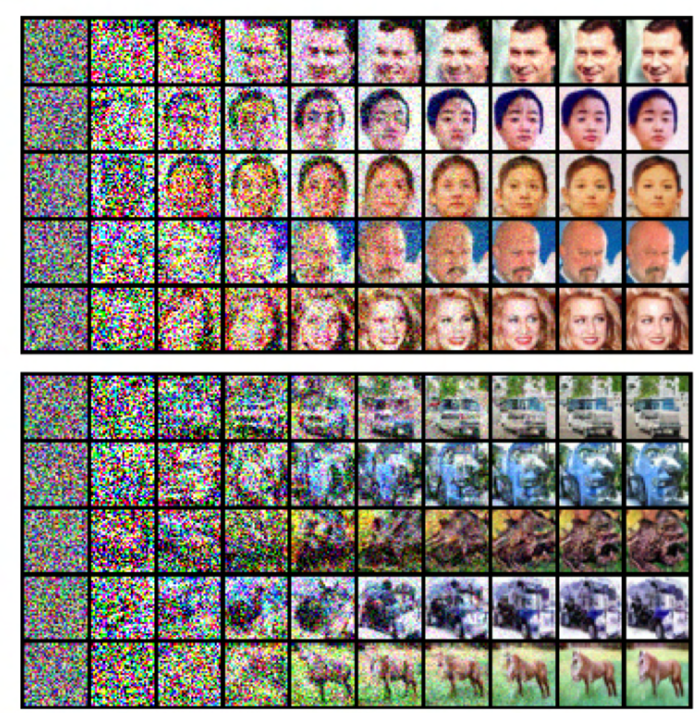
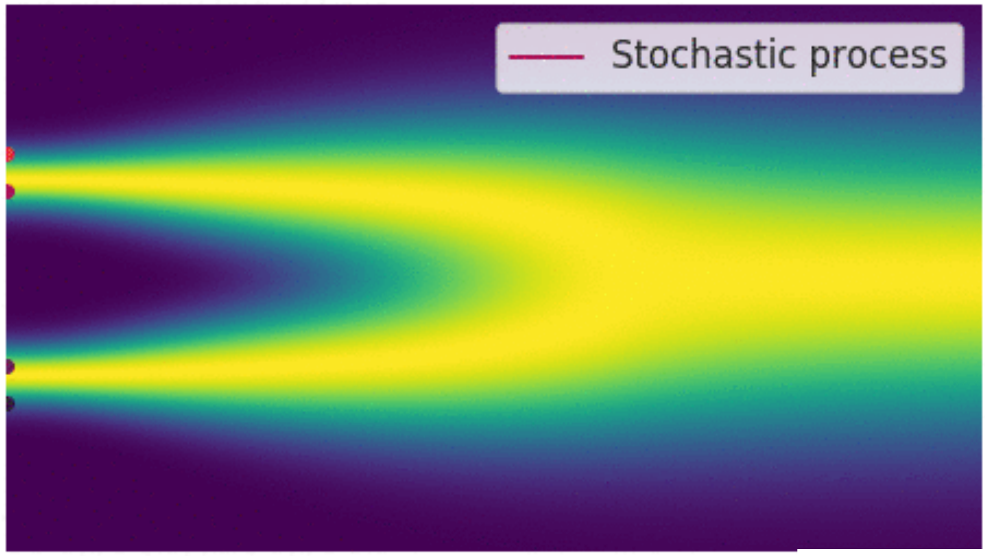


Figure 4: Intermediate samples of annealed Langevin dynamics.

Generative Modeling by Estimating Gradients of the Data Distribution

Score-based generative modeling with stochastic differential equations (SDEs)

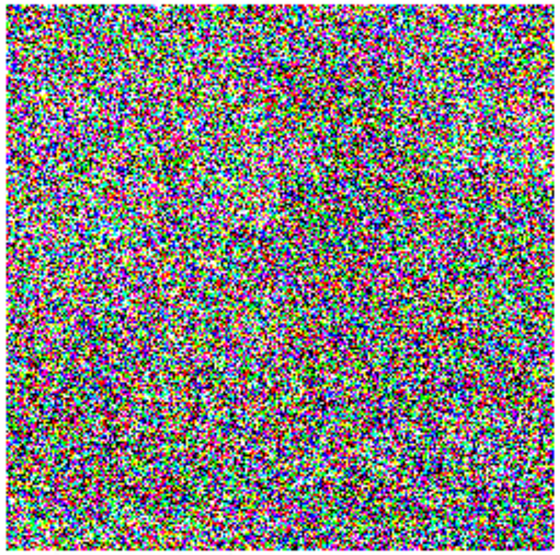
Multiple noise-levels -> infinite noise levels (SDE)

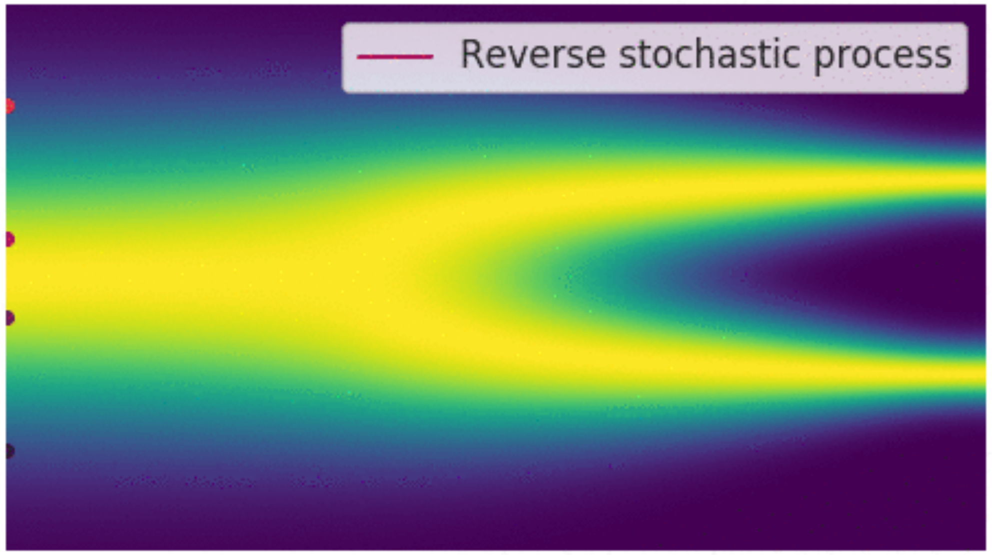


Converge to a static distribution (prior distribution)

$$dx_t = -\theta\, x_t\, dt + \sigma\, dW_t$$

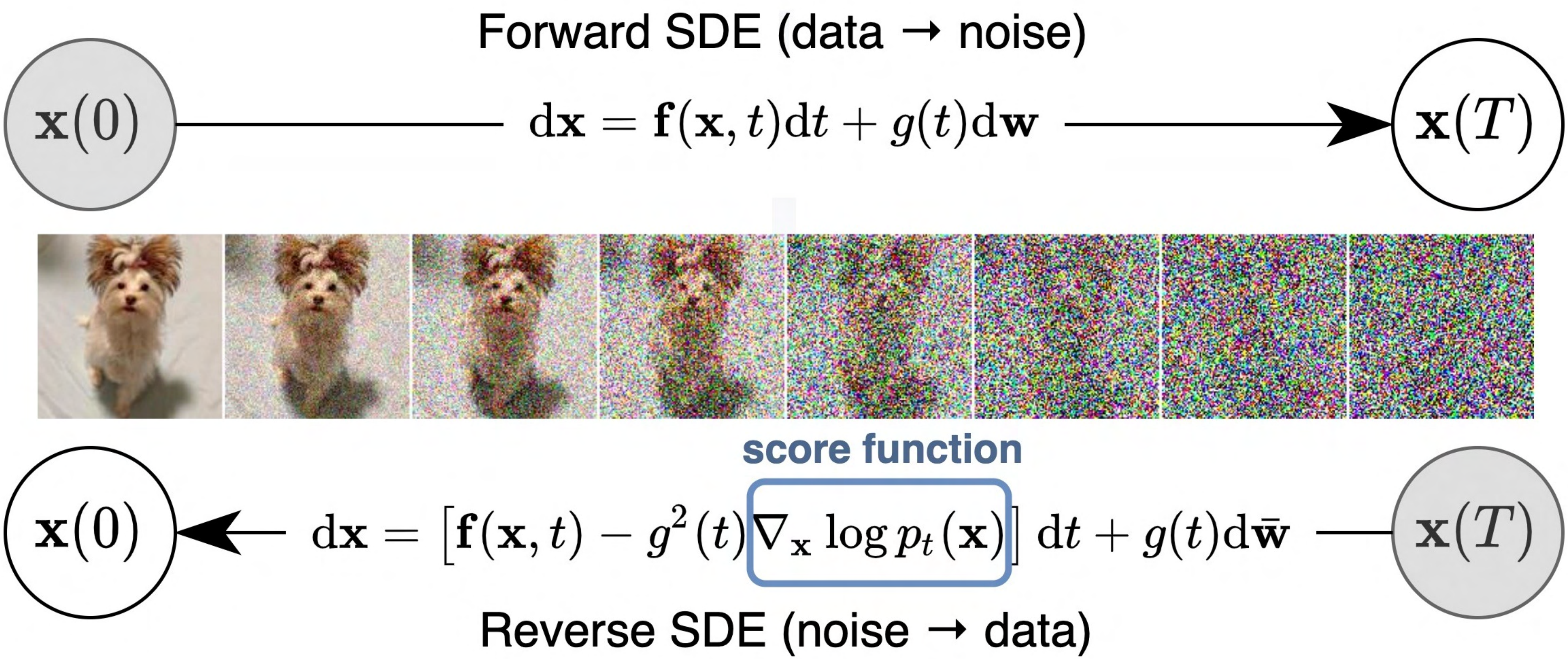
Reverse SDE is equivalent to sampling!





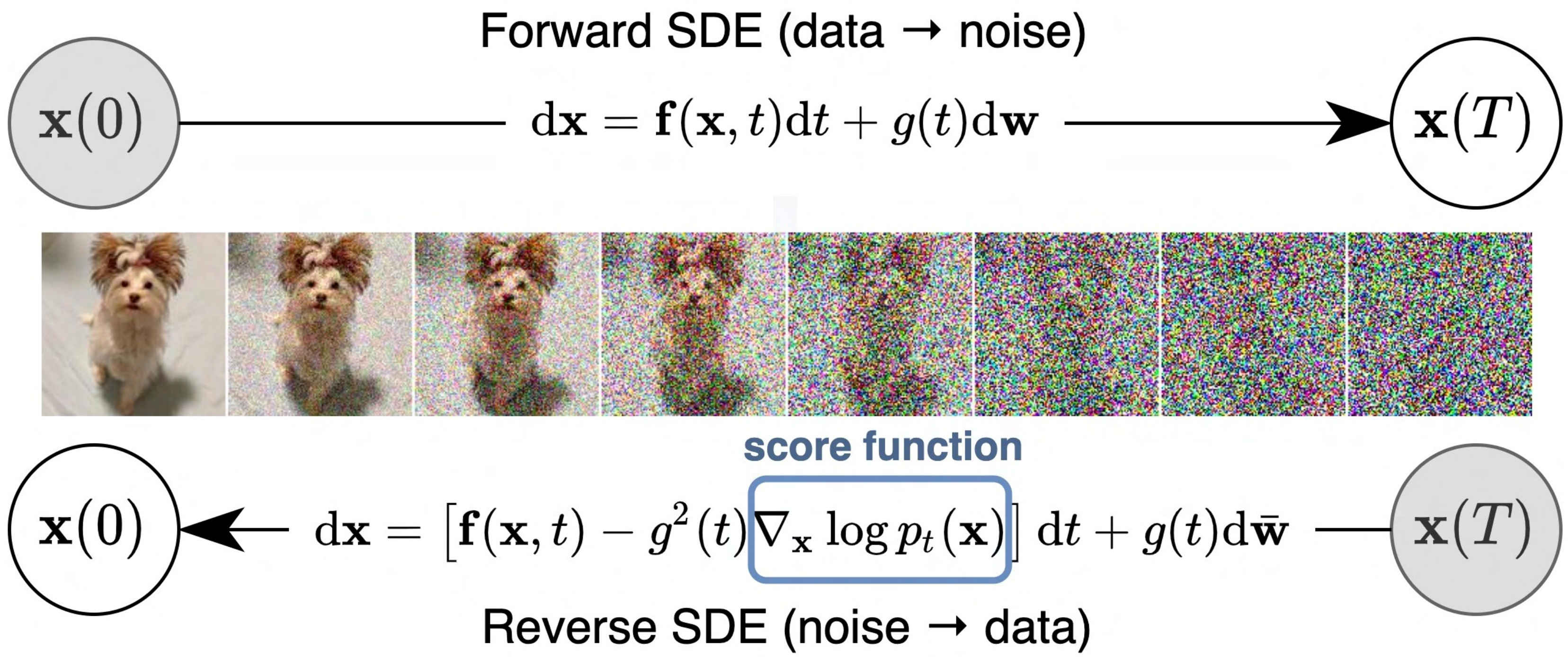
Score-Based Generative Modeling through Stochastic Differential Equations.

Score-based generative modeling with stochastic differential equations (SDEs)



Score-Based Generative Modeling through Stochastic Differential Equations.

Score-based generative modeling with stochastic differential equations (SDEs)



Learning the score function with infinite noise levels (SDE)

score-matching $\mathbb{E}_{p(\mathbf{x})}[\|\nabla_{\mathbf{x}} \log p(\mathbf{x}) - \mathbf{s}_\theta(\mathbf{x})\|_2^2]$

SDE score-matching $\mathbb{E}_{t \in \mathcal{U}(0,T)}\mathbb{E}_{p_t(\mathbf{x})}[\lambda(t)\|\nabla_{\mathbf{x}} \log p_t(\mathbf{x}) - \mathbf{s}_\theta(\mathbf{x}, t)\|_2^2]$

Score-Based Generative Modeling through Stochastic Differential Equations.

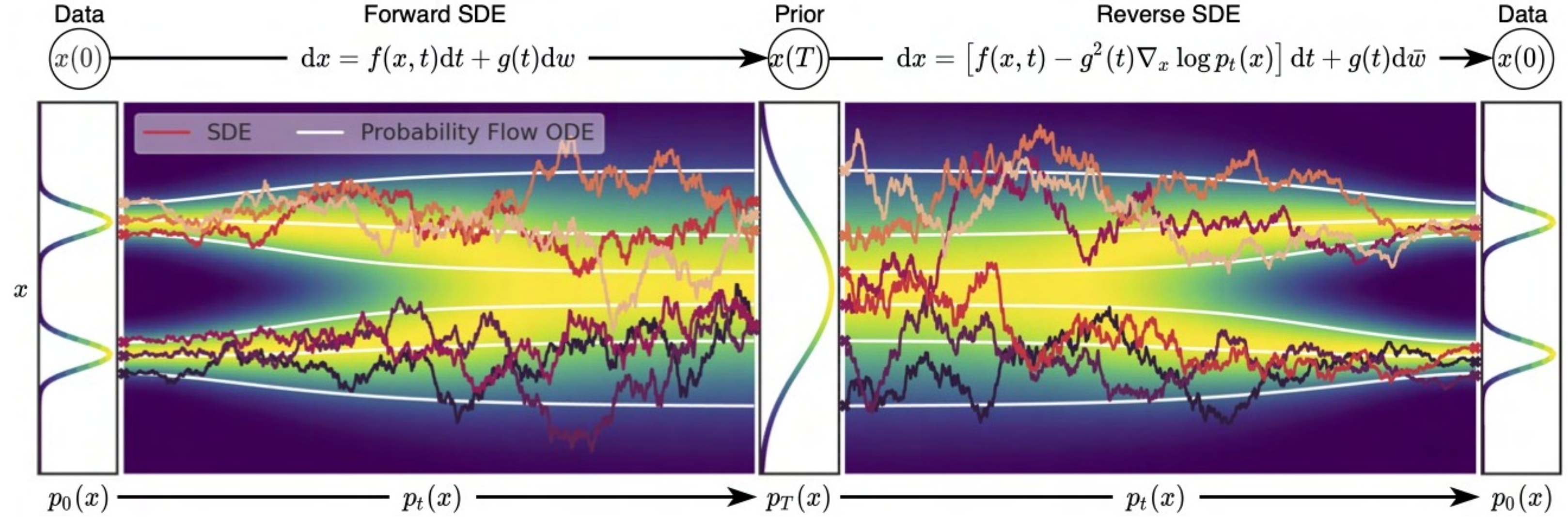
Score-based generative modeling with stochastic differential equations (SDEs)

Sampling from reverse SDE

$$\Delta \mathbf{x} \leftarrow [\mathbf{f}(\mathbf{x}, t) - g^2(t)\mathbf{s}_\theta(\mathbf{x}, t)]\Delta t + g(t)\sqrt{|\Delta t|}\mathbf{z}_t$$

Convert learned SDE to and ODE with the same distribution (probability flow ODE): allows computing likelihood!

$$\mathrm{d}\mathbf{x} = \left[\mathbf{f}(\mathbf{x}, t) - \frac{1}{2}g^2(t)\nabla_{\mathbf{x}} \log p_t(\mathbf{x})\right]\mathrm{d}t.$$



Score-Based Generative Modeling through Stochastic Differential Equations.

Score-matching for solving inverse problems

Given $P(Y|X)$ Solve $P(X|Y)$

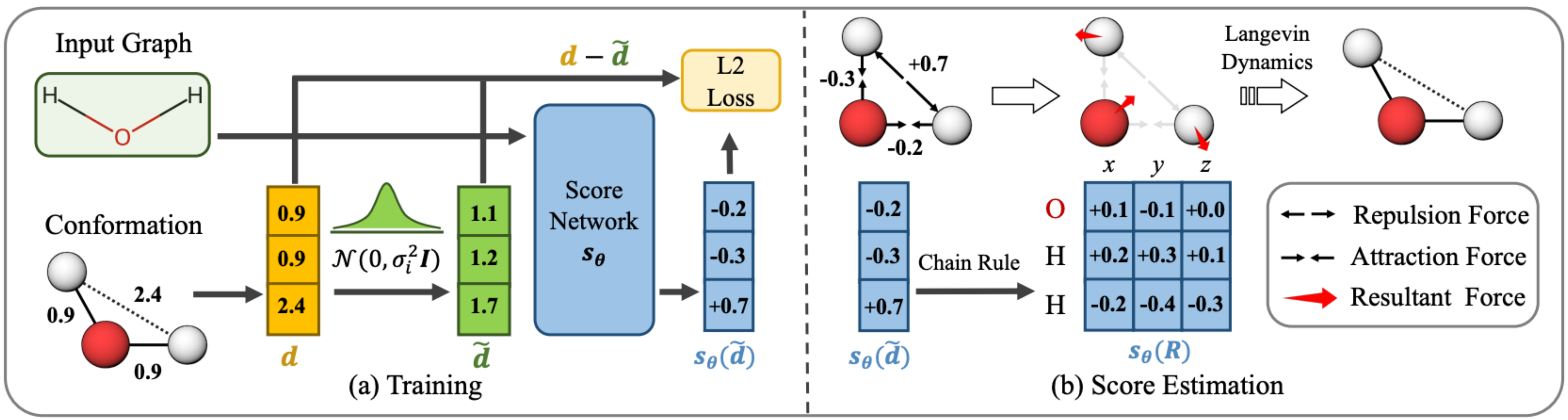
Inverse problems are typically a family of problems, which is easy to compute in one direction, but hard to compute in the reversed direction

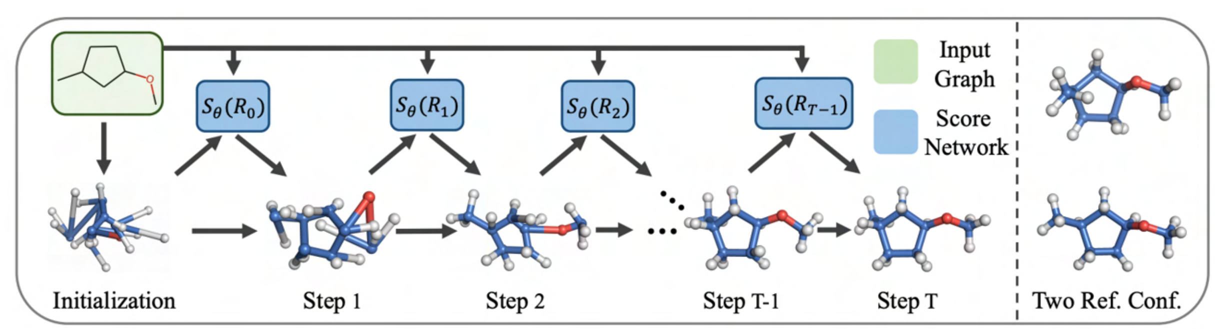
$$\nabla_{\mathbf{x}} \log p(\mathbf{x} \mid \mathbf{y}) = \nabla_{\mathbf{x}} \log p(\mathbf{x}) + \nabla_{\mathbf{x}} \log p(\mathbf{y} \mid \mathbf{x}).$$

Image colorization (x: color image, y: b/w image)

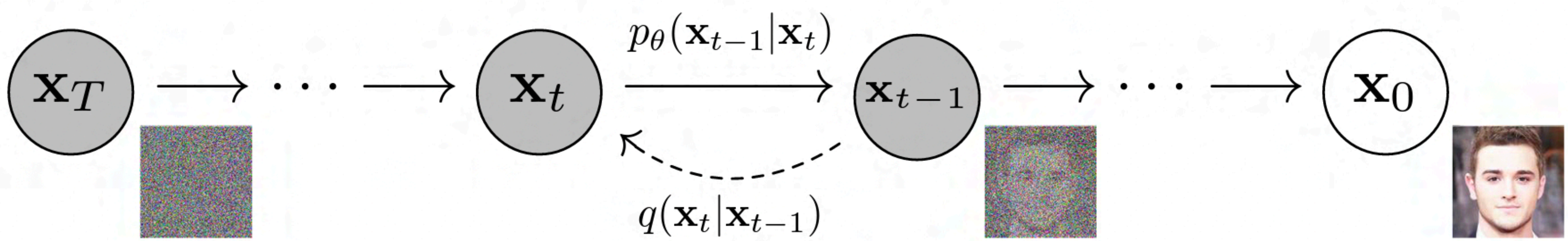
Application example: predicting 3D molecular structure

1. 3D equivariant representation of molecular structure with distances
2. Learn a conditional score network for distances with denoising score-matching
3. Sample by back-propagating gradient from distance to coordinates





Denoising diffusion probabilistic model



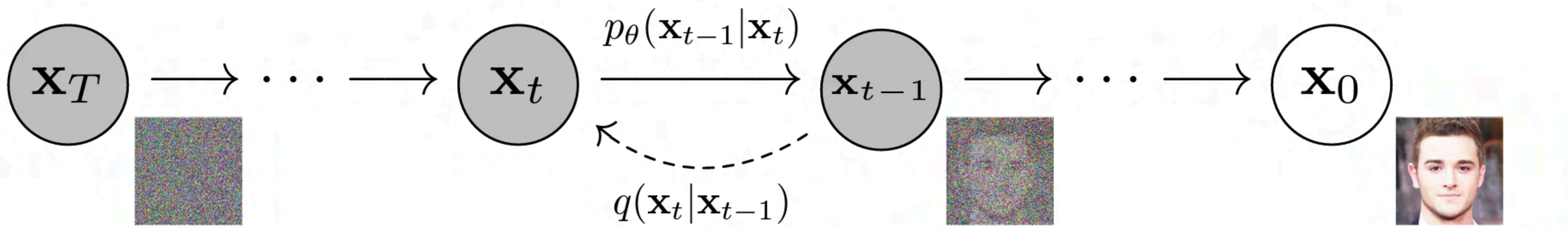
Forward “diffusion” process gradually add noise until reaching unit Gaussian distribution

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) := \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t \mathbf{I})$$

Multiple steps of diffusion is still described by Gaussian distribution

$$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I}) \qquad \alpha_t := 1-\beta_t \qquad \bar{\alpha}_t := \prod_{s=1}^t \alpha_s$$

Denoising diffusion probabilistic model



Variational ELBO objective

$$\mathbb{E}\left[-\log p_\theta(\mathbf{x}_0)\right] \leq \mathbb{E}_q\left[-\log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\right] = \mathbb{E}_q\left[-\log p(\mathbf{x}_T) - \sum_{t\geq 1} \log \frac{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_t|\mathbf{x}_{t-1})}\right]$$

Which simplifies to $$\mathbb{E}_{\mathbf{x}_0,\boldsymbol{\epsilon}}\left[\frac{\beta_t^2}{2\sigma_t^2\alpha_t(1-\bar{\alpha}_t)}\left\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t)\right\|^2\right] + \log p_\theta(x_0 \mid x_1)$$

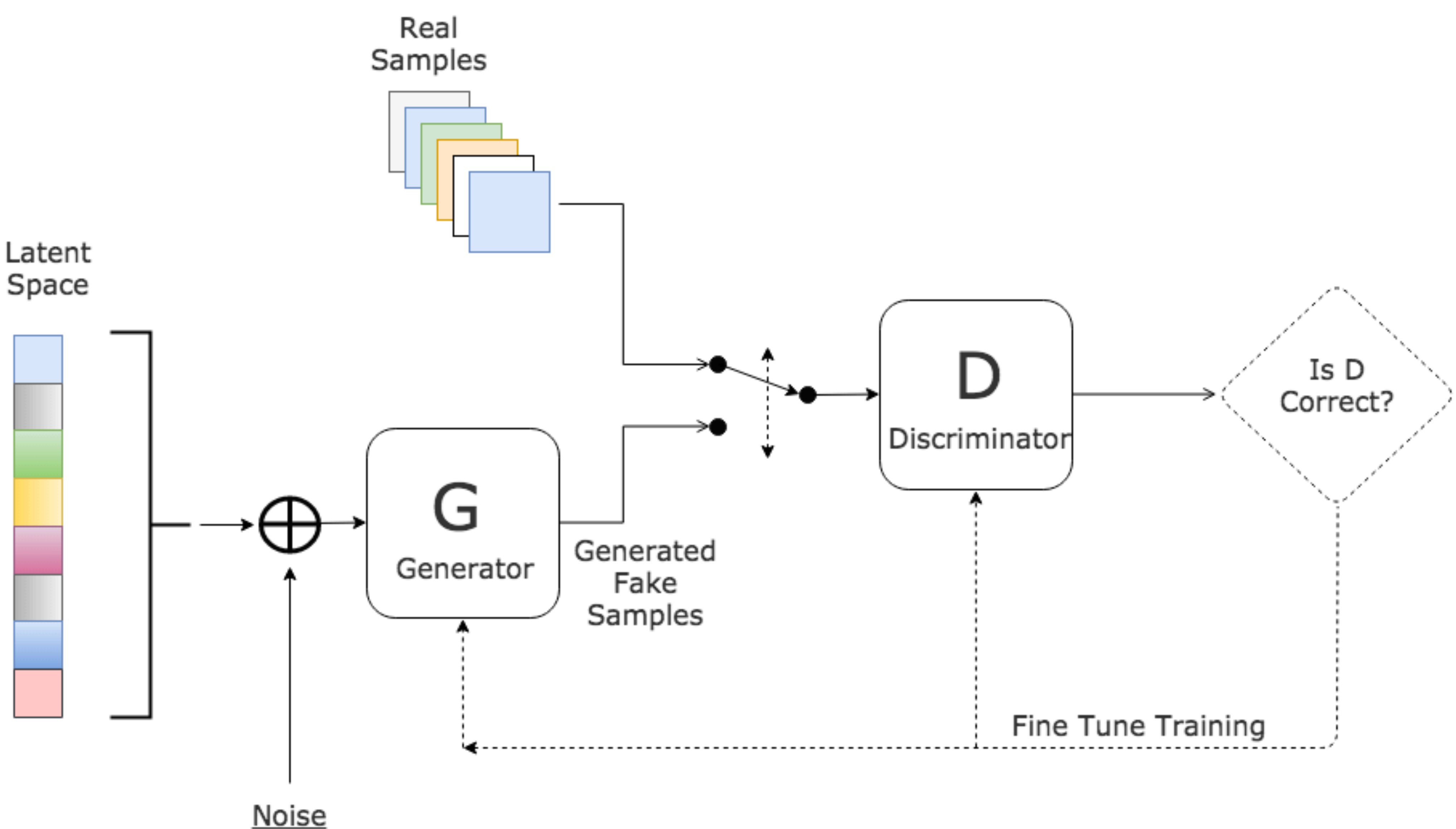
Simplified objective typically works better

$$L_{\mathrm{simple}}(\theta) := \mathbb{E}_{t,\mathbf{x}_0,\boldsymbol{\epsilon}}\left[\left\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t)\right\|^2\right]$$

p_θ is typically defined to be Gaussian and with variance matching the forward diffusion process

Probabilistic modeling with neural networks: Learn to sample

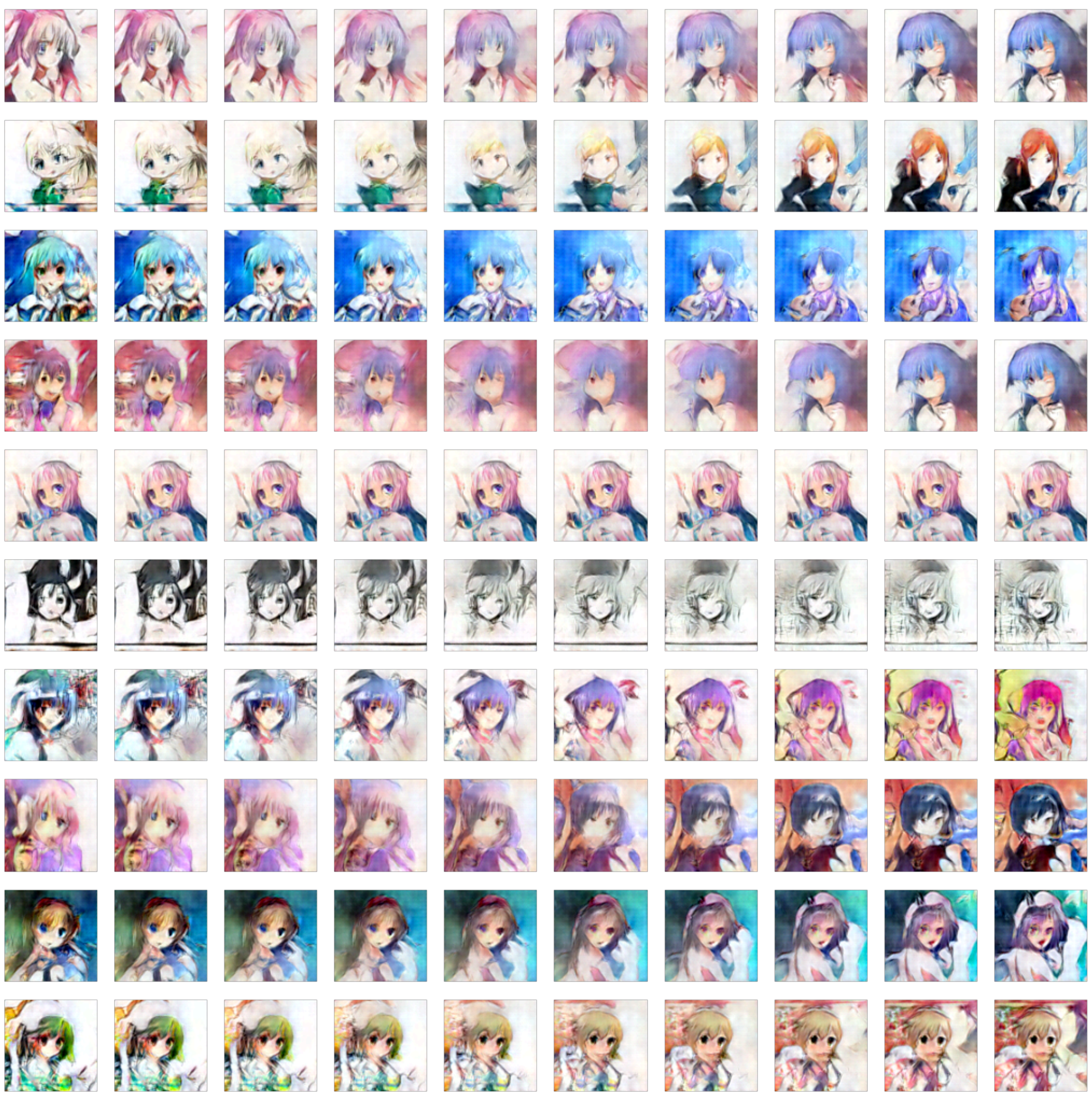
Generative adversarial networks



$$\min_{\theta} \max_{\phi} V(G_\theta, D_\phi) = \mathbb{E}_{\mathbf{x} \sim \mathbf{p}_{\text{data}}}[\log D_\phi(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})}[\log(1 - D_\phi(G_\theta(\mathbf{z})))]$$

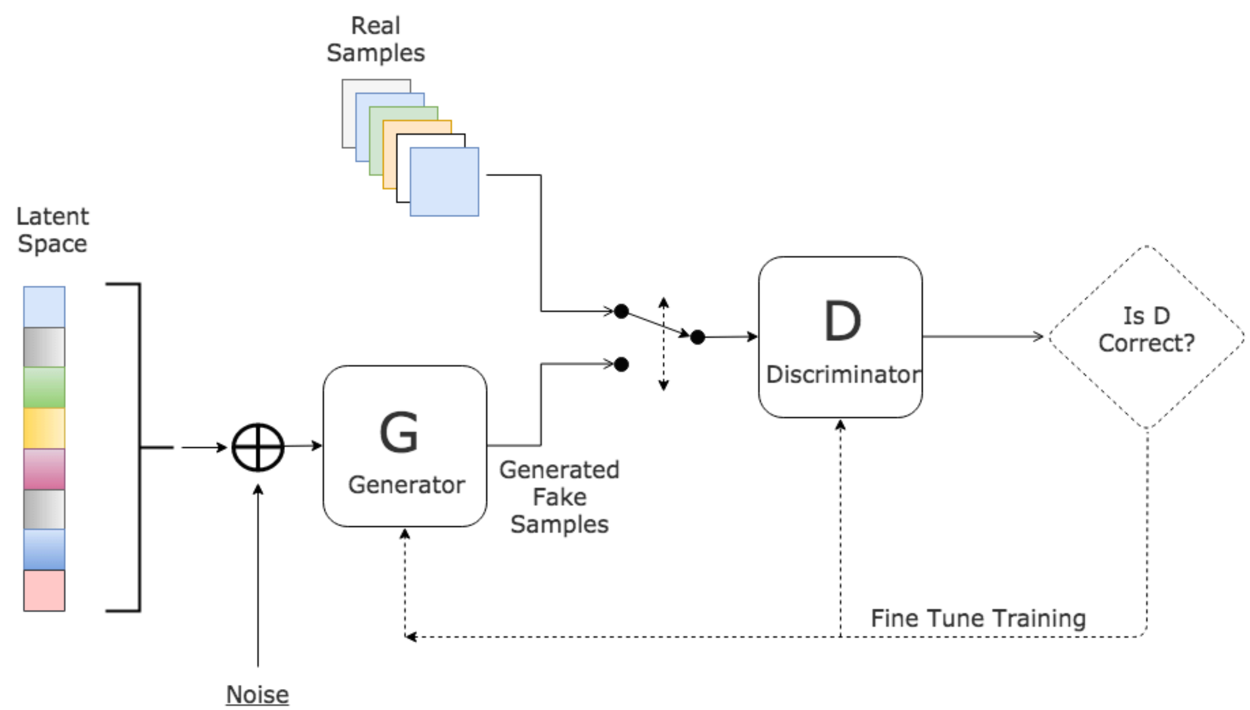
Probabilistic modeling with neural networks: Learn to sample

Generative adversarial networks



StyleGAN

Formulating Generative adversarial networks as a proper probabilistic model



$$P(x) = \frac{1}{Z} \exp f(x)$$

$$\frac{\partial \log P}{\partial \theta} = E_{x\sim data}(\frac{\partial f(x)}{\partial \theta}) - E_{x\sim model}(\frac{\partial f(x)}{\partial \theta})$$

Generator network: use x~Generator instead of x~model

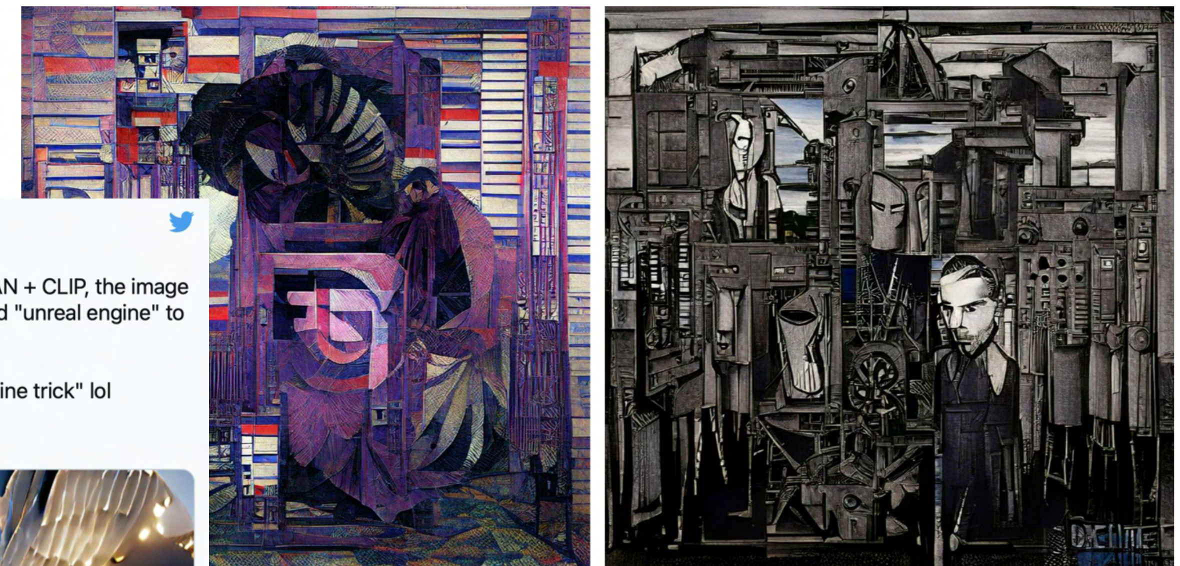
Discriminator network: $f(x)$

Wasserstein GAN objective: $E_{x\sim \text{data}} f(x) - E_{x\sim \text{generator}} f(x)$

Sentence-guided generation: VQGAN + CLIP

“Planetary City C” from VQ-GAN+CLIP (source: @RiversHaveWings on Twitter)

“Dancing in the moonlight” from VQ-GAN+CLIP (source: @advadnoun on Twitter)

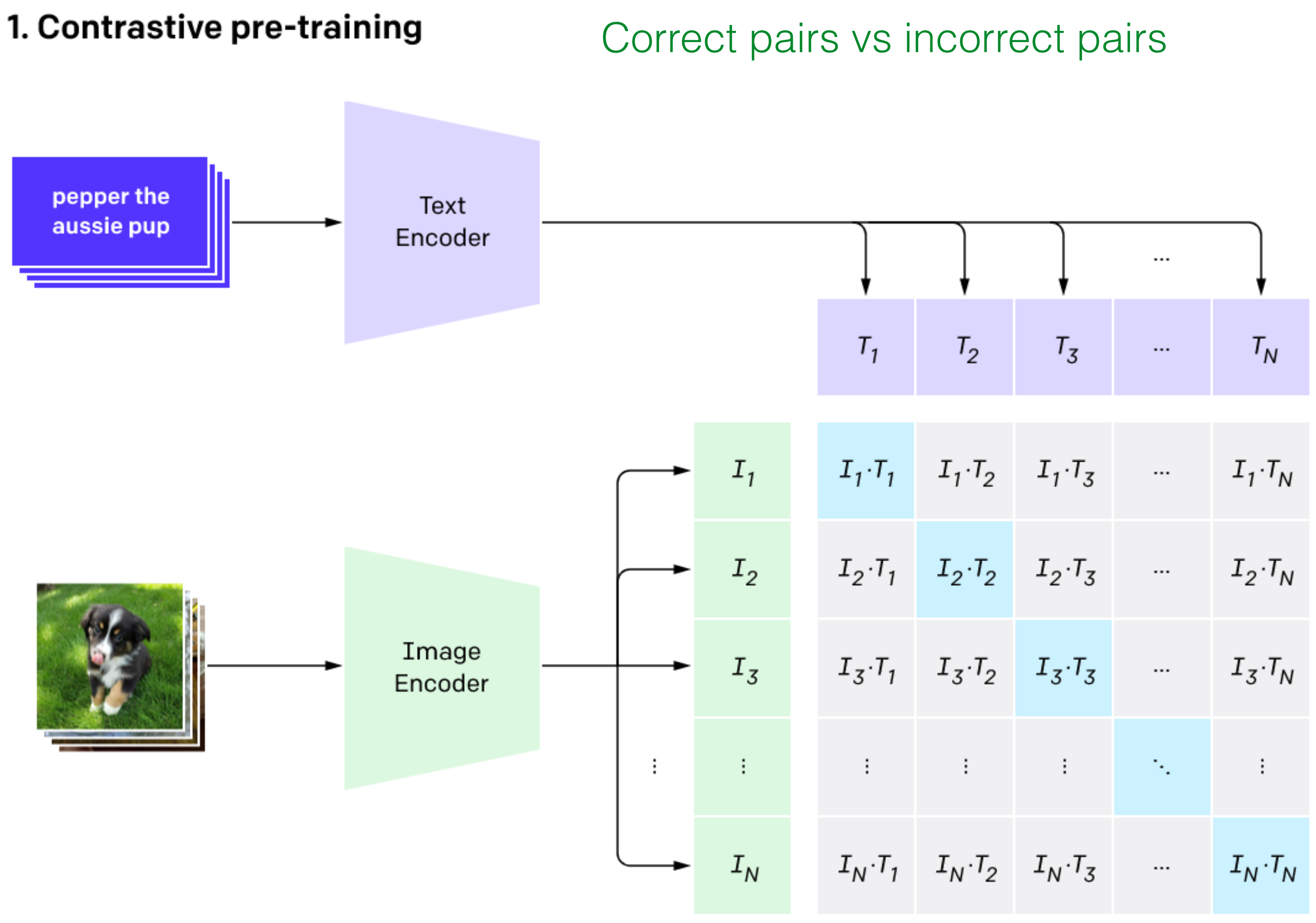


re” from VQ-GAN+CLIP (source: @RiversHaveWings on Twitter)

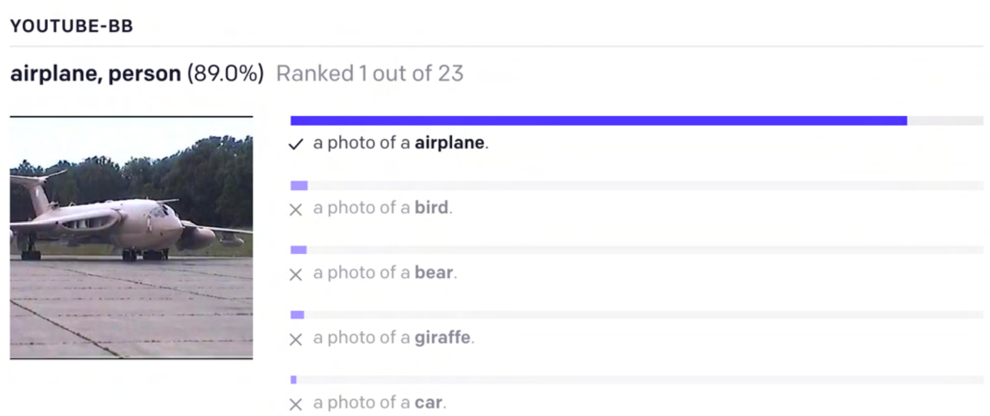
Sentence-guided generation: VQGAN + CLIP

CLIP: embed sentence and image to the same space

Correct pairs vs incorrect pairs

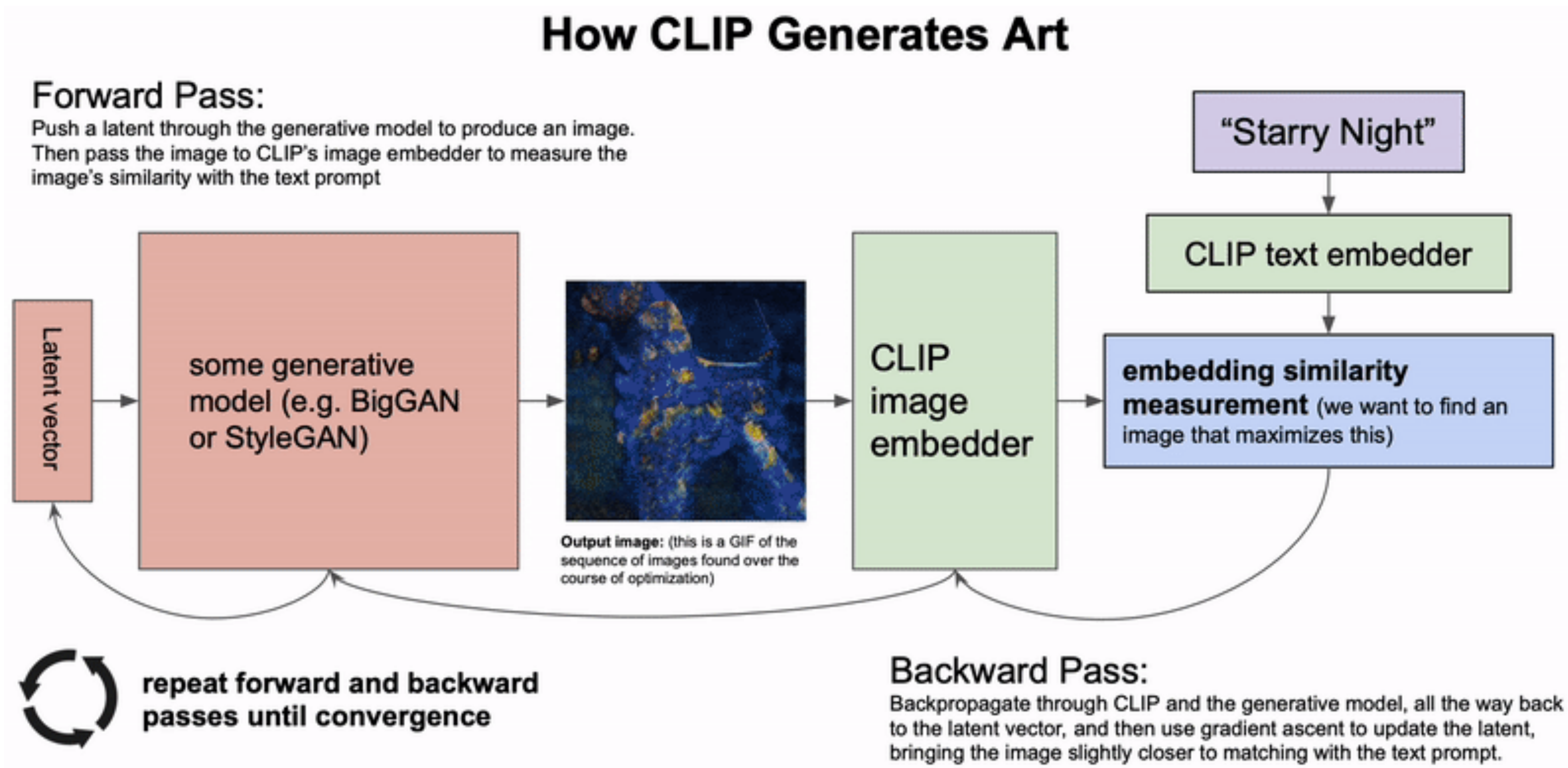


Original application:
Text choices



https://openai.com/blog/clip/

Sentence-guided generation: VQGAN + CLIP

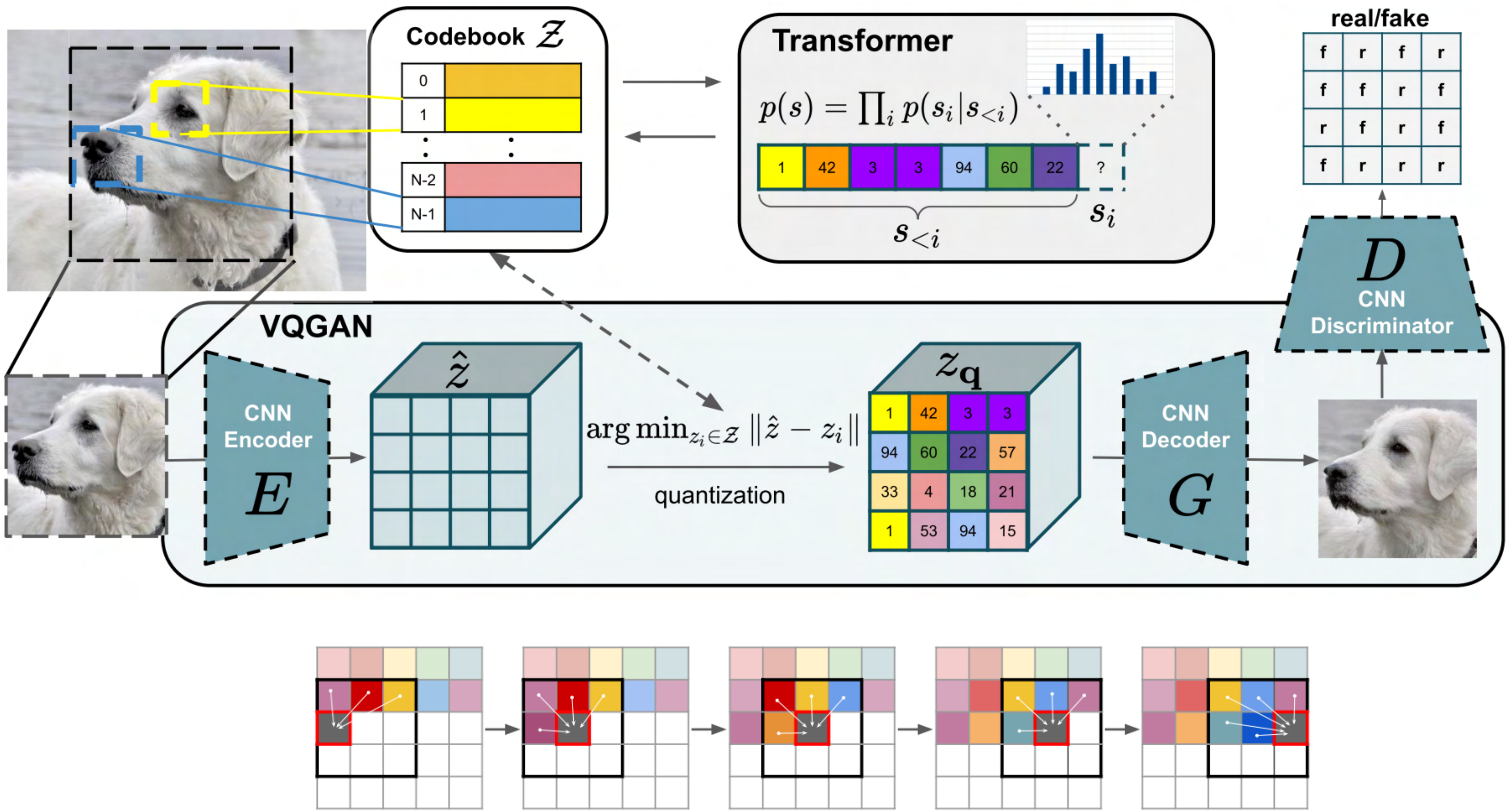


https://ml.berkeley.edu/blog/posts/clip-art/

Sentence-guided generation: VQGAN + CLIP

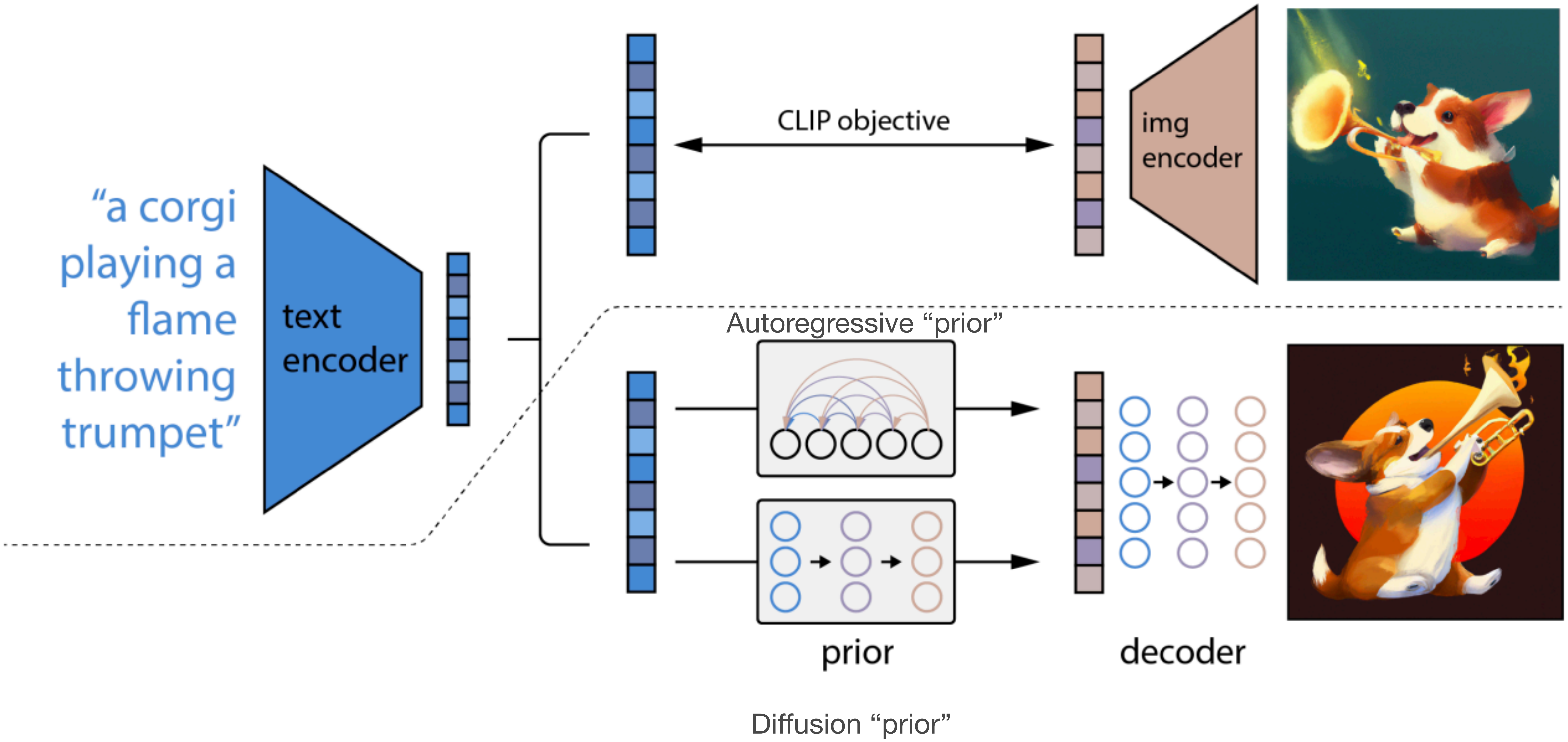
Encoder-Decoder architecture similar to VAE (efficient sampling in Z space)

Discrete Z latent space distribution (prior) with transformer modeling



Taming Transformers for High-Resolution Image Synthesis

DALLE-2 Replace optimization-based generation with “prior”+decoder



DALLE-2 Replace optimization-based generation with “prior”+decoder

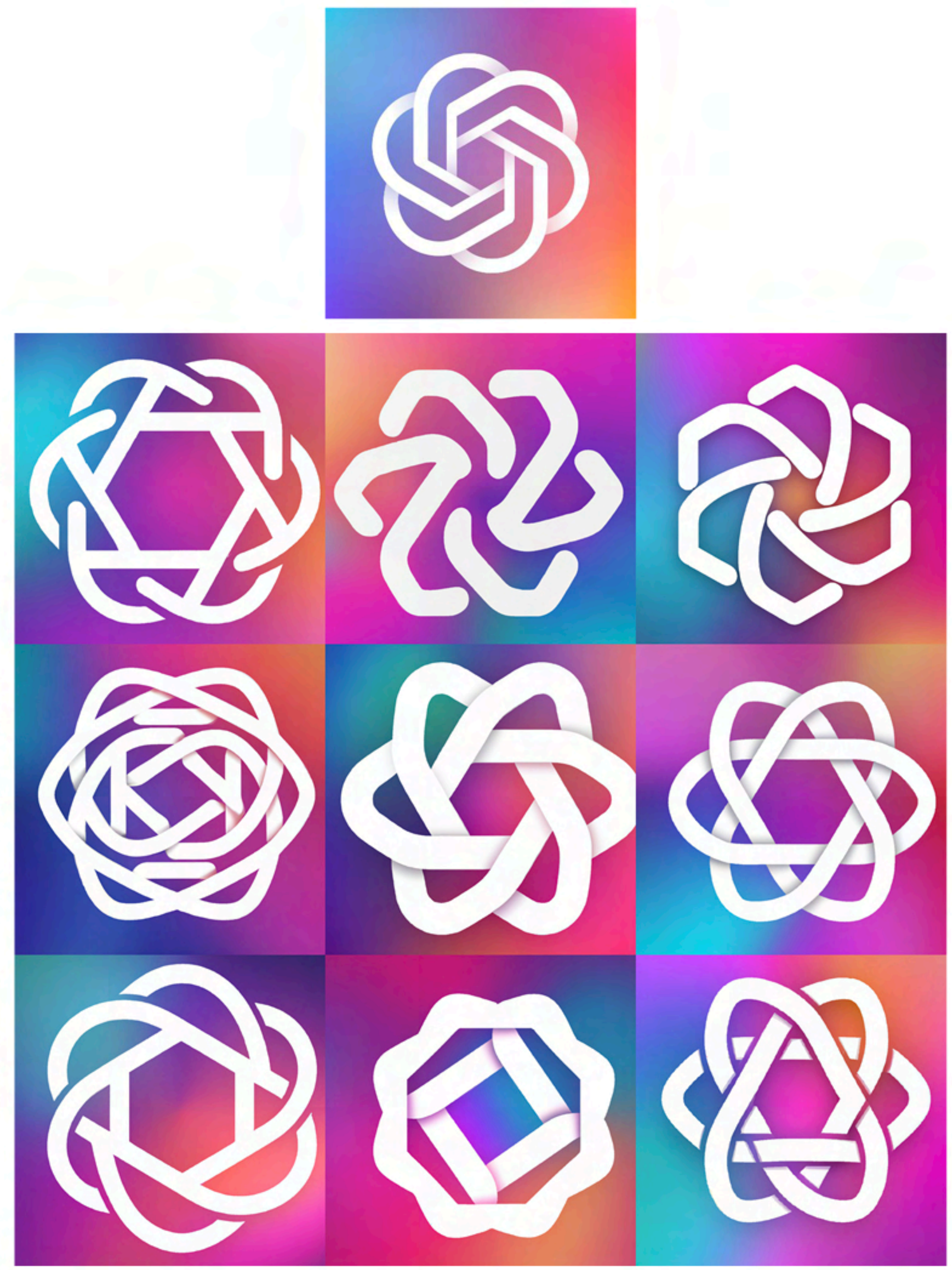
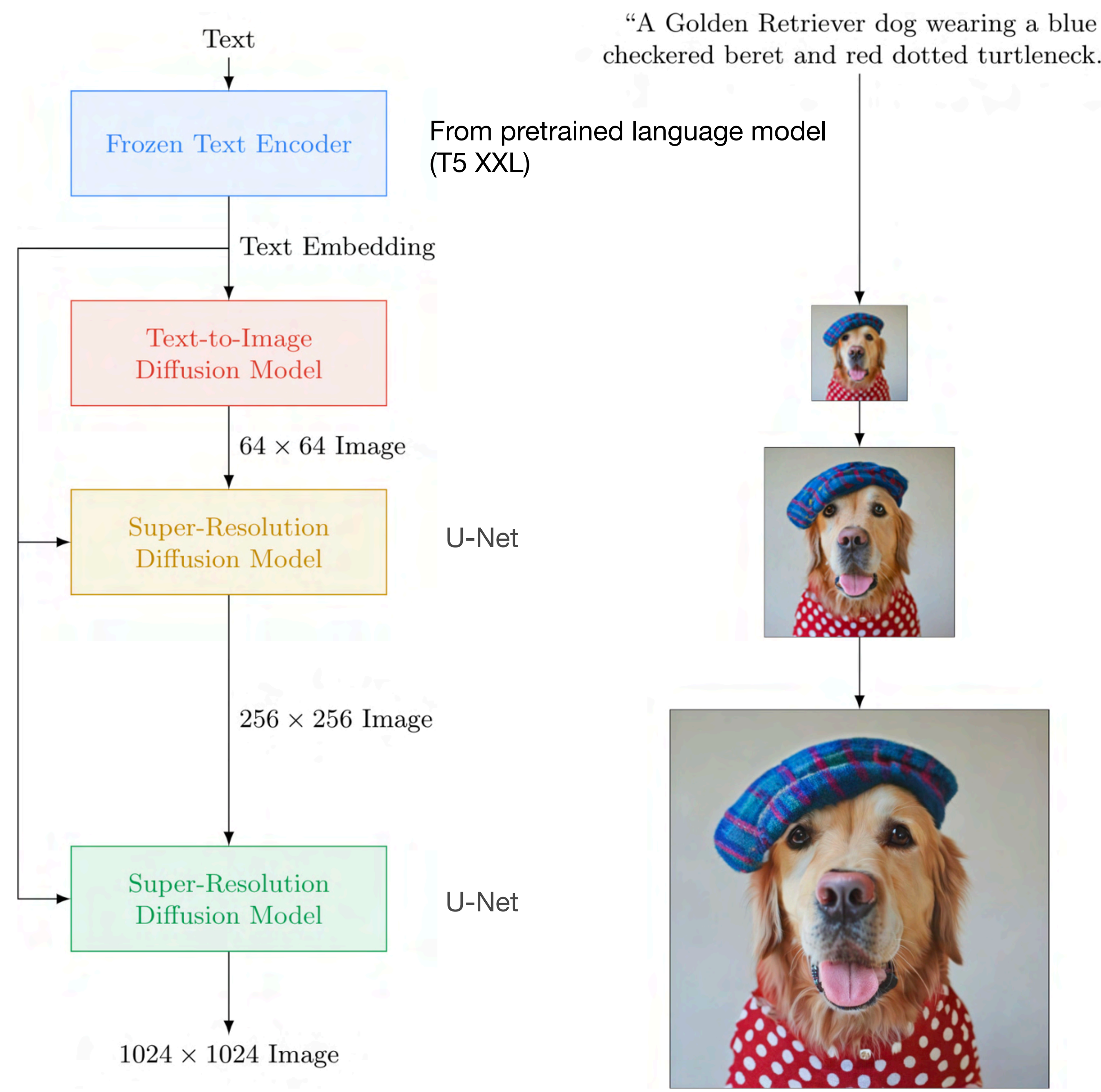


ImaGen: CLIP-free GAN generator architecture for image generation

A dragonfruit wearing karate belt in the snow



Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding

Reinforcement learning

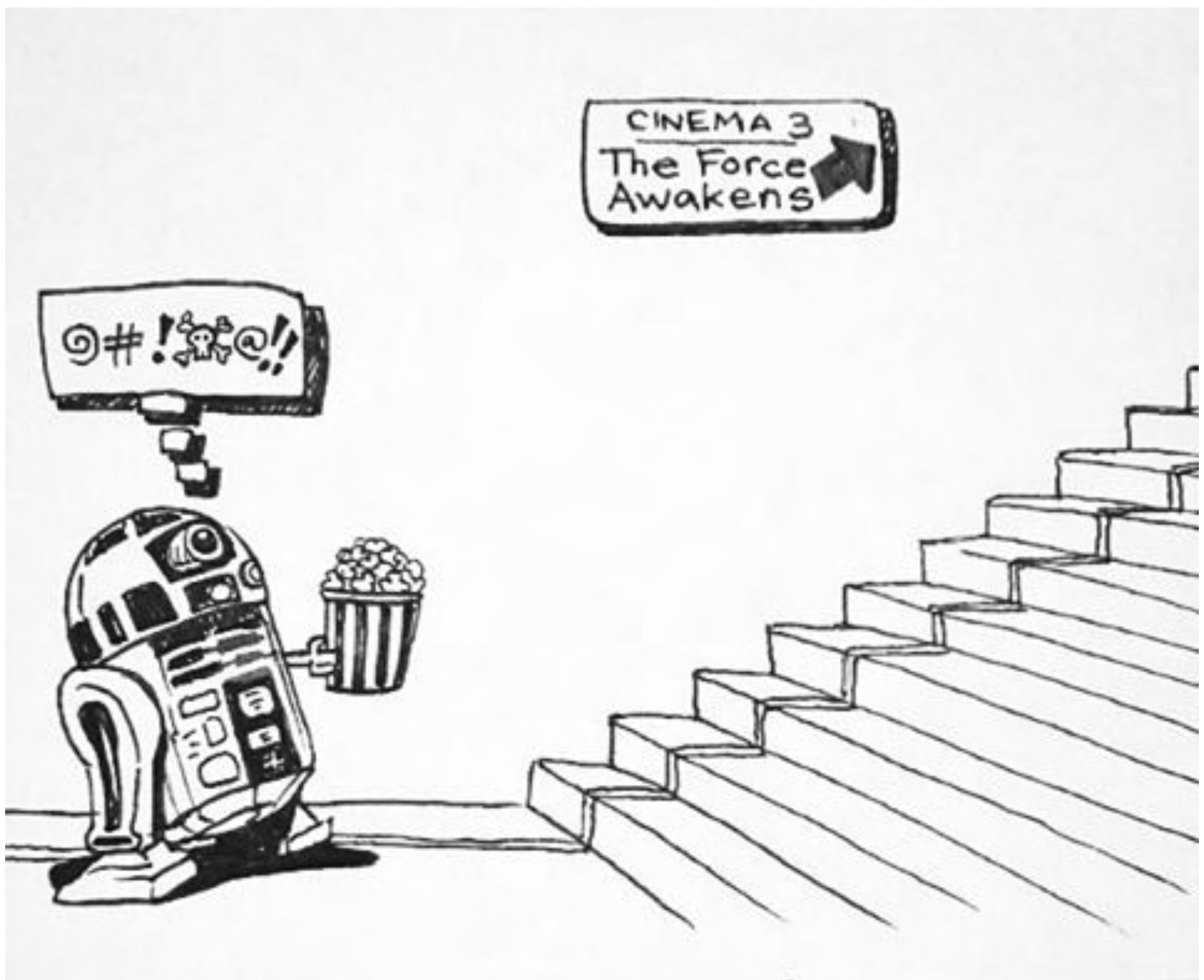
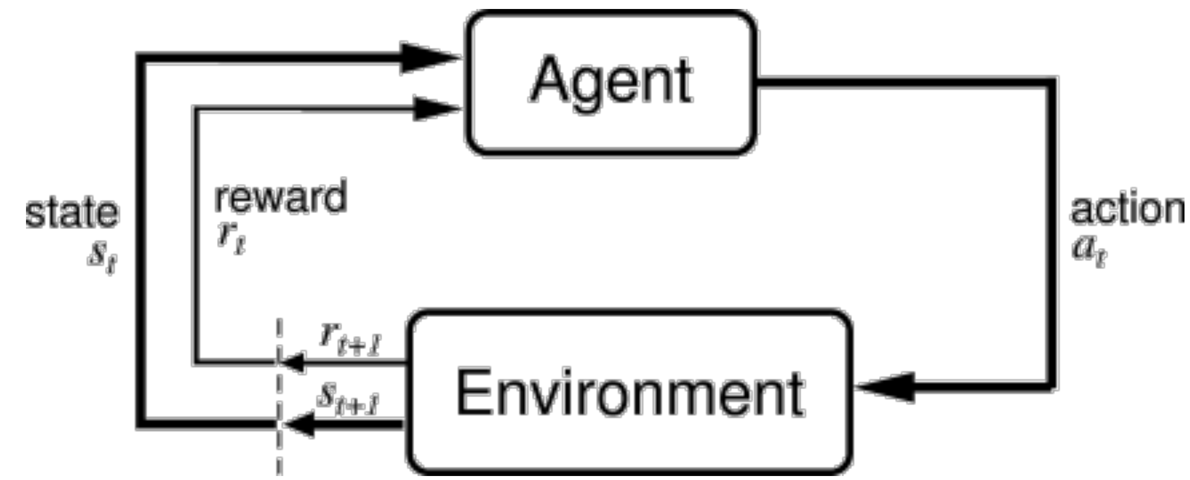


Image credit: daily.doodl @ instagram



Given state, choose action, get reward

Deep Q learning:
Predict future rewards with deep networks

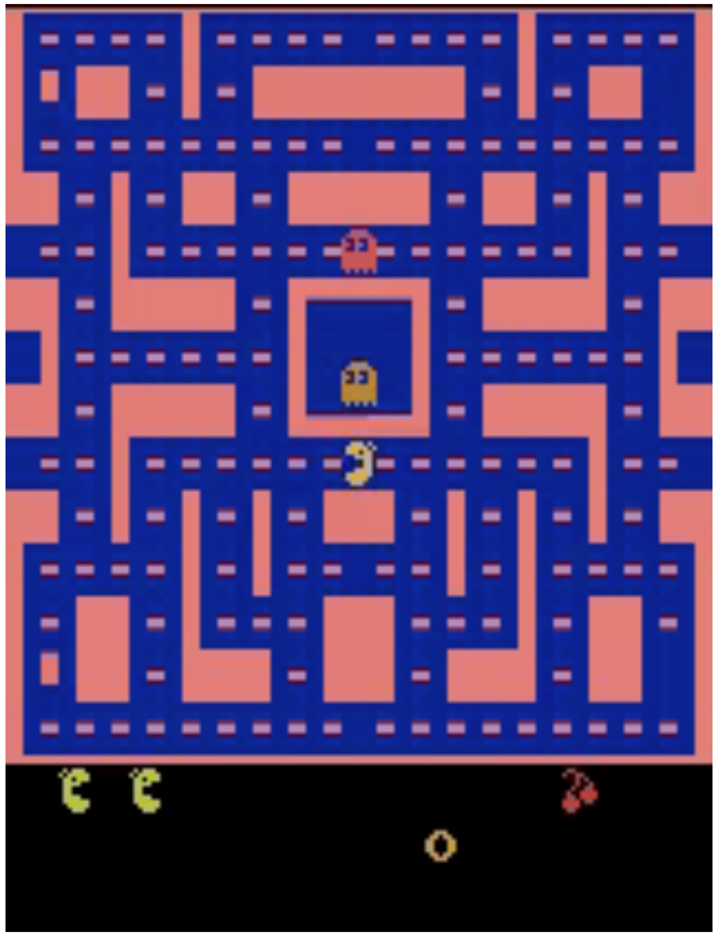
Q Learning

Q(state, action) = maximal future rewards (with the optimal actions)

Bellman equation

$$\underline{Q(s,a)} = r \ \underline{+ \gamma max_{a'} Q(s',a')}$$

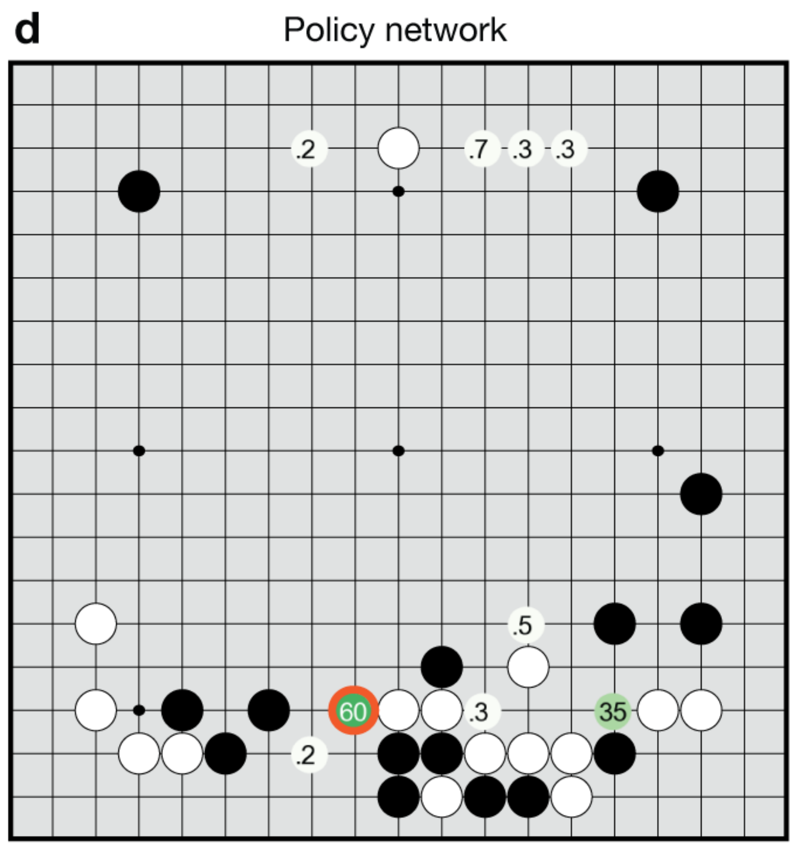
Training: minimize MSE

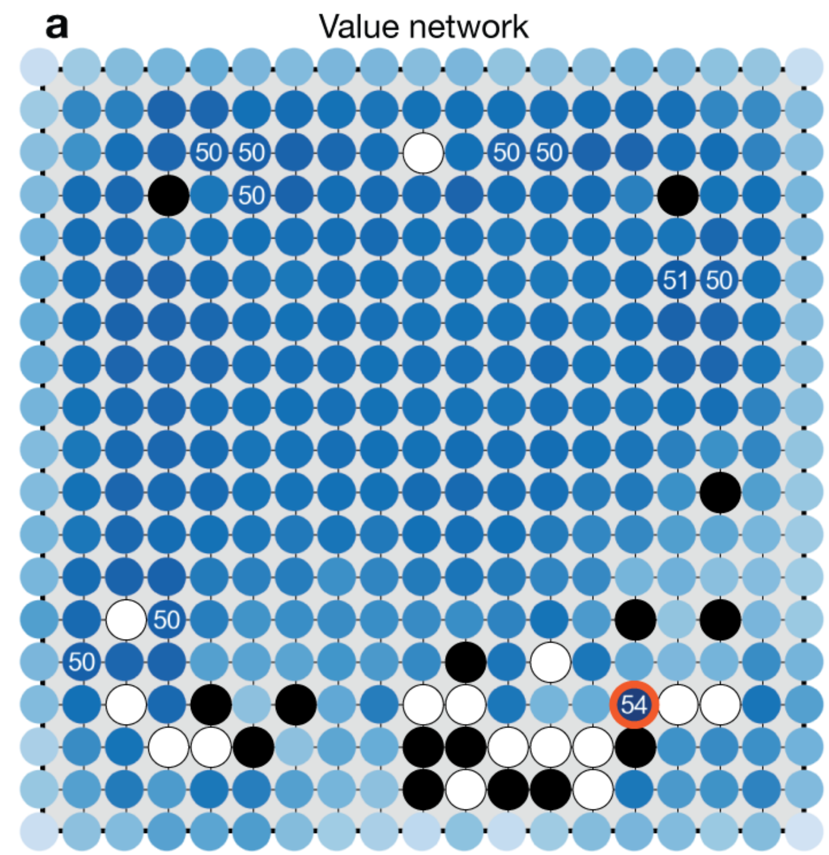


Minh et al, 2013 Playing Atari with Deep Reinforcement Learning

AlphaGo - surpass human-level game playing in Go

(the nature publication version)





SL policy network: predict expert human moves
convnet / GLM

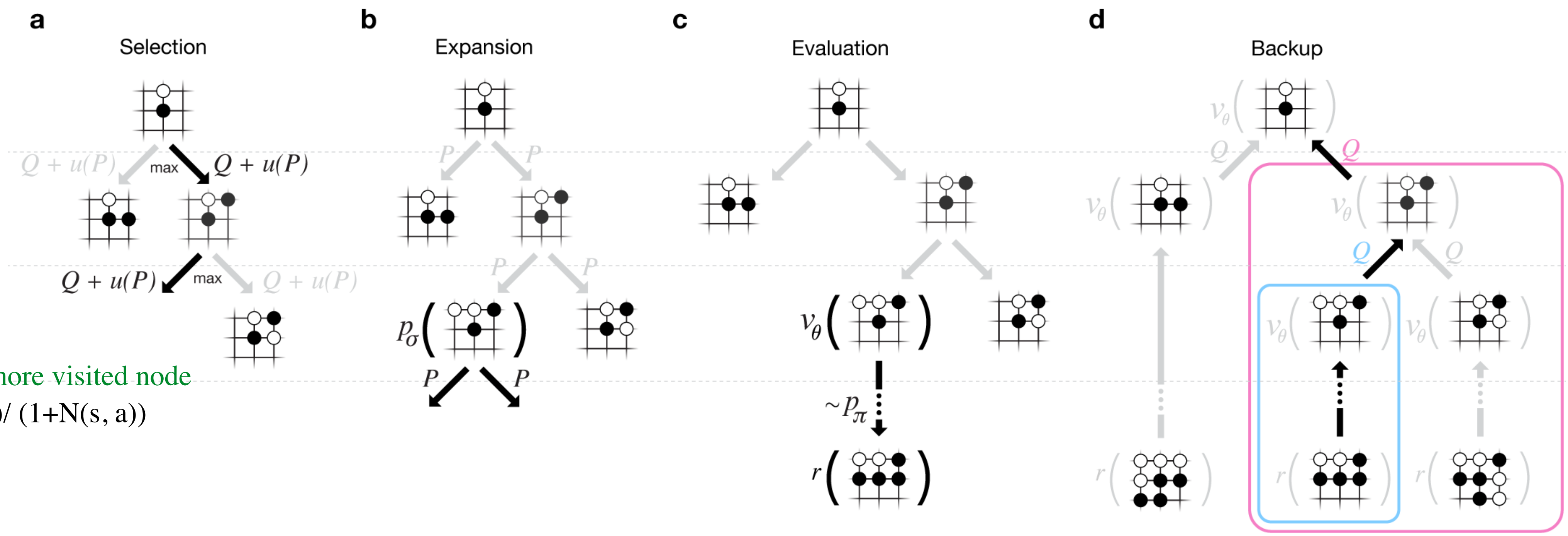
RL policy network: optimized by self-play
convnet

REINFORCE algorithm (Williams, 1992)

Value network: predict outcome of self-play
convnet

Silver et al., 2016, Mastering the game of Go with deep neural networks and tree search

AlphaGo - Monte carlo tree search

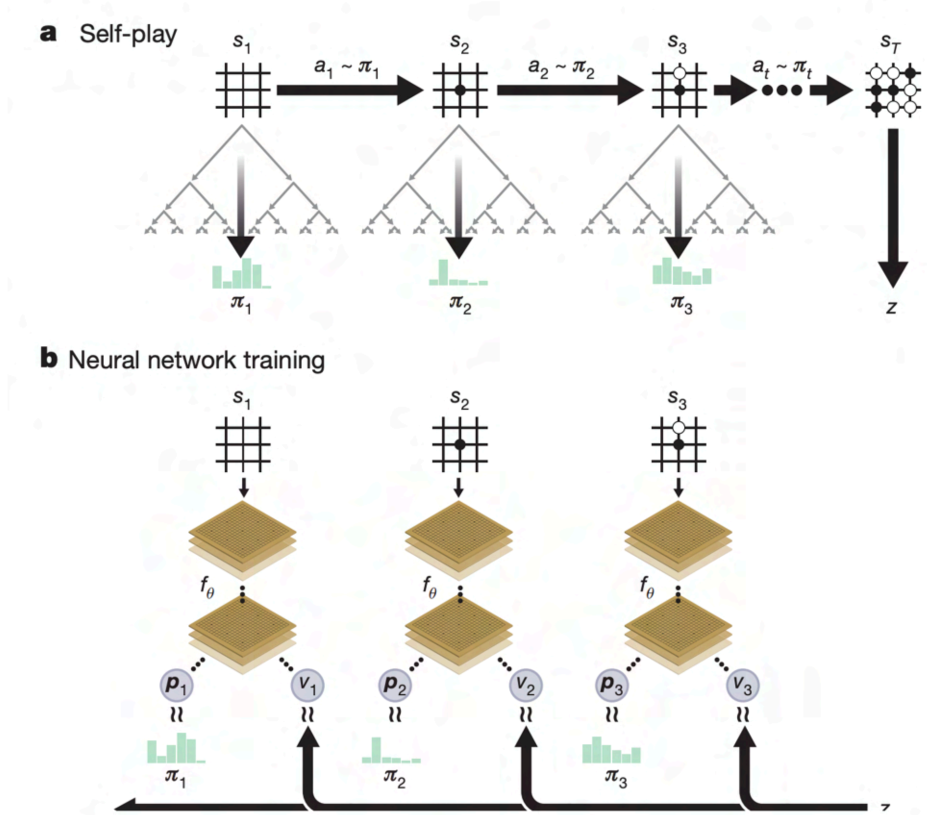


Discounting more visited node

u(s, a)∝P(s, a)/ (1+N(s, a))

Final game play:

$$\pi_a \propto N(s,a)^{1/\tau}$$

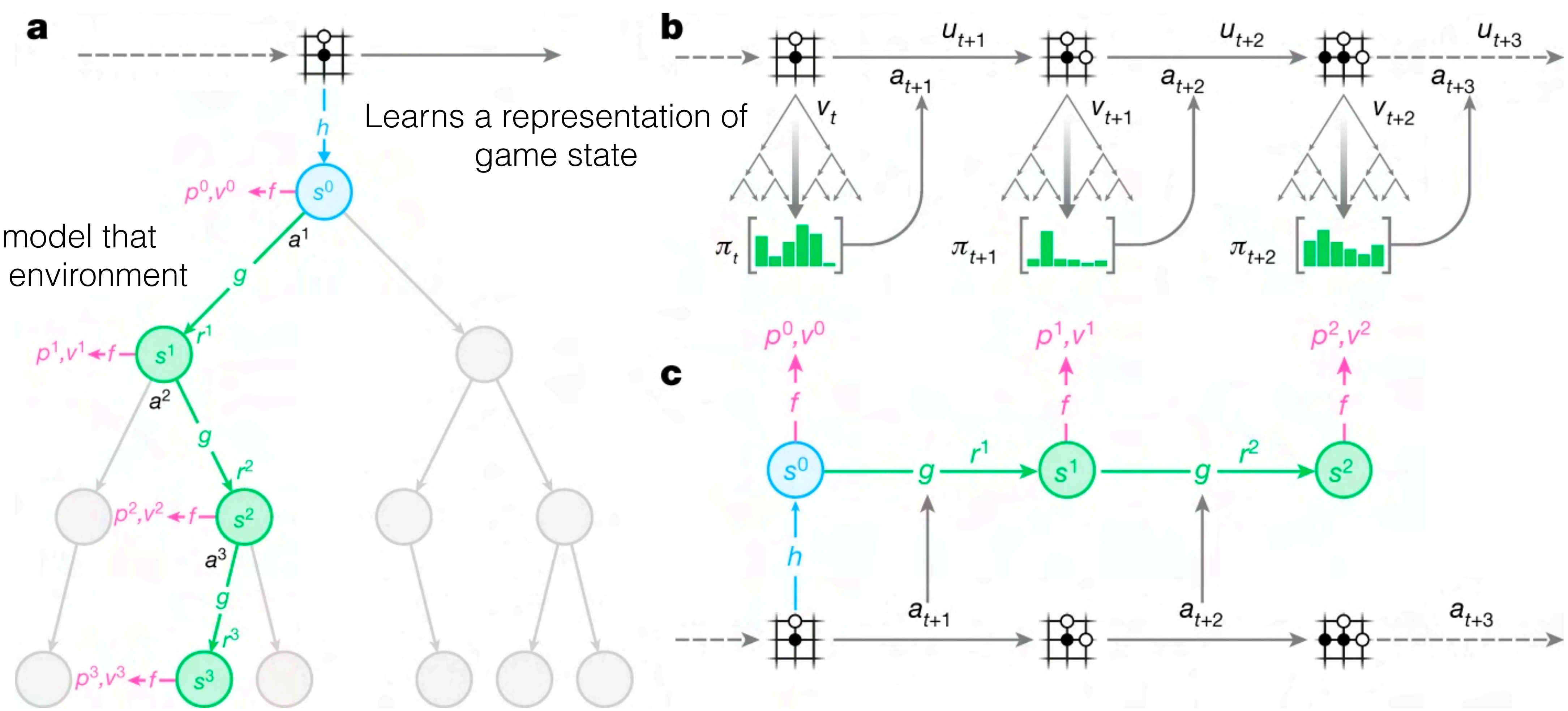


AlphaGo Zero:

Train policy network using MCTS policy

Learning without access to environment during planning (MCTS)

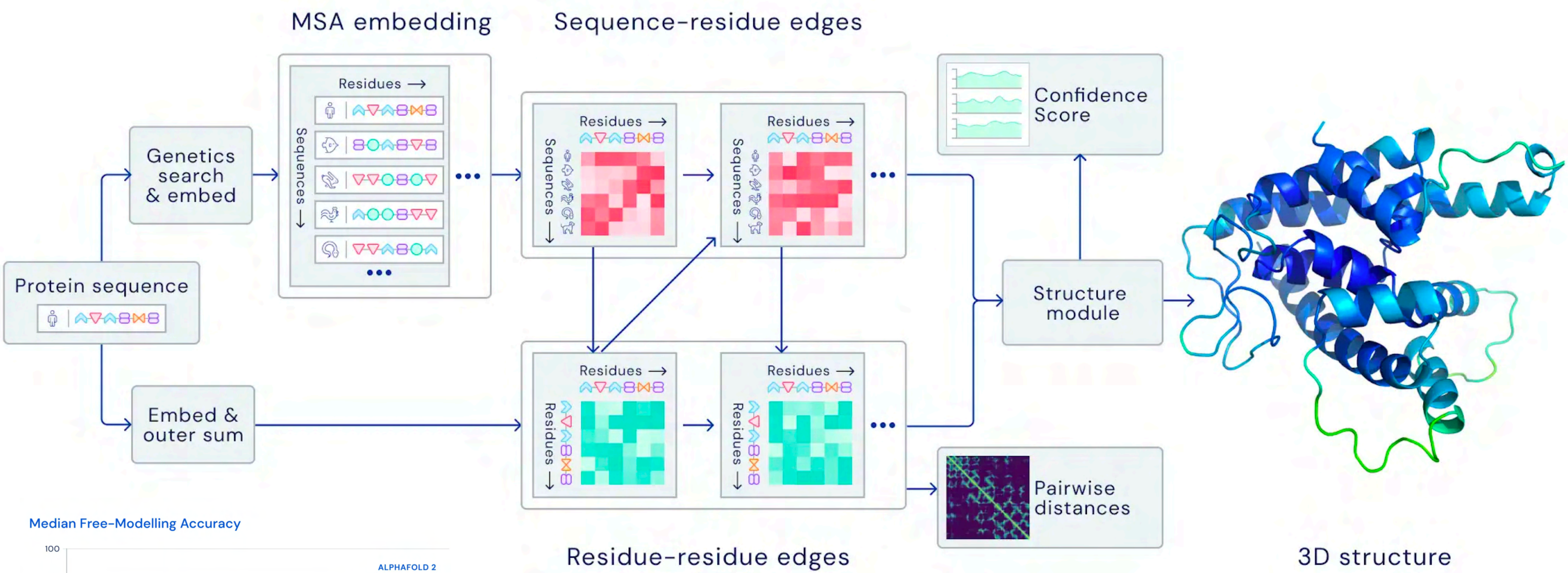
MuZero:
Training model without access to the environment during MCTS

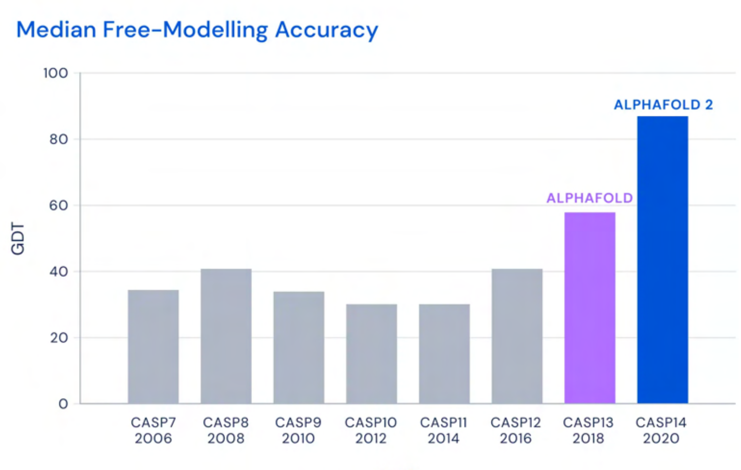


Learns a representation of game state

Dynamics model that captures the environment

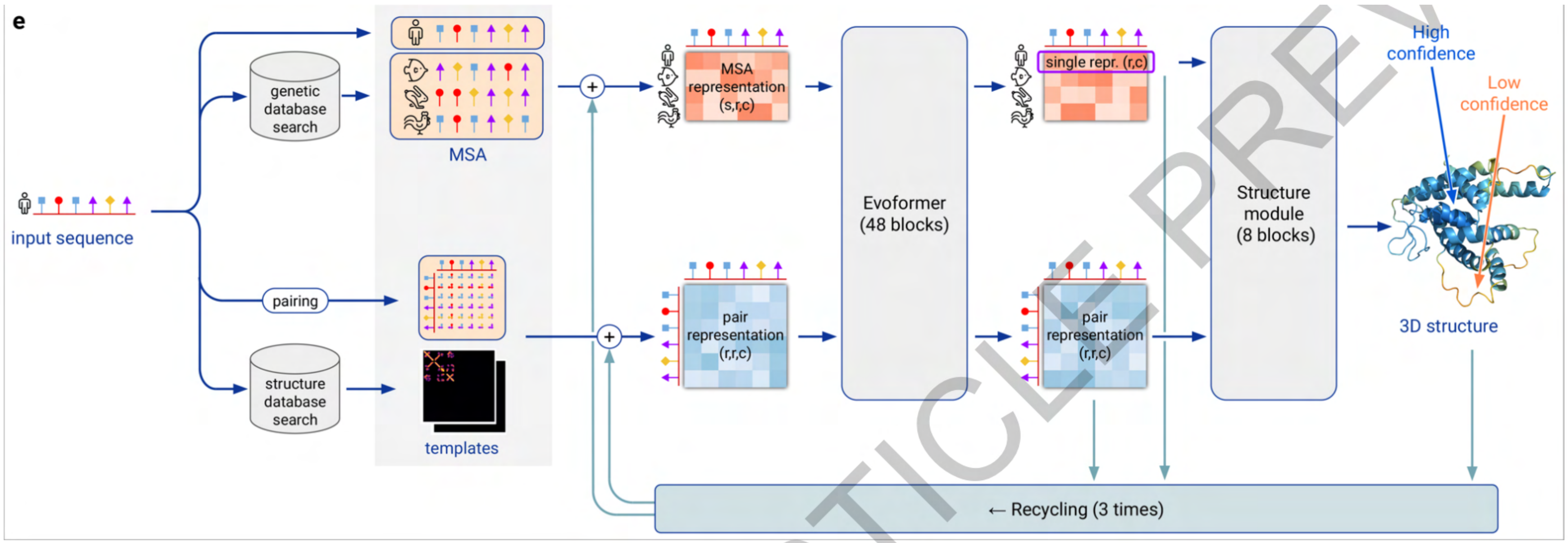
AlphaFold2 - X-ray level atomic resolution prediction



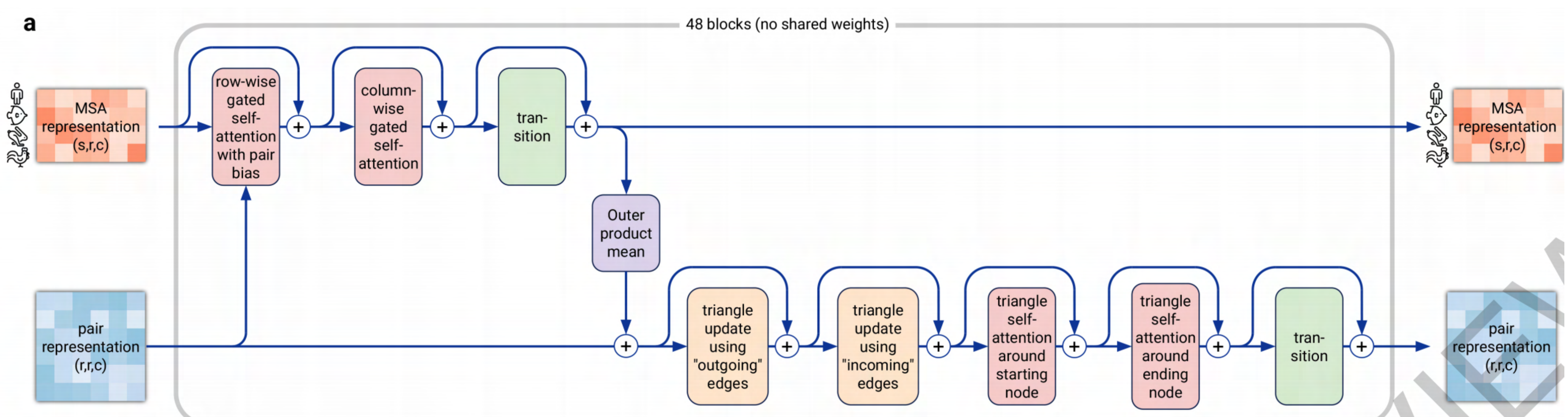


AlphaFold v2.0

Overall structure



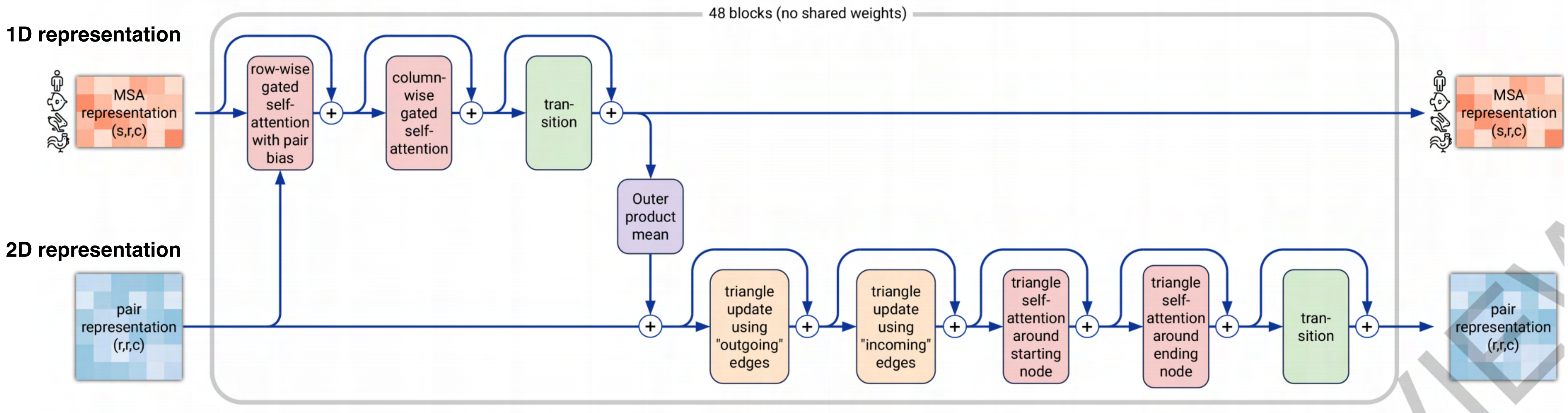
Sequence model structure



Jumper et al., 2021

AlphaFold v2.0

Sequence model structure



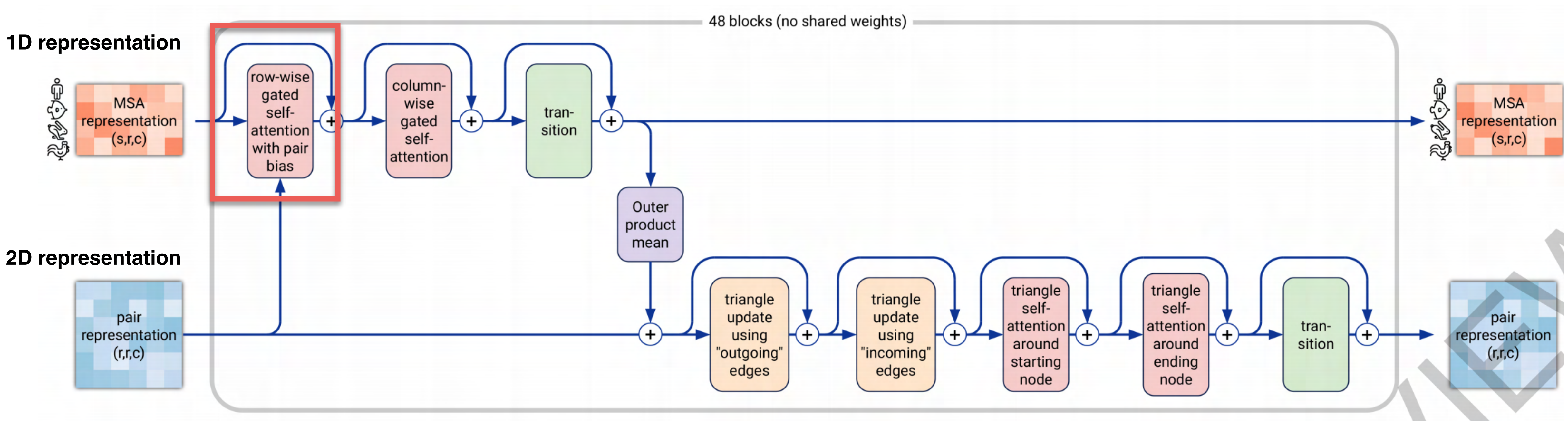
Template protein data also included (torsion angles)

+

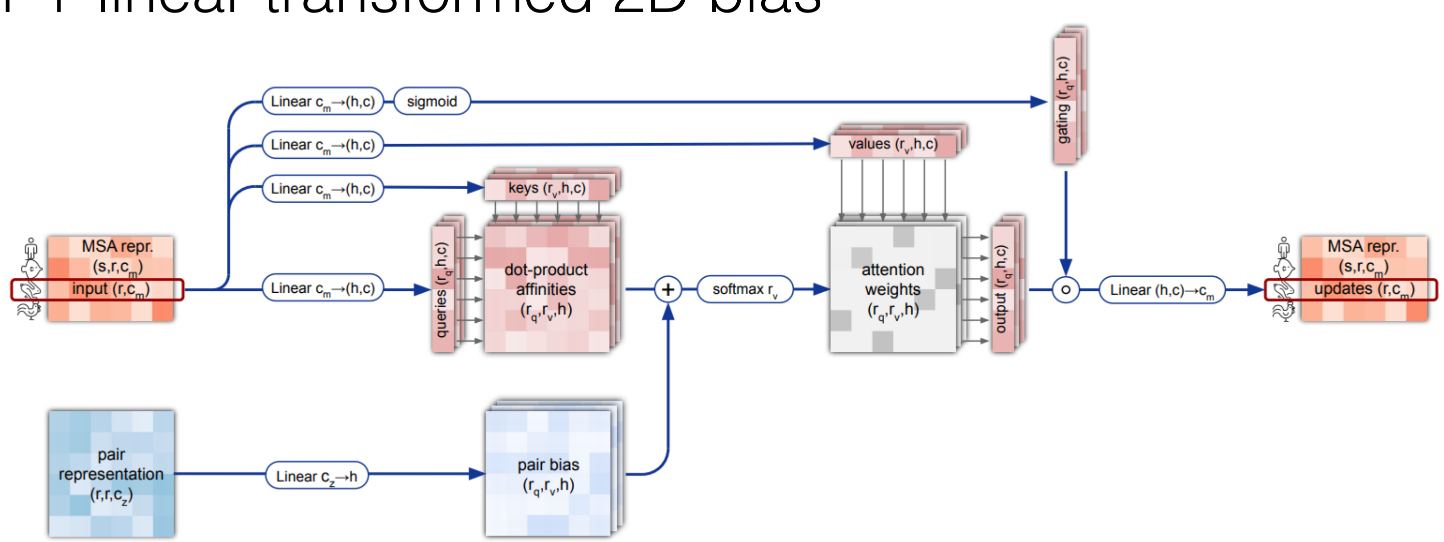
Jumper et al., 2021

AlphaFold v2.0 : model structure

Sequence model structure



Gated transformer + linear transformed 2D bias

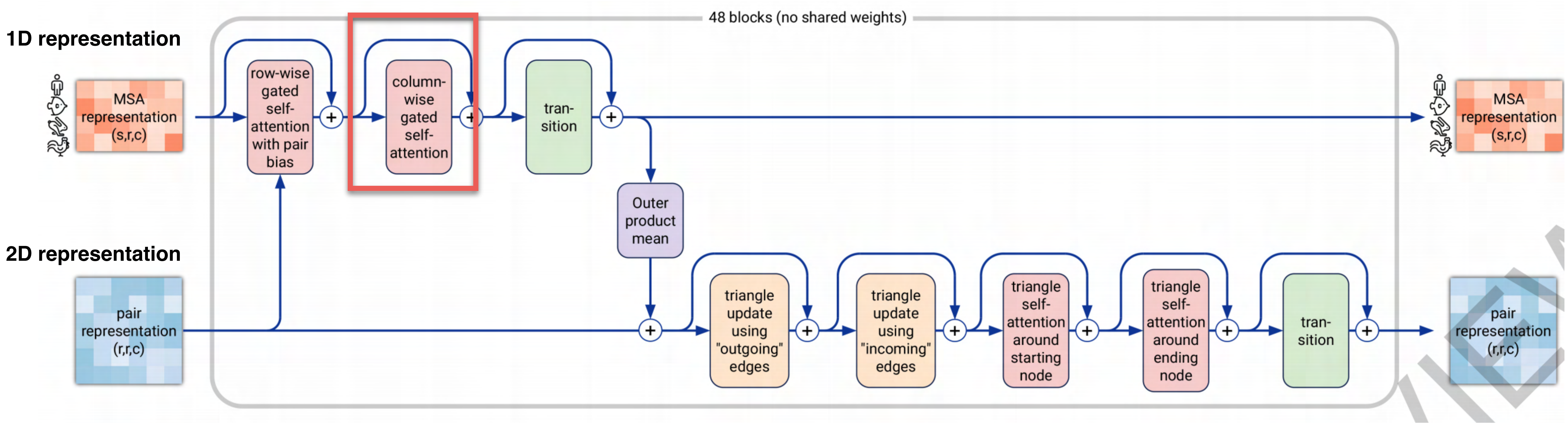


Supplementary Figure 2 | MSA row-wise gated self-attention with pair bias. Dimensions: s: sequences, r: residues, c: channels, h: heads.

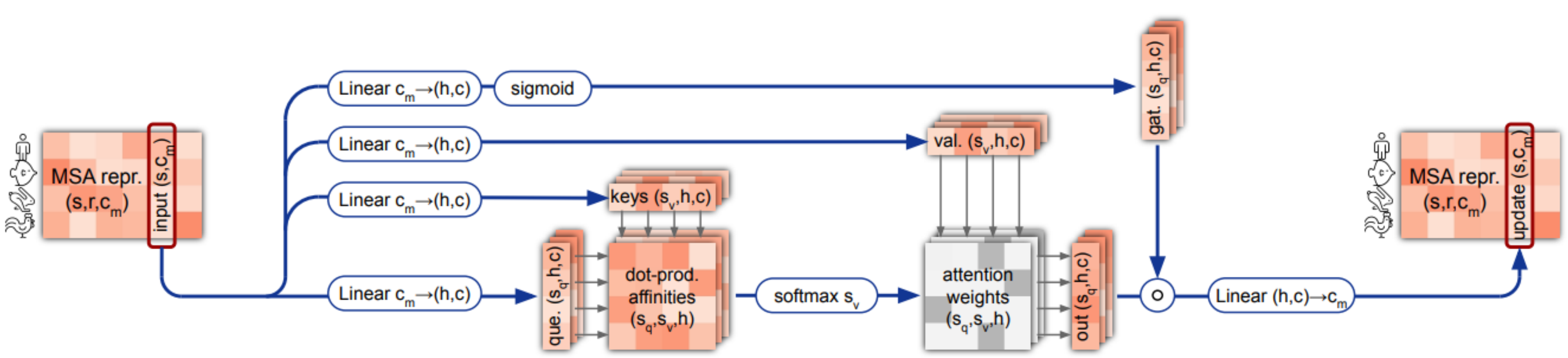
Jumper et al., 2021

AlphaFold v2.0 : model structure

Sequence model structure



Gated transformer

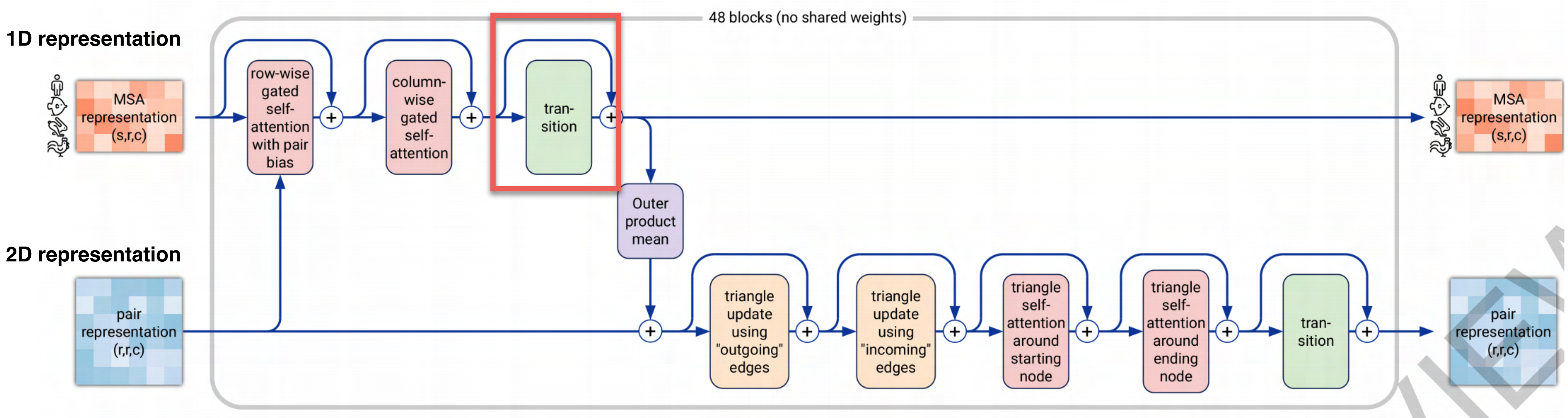


Supplementary Figure 3 | MSA column-wise gated self-attention. Dimensions: s: sequences, r: residues, c: channels, h: heads.

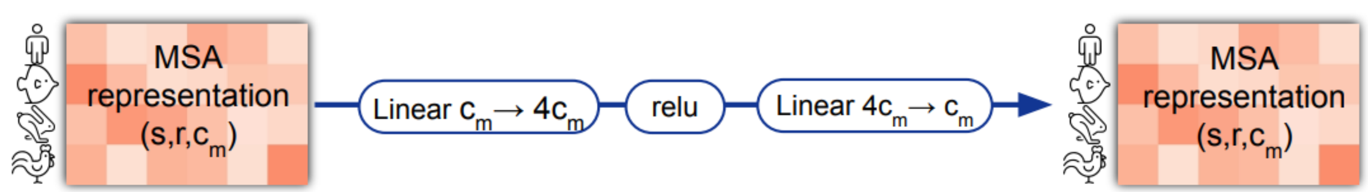
Jumper et al., 2021

AlphaFold v2.0 : model structure

Sequence model structure



Linear - ReLU- Linear (with LayerNorm)

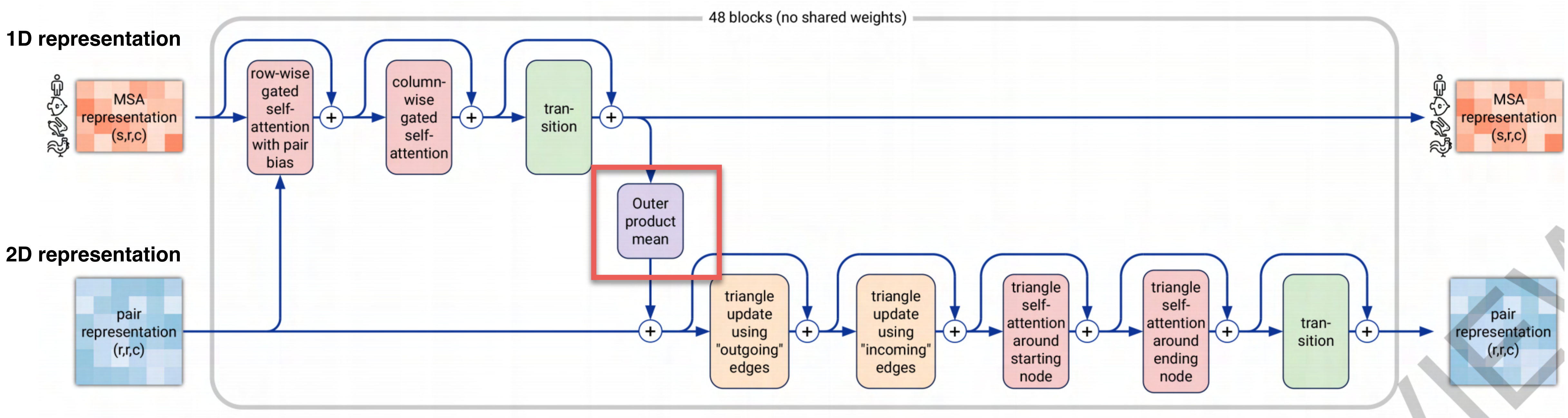


Supplementary Figure 4 | MSA transition layer. Dimensions: s: sequences, r: residues, c: channels.

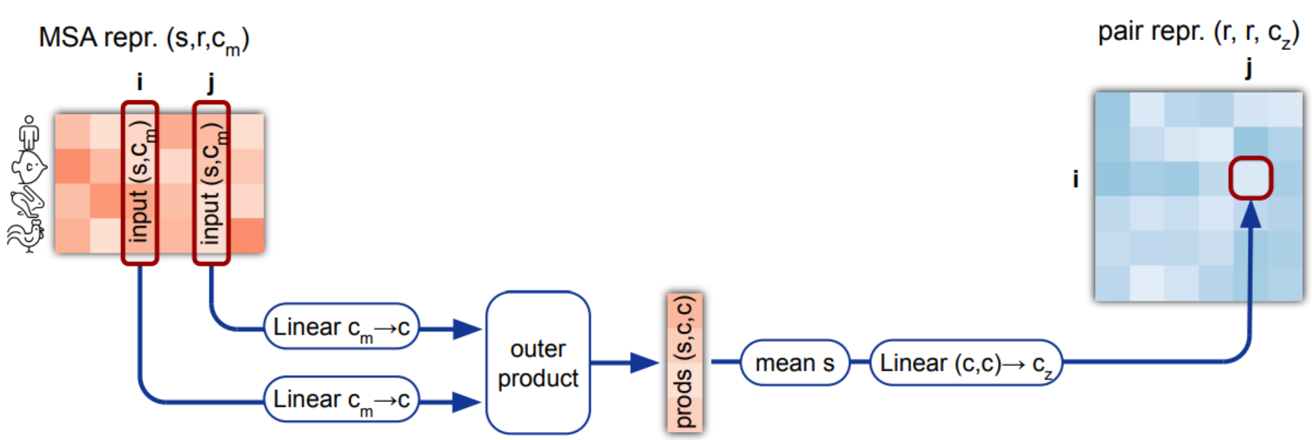
Jumper et al., 2021

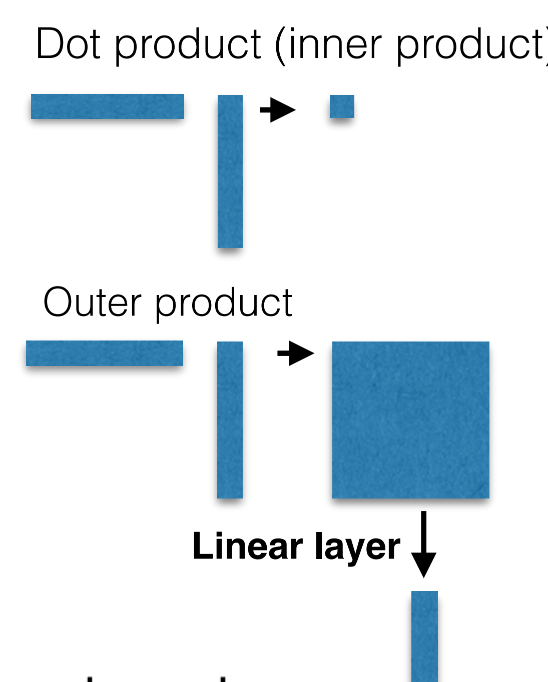
AlphaFold v2.0 : model structure

Sequence model structure



Outer product -> linear : more flexible than inner product



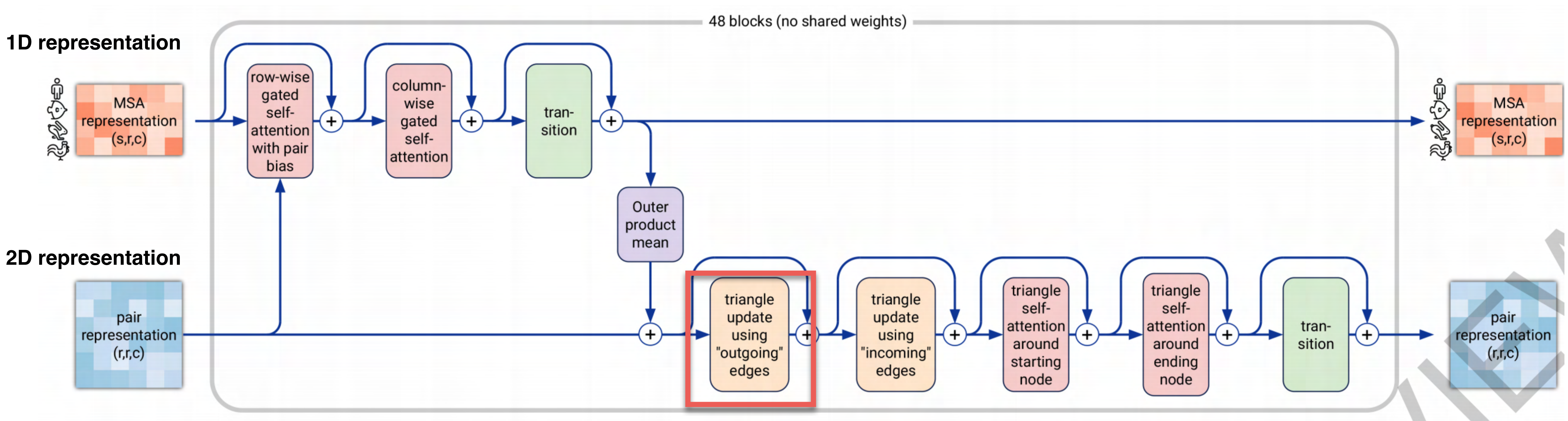


Supplementary Figure 5 | Outer product mean. Dimensions: s: sequences, r: residues, c: channels.

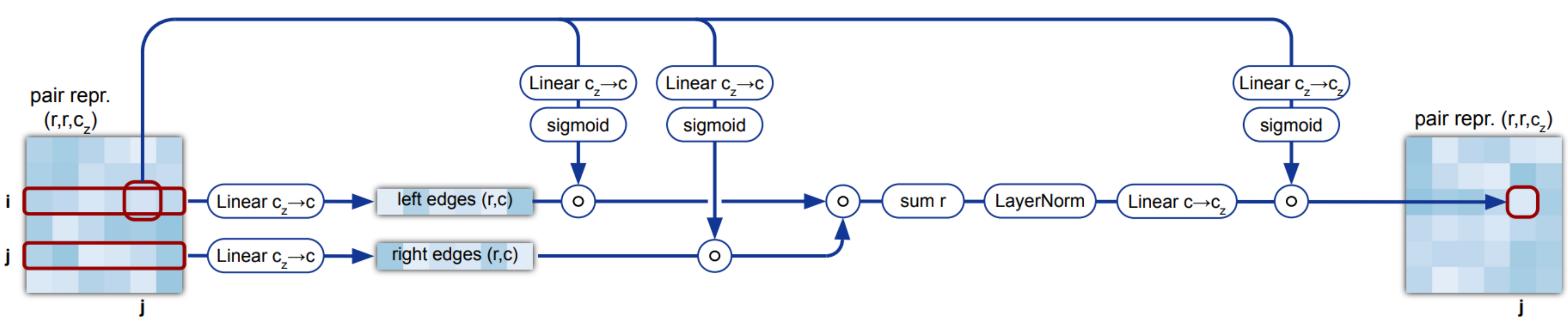
er et al., 2021

AlphaFold v2.0 : model structure

Sequence model structure



Similar to row-wise gated self attention

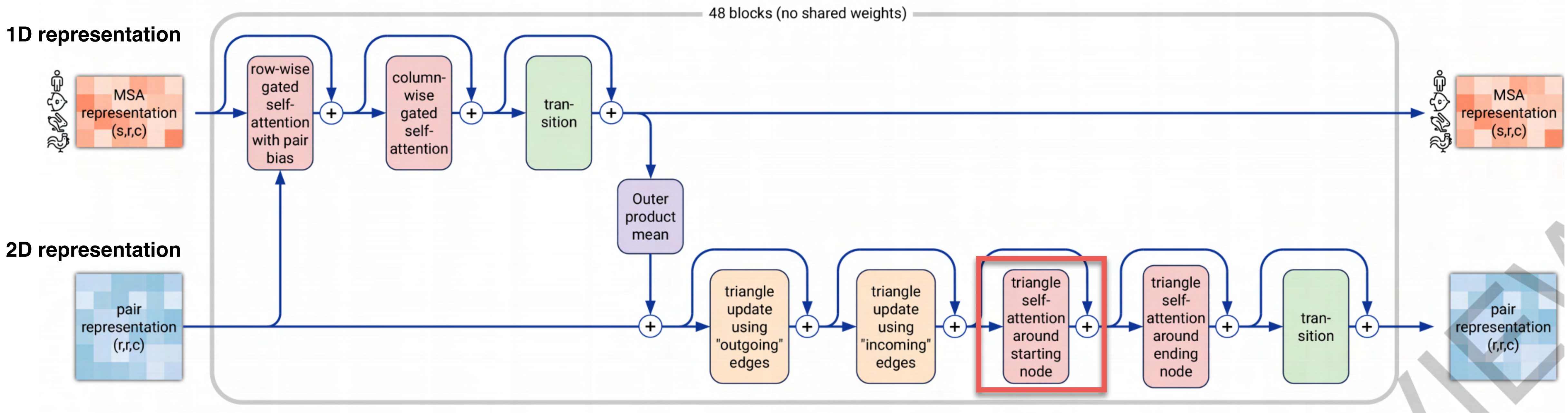


Supplementary Figure 6 | Triangular multiplicative update using "outgoing" edges. Dimensions: r: residues, c: channels.

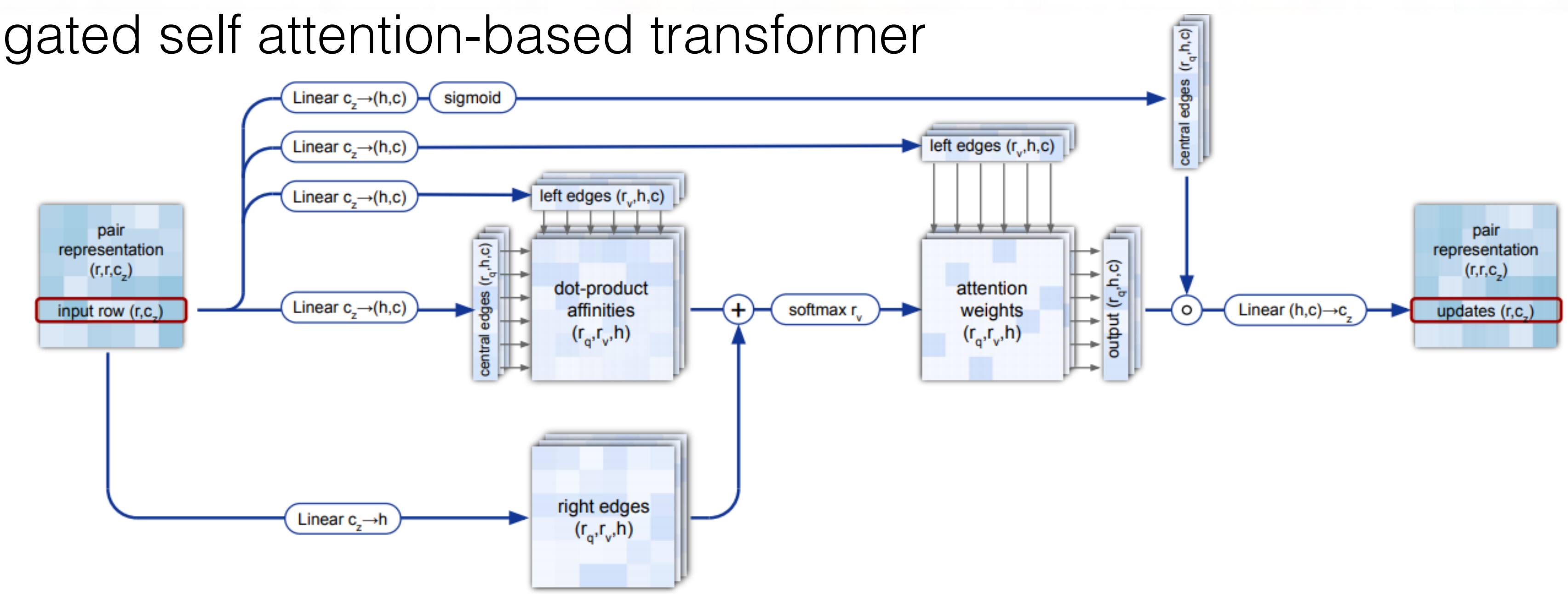
Jumper et al., 2021

AlphaFold v2.0 : model structure

Sequence model structure



Similar to row-wise gated self attention-based transformer

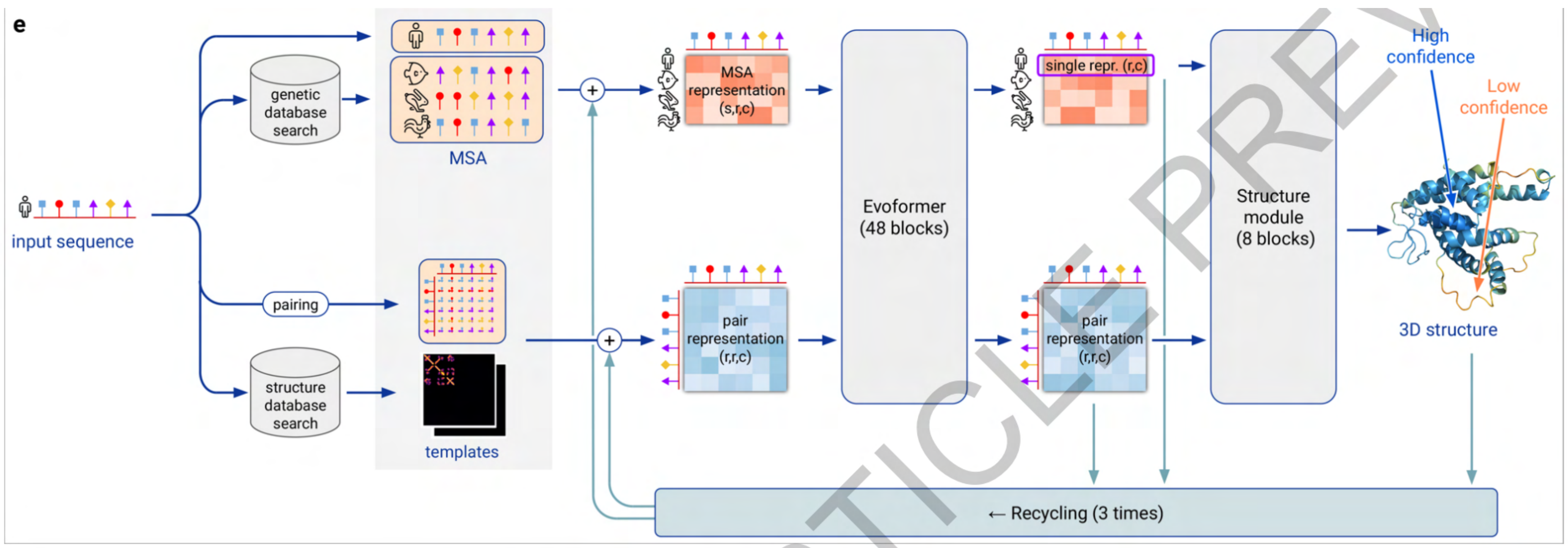


Supplementary Figure 7 | Triangular self-attention around starting node. Dimensions: r: residues, c: channels, h: heads

Jumper et al., 2021

AlphaFold v2.0 : Recycling mechanism

Learn to iteratively refine rather than jumping right at the results



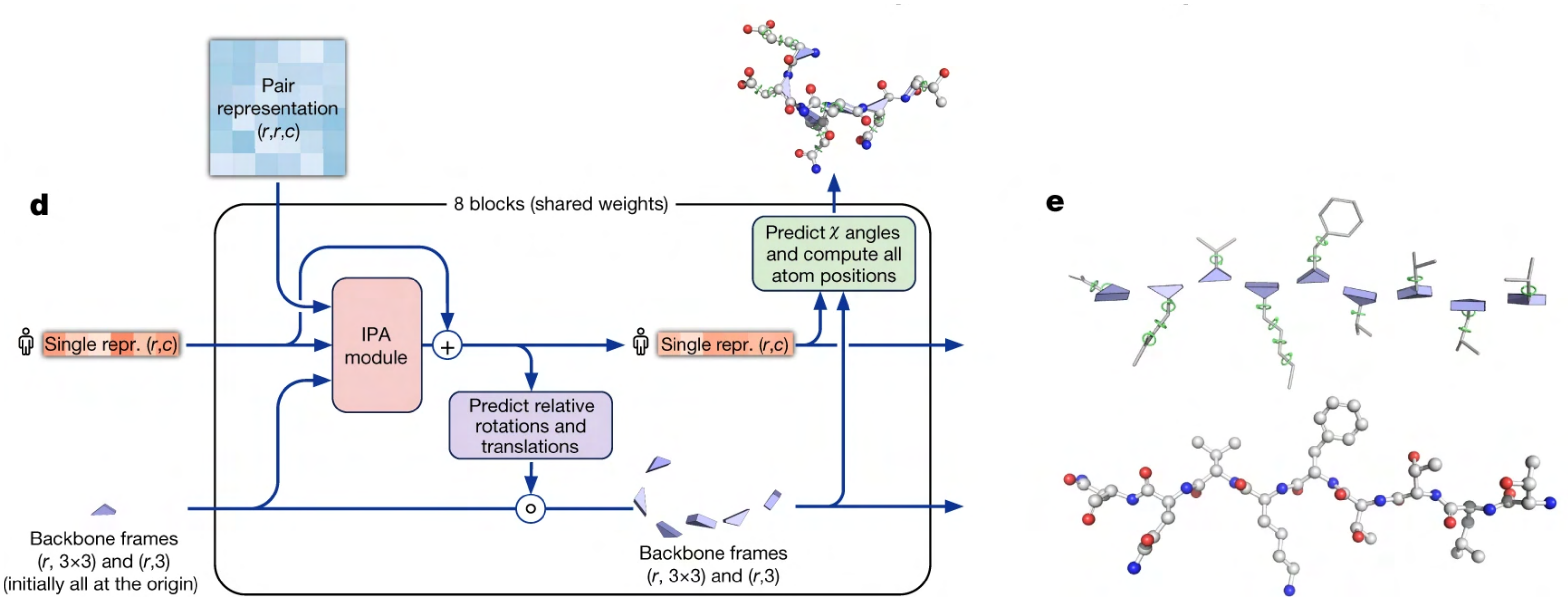
Simple approach: $X \to Y$

Recycled / recurrent prediction: $X + Y^* \to Y$

Jumper et al., 2021

AlphaFold v2.0 : Structure module

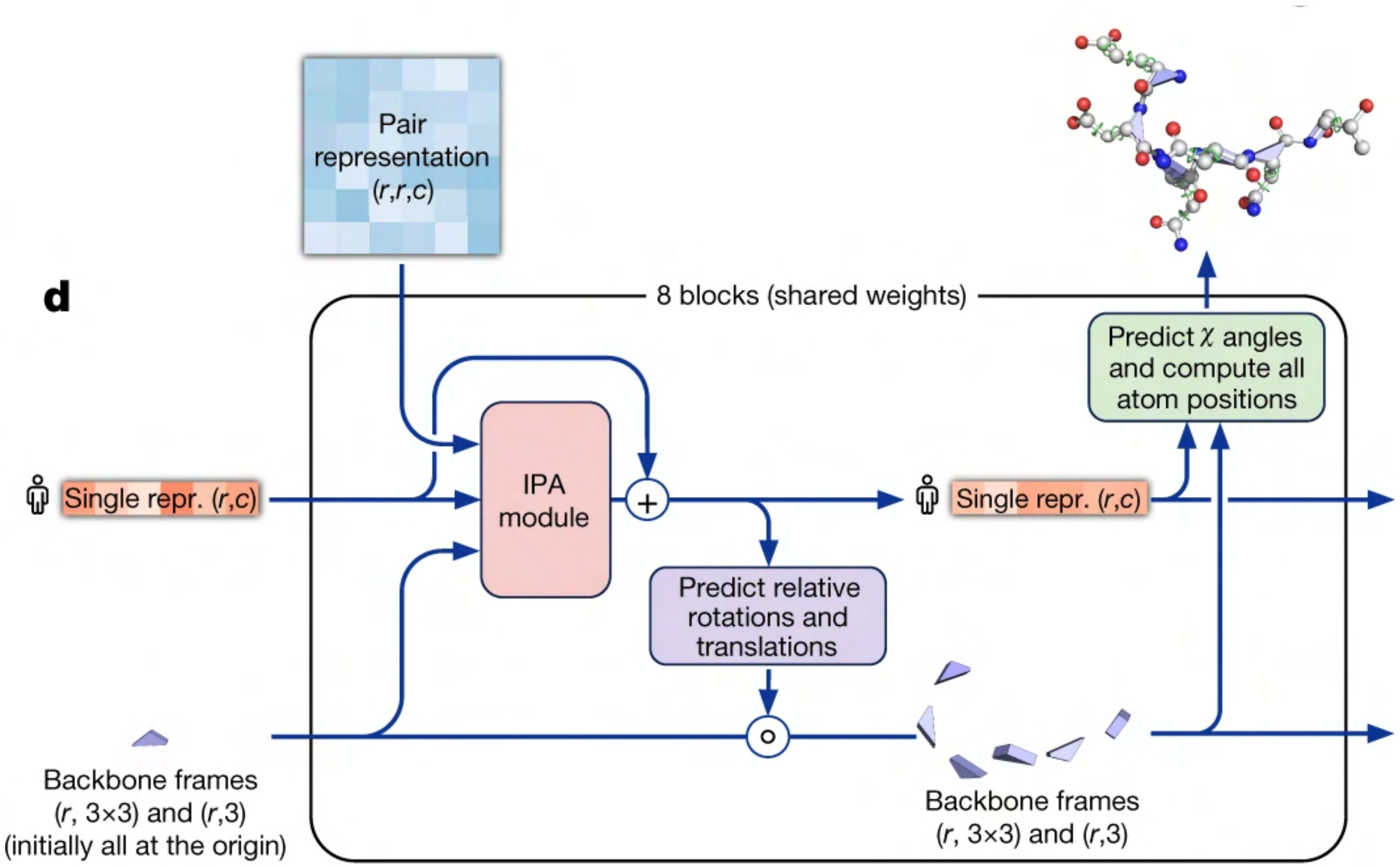
From intermediate representations to 3D coordinates



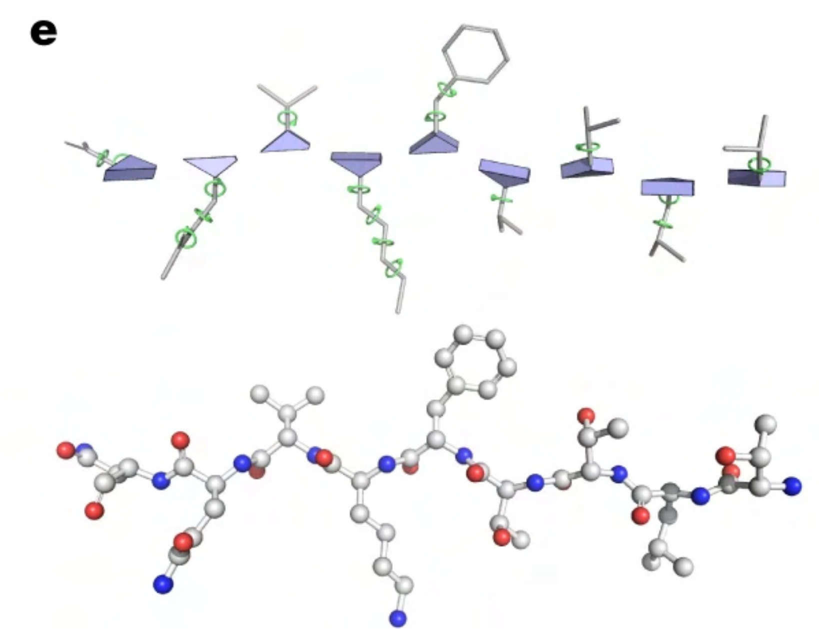
Jumper et al., 2021

AlphaFold v2.0 : Structure module

From intermediate representations to 3D coordinates



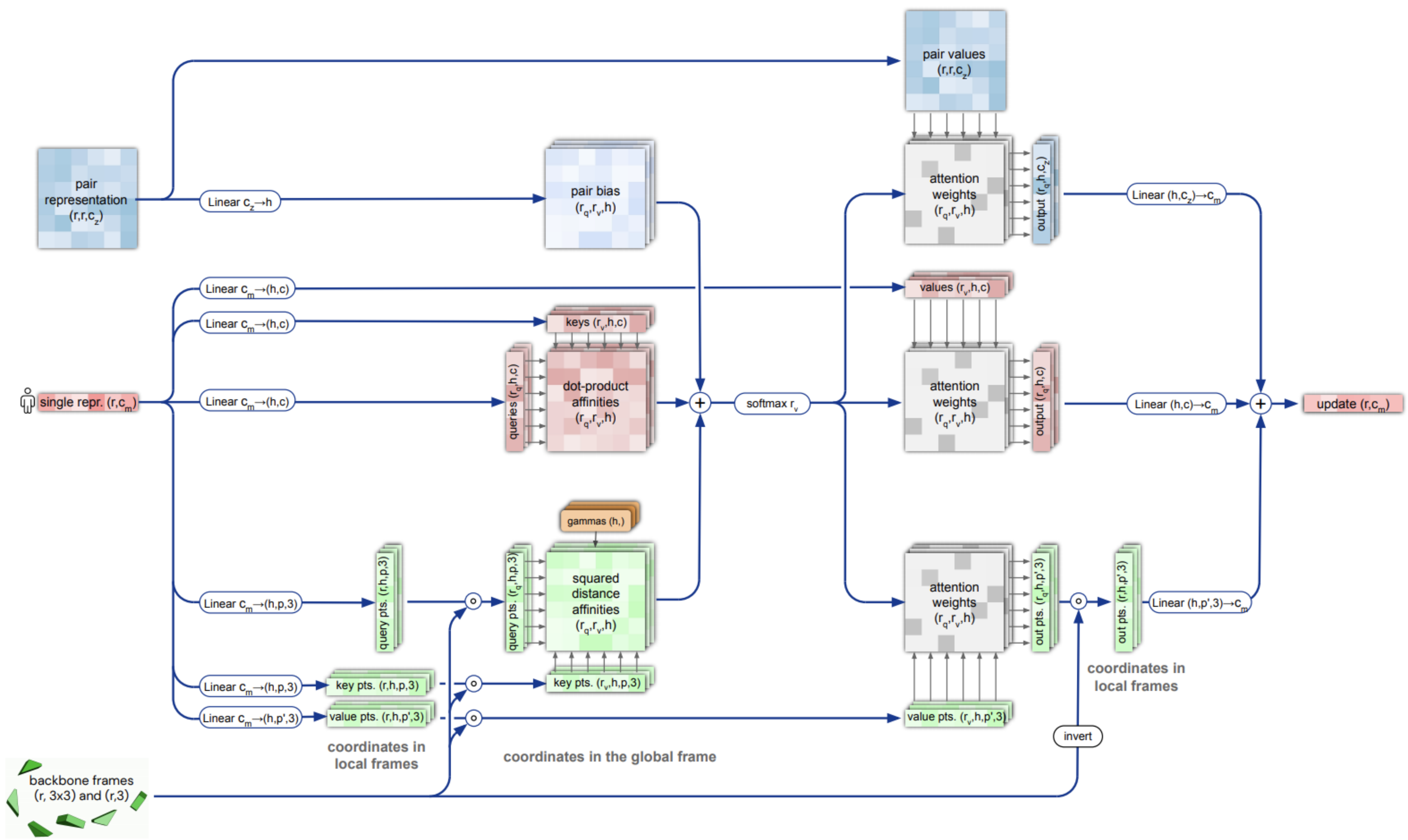
Side chain angles are computed by per-residue network



Jumper et al., 2021

AlphaFold v2.0 : Structure module

Invariant Point Attention module



equivariant to the rotation of backbone frames

Supplementary Figure 8 | Invariant Point Attention Module. (**top, blue arrays**) modulation by the pair representation. (**middle, red arrays**) standard attention on abstract features. (**bottom, green arrays**) Invariant point attention. Dimensions: r: residues, c: channels, h: heads, p: points.

Jumper et al., 2021