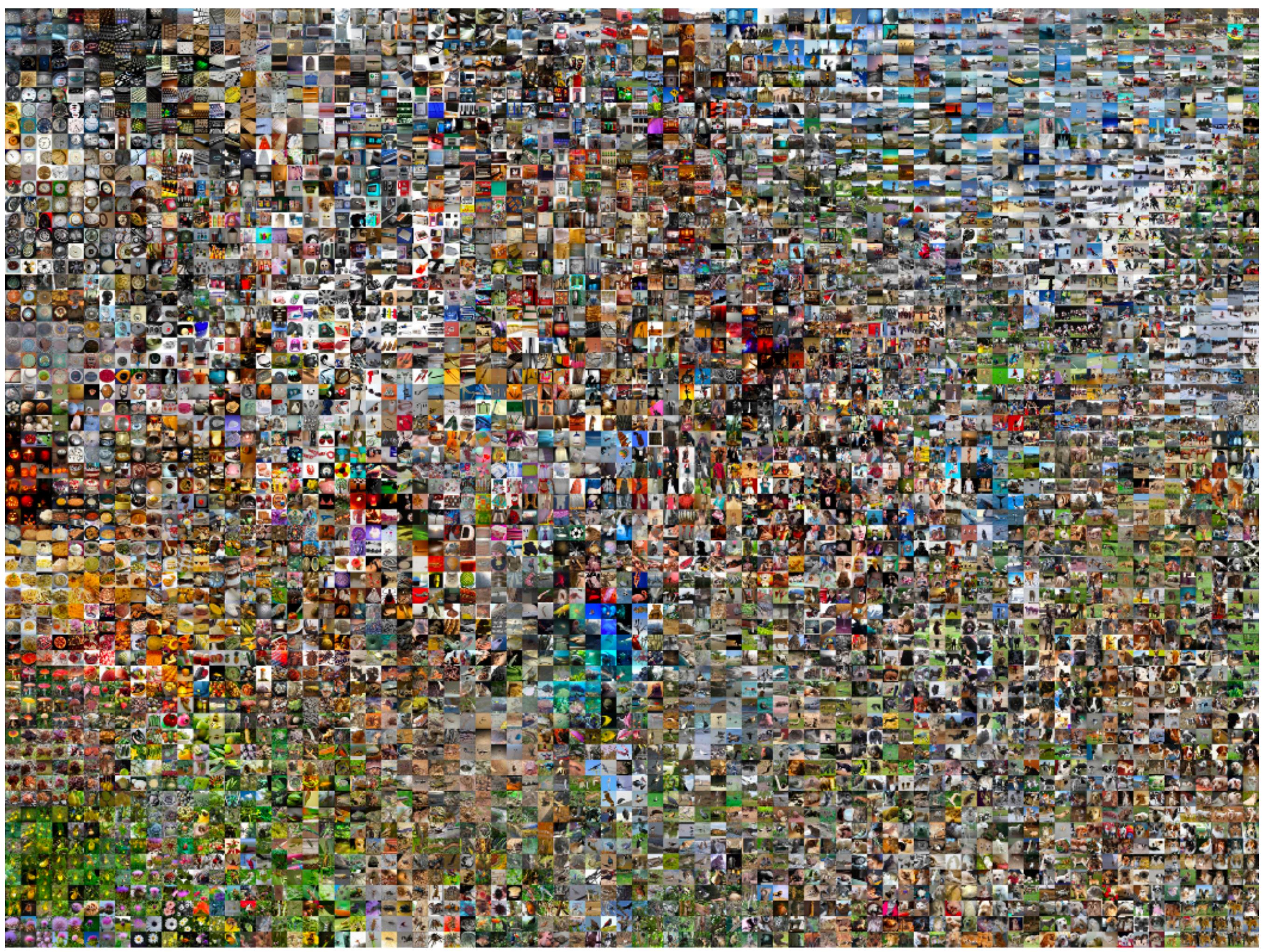
Simulation & Generative Models

Gregor Kasieczka
Email: gregor.kasieczka@uni-hamburg.de
Twitter/X: @GregorKasieczka
COFI Winter School 2023

CLUSTER OF EXCELLENCE
QUANTUM UNIVERSE
Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG
KISS
CDCS
CENTER FOR DATA AND COMPUTING
IN NATURAL SCIENCES
FSP CMS
PUNCH 4NFDI
DASHH
PIER
Partnership of
Universität Hamburg and DESY
GEFÖRDERT VOM
Bundesministerium
für Bildung
und Forschung
Emmy Noether-Programm
Deutsche Forschungsgemeinschaft
DFG

Motivation

Have: input examples (collision events, detector readouts, …)



Want: more data

Specifically: new data similar to the input, but not exact copies

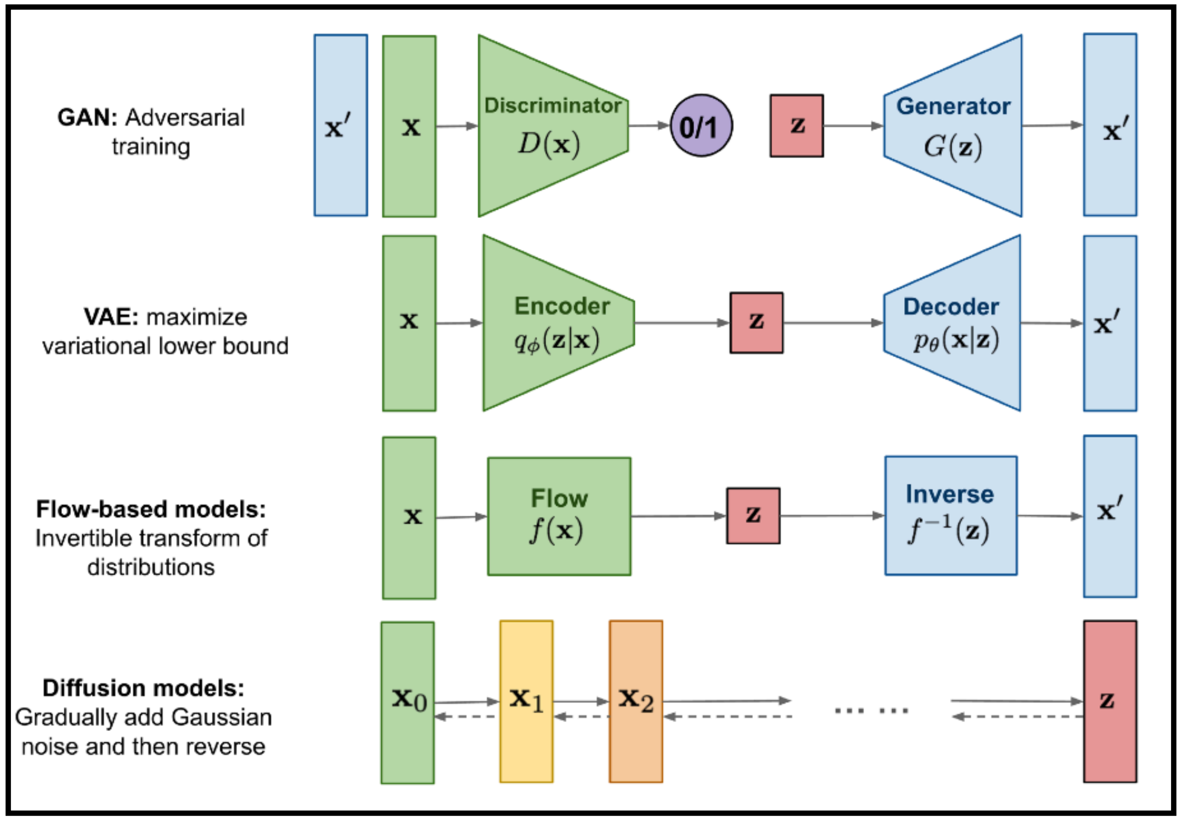
How to encode in neural net?

Uses:

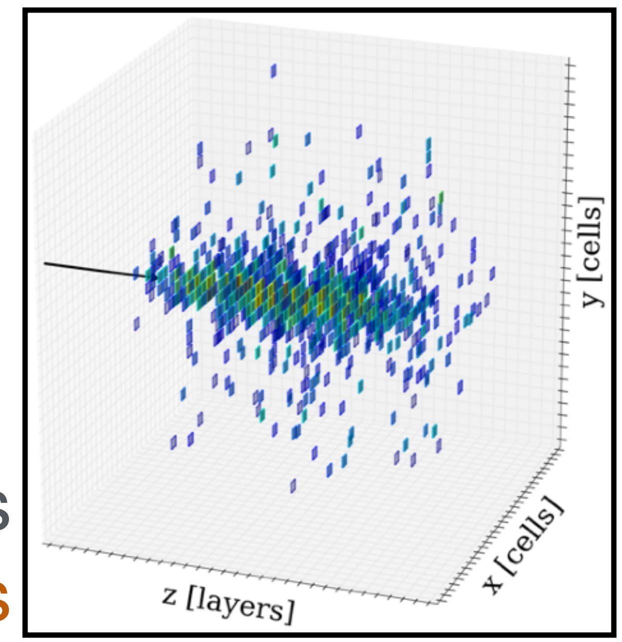
- Detector Simulation
- In-situ background estimation
- Surrogate models
- …

Overview

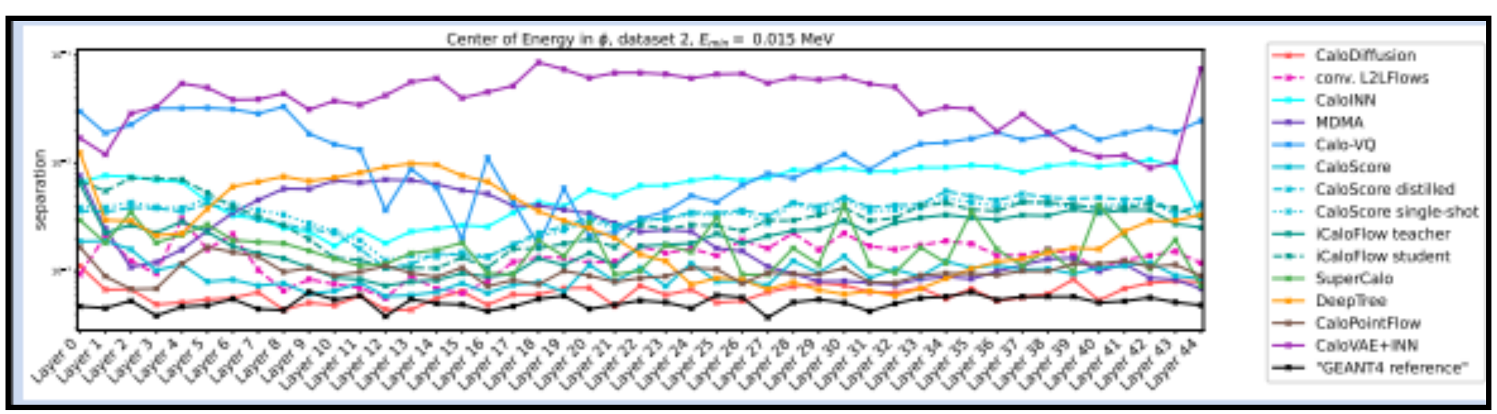
1. Common architectures*
   -> GANs, VAEs, NF today
   -> Diffusion & CNF tomorrow



2. Physics applications



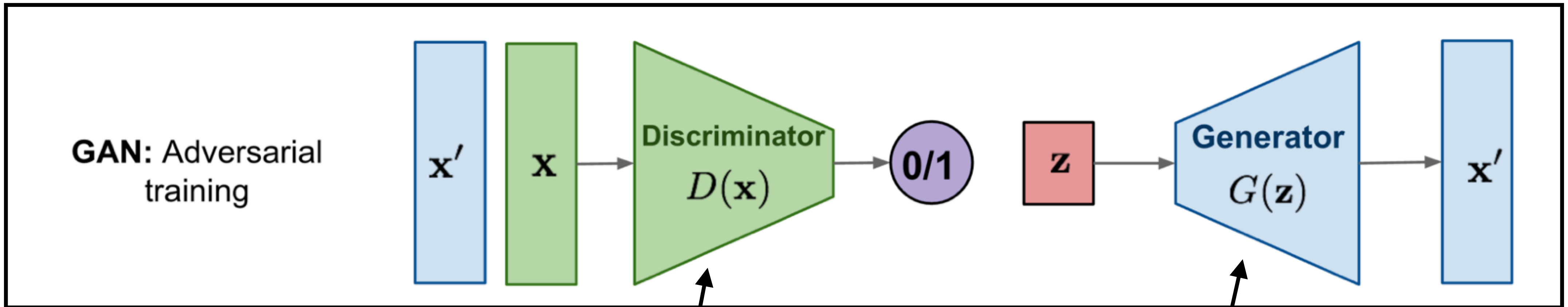
3. Quality metrics



*excluding transformers

Generative Adversarial Networks

Generative Adversarial Networks



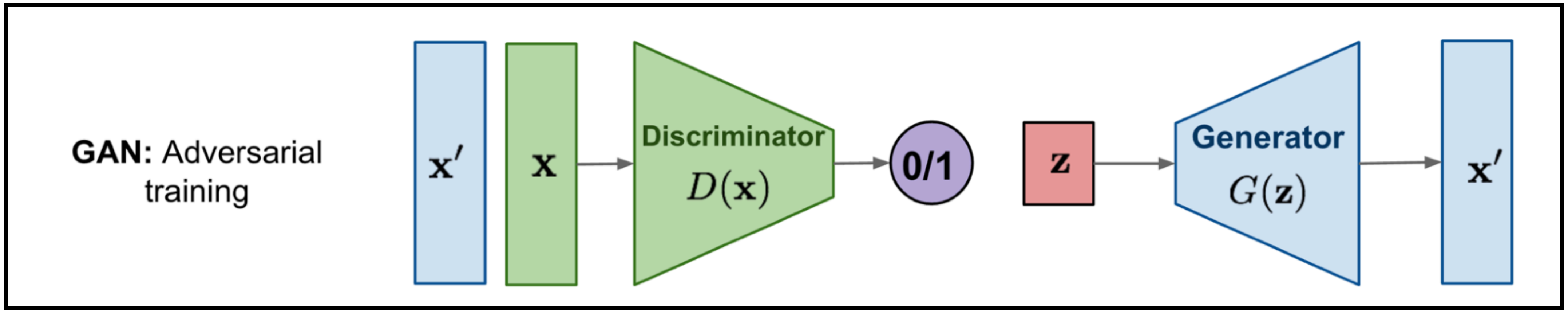
Generative Adversarial Networks (GANs) consist of 2 networks

Provides feedback on quality of examples

Maps random noise to realistic examples

1406.2661
lilianweng.github.io

Generative Adversarial Networks



Training objective:
Binary cross entropy

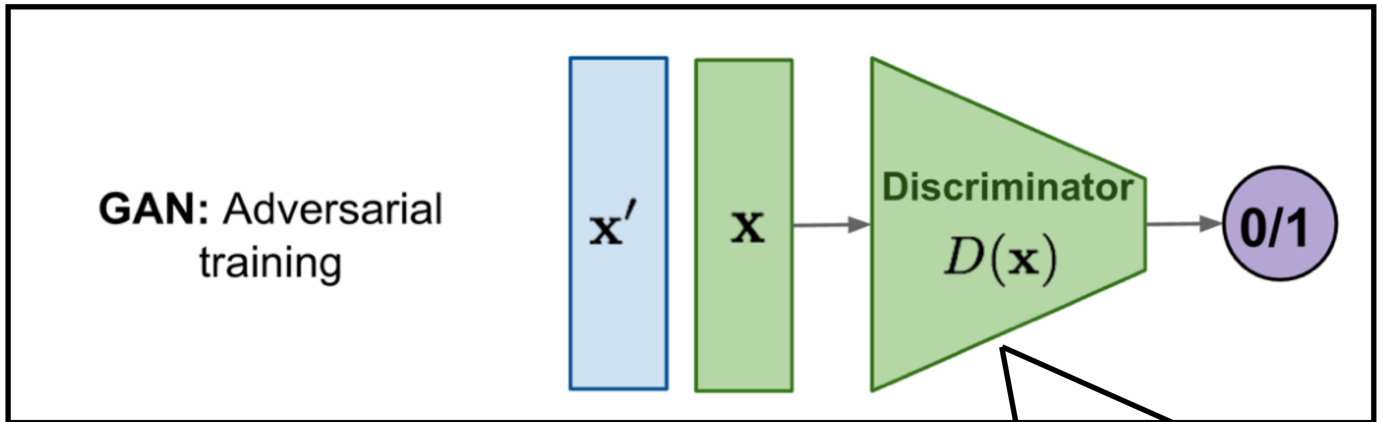
$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

True examples

Fake examples

1406.2661

Generative Adversarial Networks



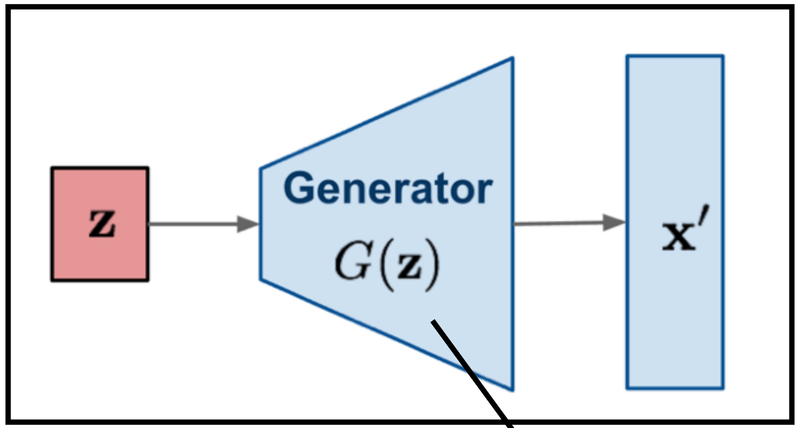
Training objective:
Binary cross entropy

Maximise for discriminator

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

1406.2661

Generative Adversarial Networks



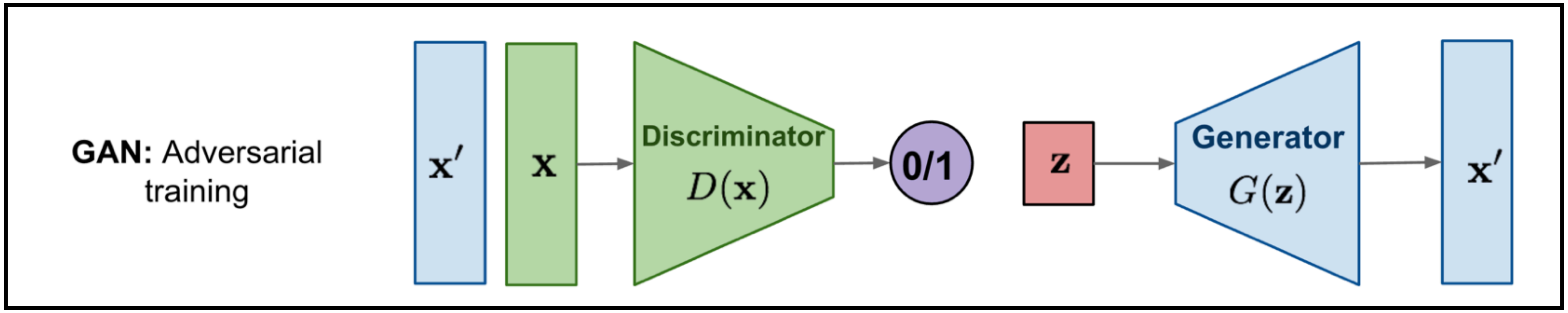
Training objective:
Binary cross entropy

Minimise for generator

$$\min_G \max_D V(D, G) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

1406.2661

Generative Adversarial Networks



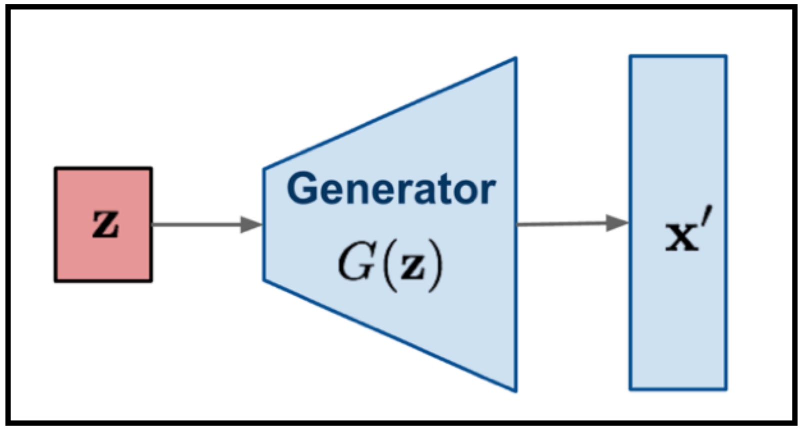
Training objective:
Binary cross entropy

$$\min_G \max_D V(D, G) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

At (Nash) equilibrium:
Generator produces realistic examples
Discriminator is maximally confused

1406.2661

Generative Adversarial Networks



Training objective:
Binary cross entropy

$$\min_G \max_D V(D, G) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

For generation:
Sample from Generator
Discard Discriminator

1406.2661

Comments on GANs

Architecture:

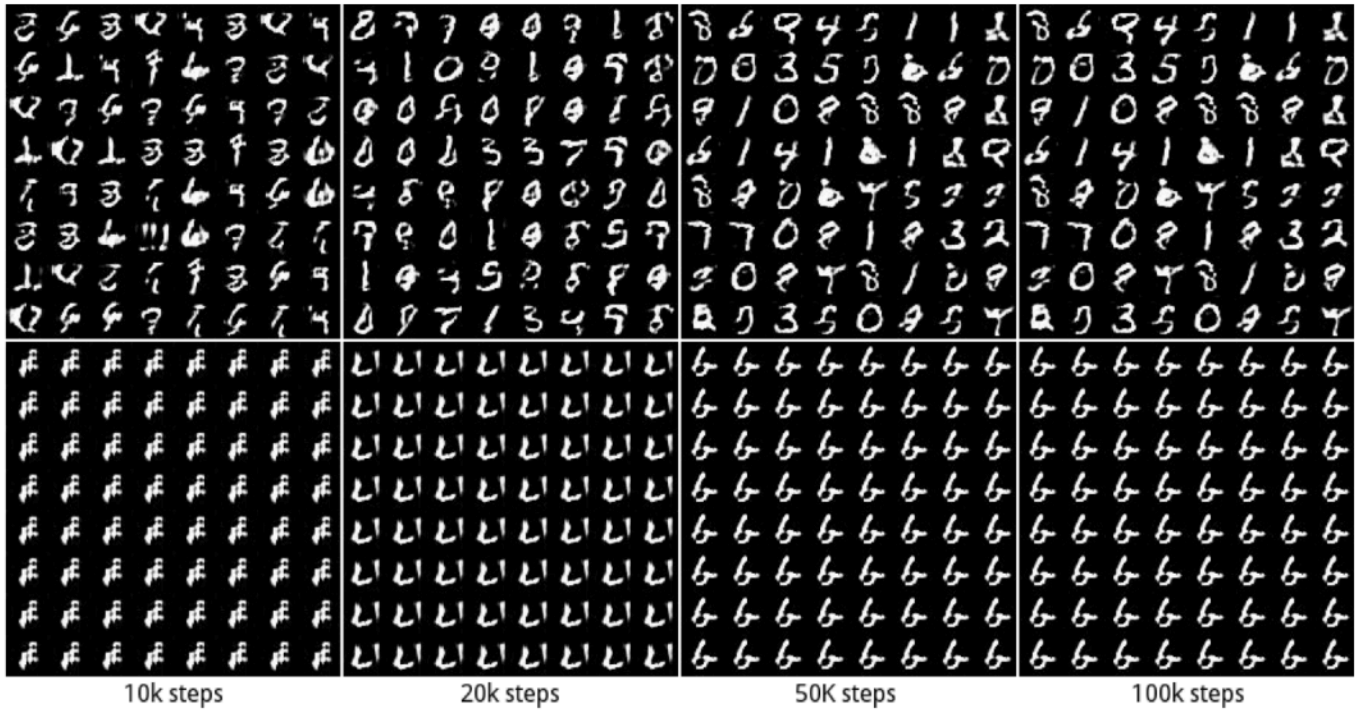
- Low complexity, fast and adaptable

Learning:

- Unstable training
- Matching of generator/discriminator (vanishing gradients)
- Mode collapse
- Loss function not interpretable

Maturity:

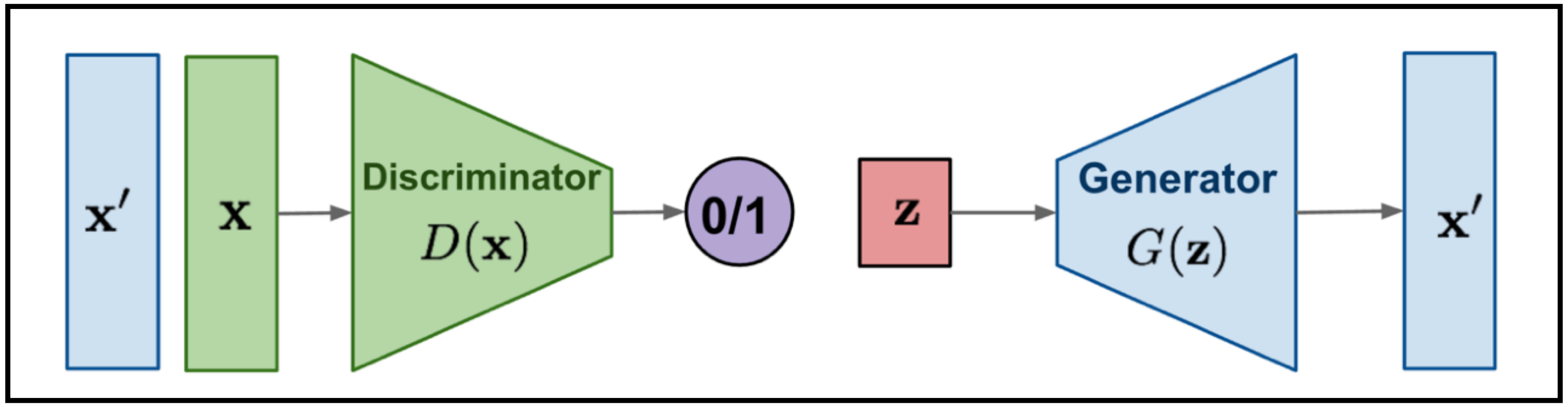
- Well established, many variants and extensions



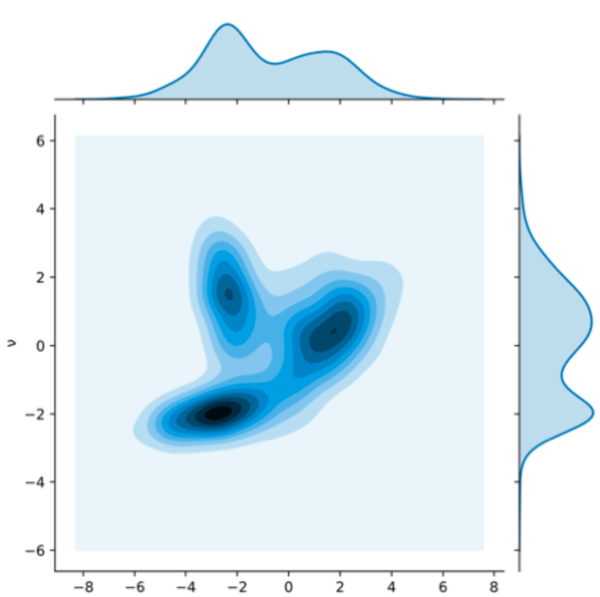
Mode collapse

1611.02163

Wasserstein GAN



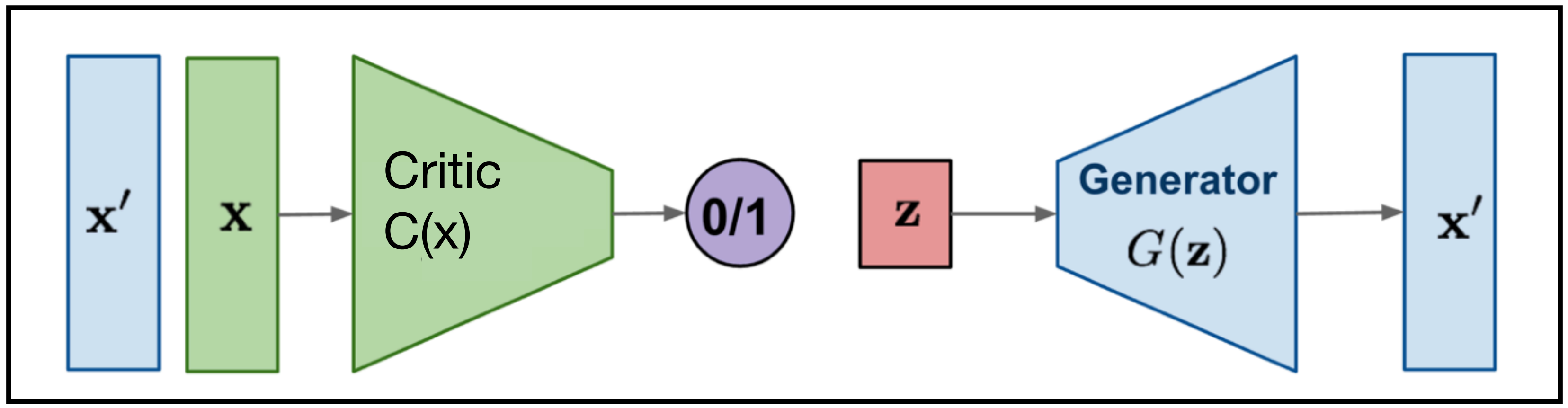
- Standard GANs minimise Jensen-Shannon divergence of generator output and true data
    - Not best measure, e.g. for non-overlapping distributions
- Replace with Wasserstein / Earth-Mover-Distance



$$W_p(\mu, \nu) = \left(\inf_{\gamma \in \Gamma(\mu,\nu)} \mathbf{E}_{(x,y)\sim\gamma} d(x, y)^p \right)^{1/p}$$

1704.00028

Wasserstein GAN



GAN loss:

$$\min_G \max_D \mathop{\mathbb{E}}_{\boldsymbol{x}\sim\mathbb{P}_r} [\log(D(\boldsymbol{x}))] + \mathop{\mathbb{E}}_{\tilde{\boldsymbol{x}}\sim\mathbb{P}_g} [\log(1 - D(\tilde{\boldsymbol{x}}))]$$

Wasserstein GAN loss*:

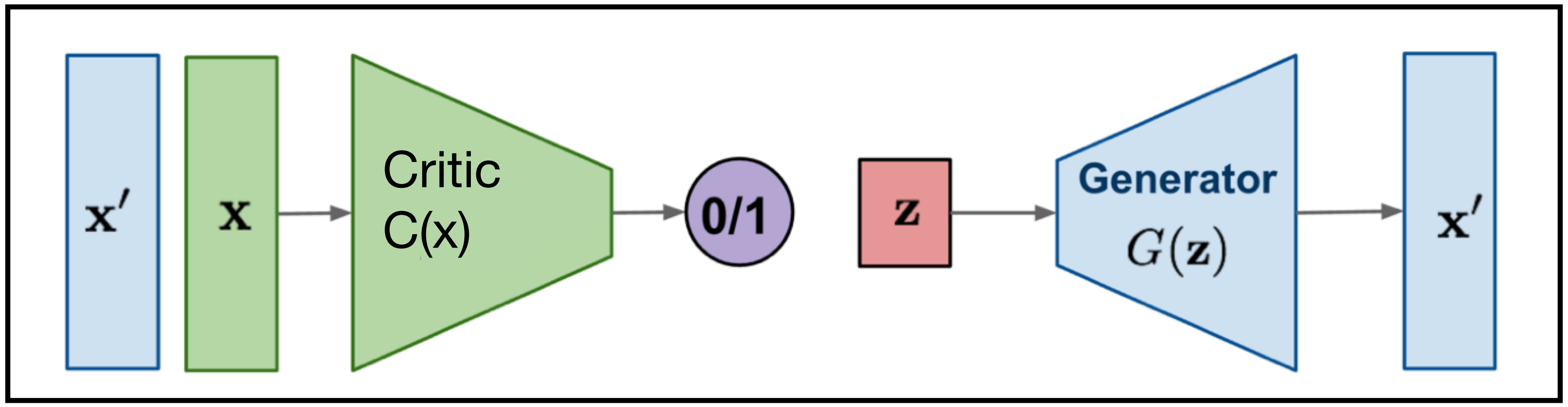
$$\min_G \max_{D\in\mathcal{D}} \mathop{\mathbb{E}}_{\boldsymbol{x}\sim\mathbb{P}_r} \big[D(\boldsymbol{x})\big] - \mathop{\mathbb{E}}_{\tilde{\boldsymbol{x}}\sim\mathbb{P}_g} \big[D(\tilde{\boldsymbol{x}}))\big]$$

Requires bounded Lipschitz norm, e.g. via term in loss

* Some mathematics involved from earth mover distance to here

1704.00028

Wasserstein GAN



GAN loss:

$$\min_G \max_D \mathop{\mathbb{E}}_{\boldsymbol{x}\sim\mathbb{P}_r}[\log(D(\boldsymbol{x}))] + \mathop{\mathbb{E}}_{\tilde{\boldsymbol{x}}\sim\mathbb{P}_g}[\log(1 - D(\tilde{\boldsymbol{x}}))]$$

Wasserstein GAN loss:

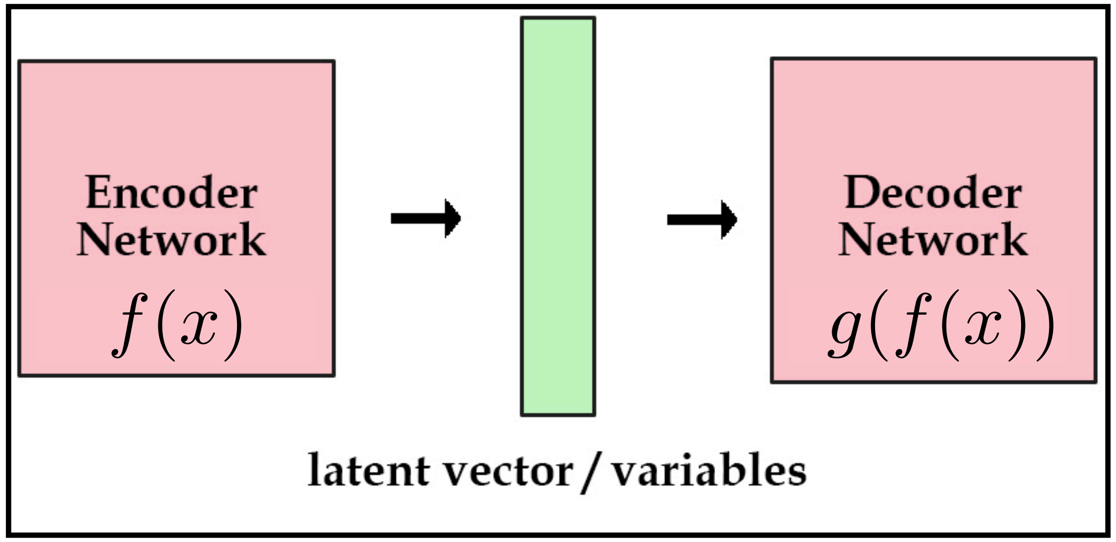
$$\min_G \max_{D\in\mathcal{D}} \mathop{\mathbb{E}}_{\boldsymbol{x}\sim\mathbb{P}_r}\big[D(\boldsymbol{x})\big] - \mathop{\mathbb{E}}_{\tilde{\boldsymbol{x}}\sim\mathbb{P}_g}\big[D(\tilde{\boldsymbol{x}}))\big]$$

Improves training stability and sample quality (e.g. mode collapse)

1704.00028

Variational Autoencoders

Autoencoder



Two networks

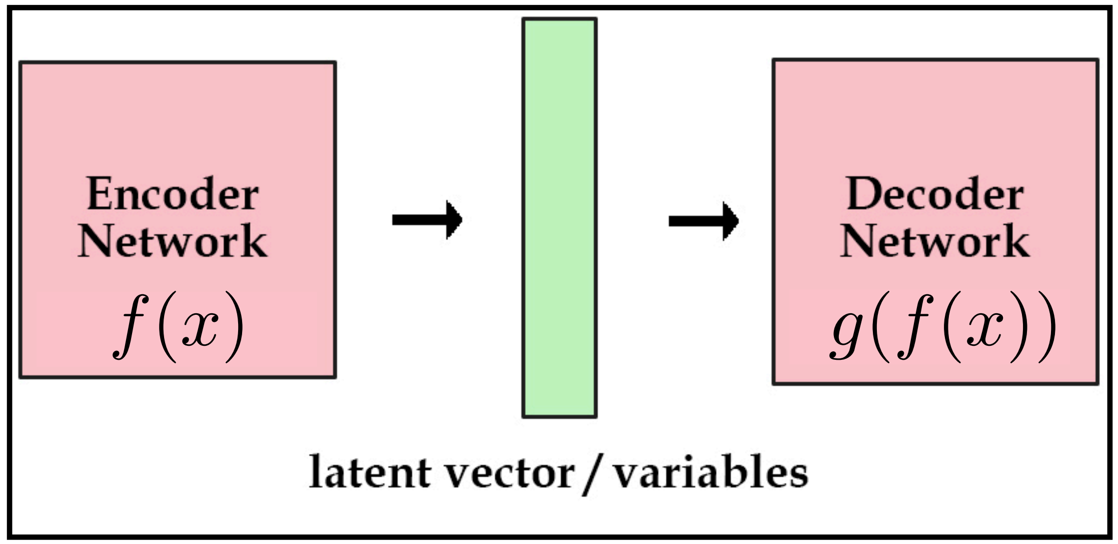
Encoder: data → latent space

Decoder: latent space → data

1312.6114

lilianweng.github.io

Autoencoder



Two networks

Encoder: data → latent space

Decoder: latent space → data

Training objective:

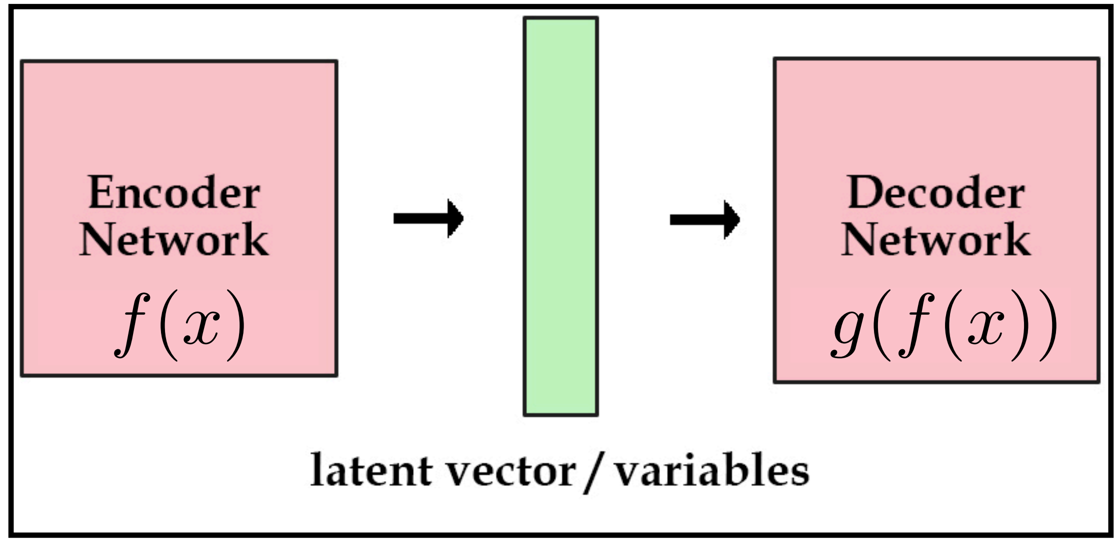
Minimise input/output difference

Decoder Encoder

$$L = (x - f(g(x)))^2$$

1312.6114

Autoencoder



Two networks
Encoder: data → latent space
Decoder: latent space → data

Decoder Encoder

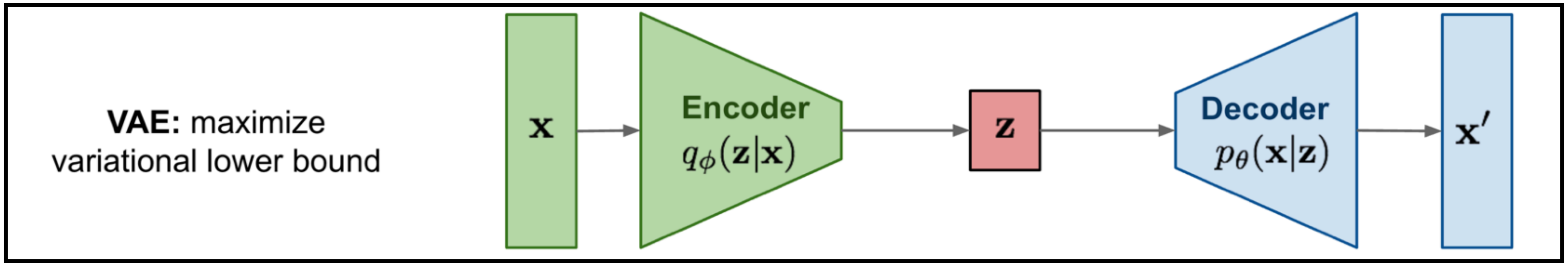
$$L = (x - f(g(x)))^2$$

Training objective:
Minimise input/output difference

Uses:
Dimension reduction
Denoising
Anomaly detection
Generation?

1312.6114

Variational Autoencoder

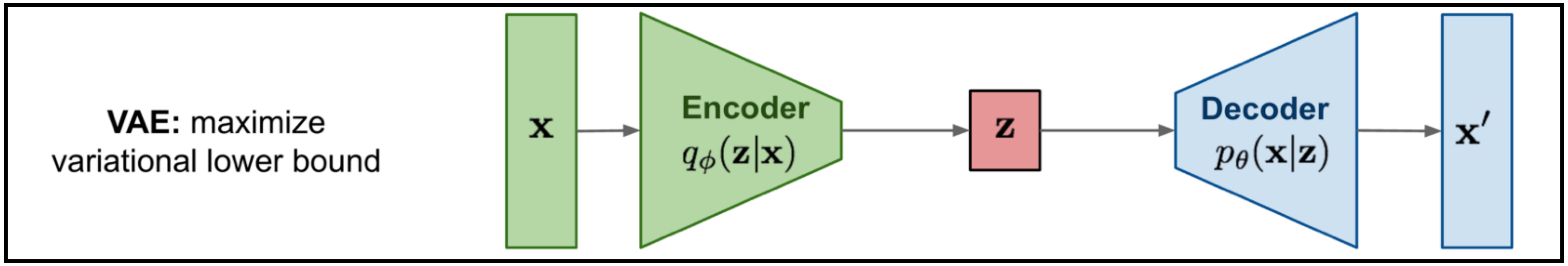


$$f(x) = (\mu, \sigma)$$

Variational Autoencoder (VAE):
Split latent space

1312.6114

Variational Autoencoder



Variational Autoencoder (VAE):
Split latent space
Sample before decoder

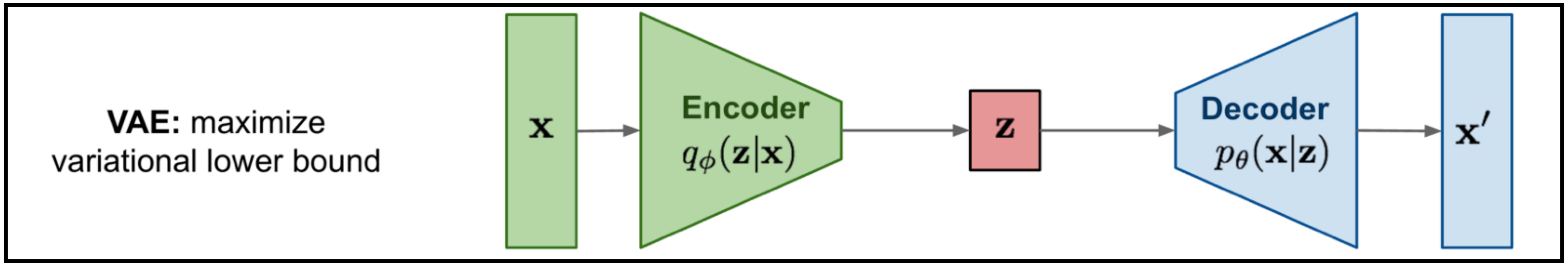
$$f(x) = (\mu, \sigma)$$

$$z = \mathrm{Gaussian}(\mu, \sigma)$$

$$x' = g(z)$$

1312.6114

Variational Autoencoder



Variational Autoencoder (VAE):
Split latent space
Sample before decoder
Penalty so mean/std are close to unit Gaussian

$$f(x) = (\mu, \sigma)$$

$$z = \text{Gaussian}(\mu, \sigma)$$

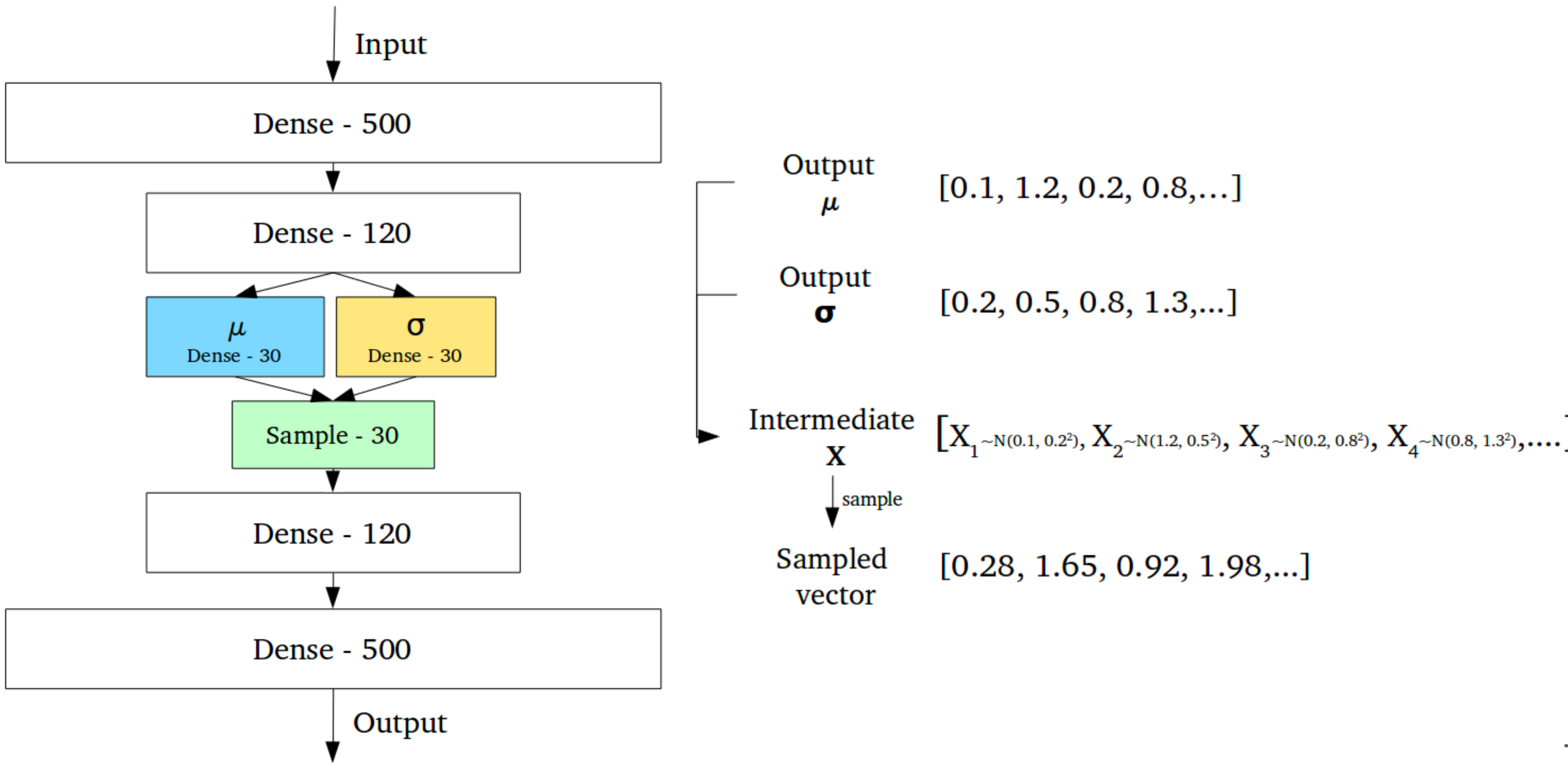
$$x' = g(z)$$

$$L = (x - g(z))^2 + \sigma^2 + \mu^2 - \log(\sigma) - 1$$

(Calculate KL-divergence
between Gaussians)

1312.6114

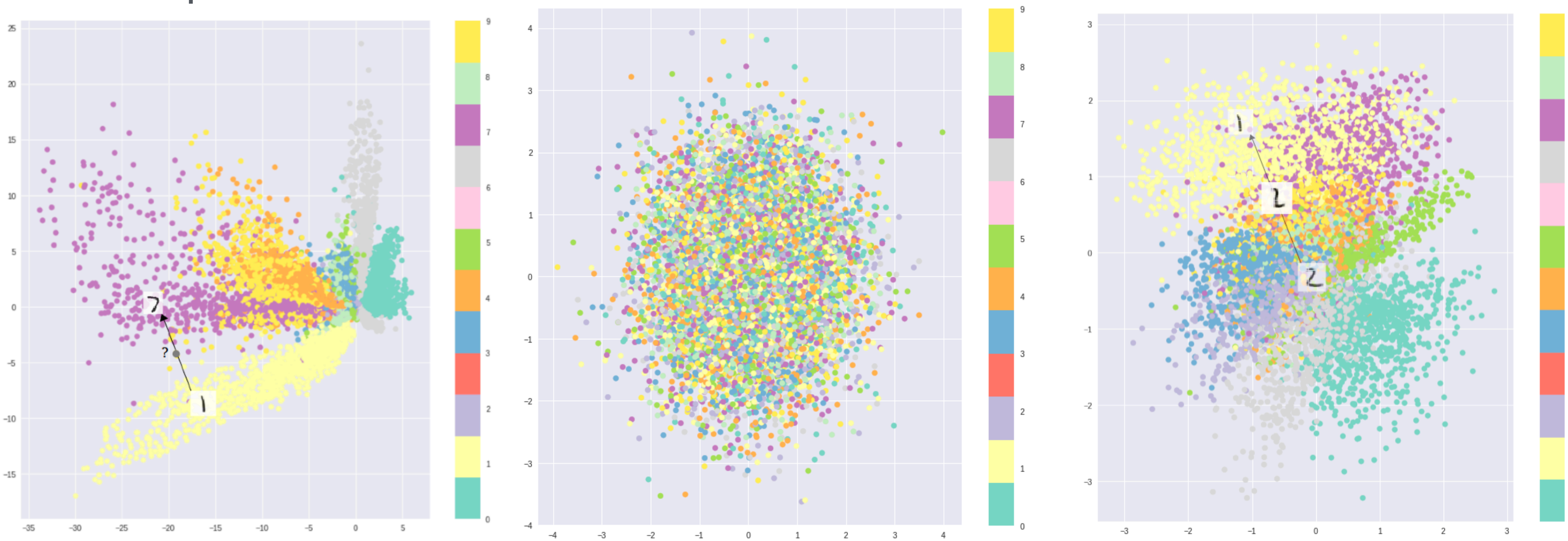
VAE Example



Input
Dense - 500
Dense - 120
μ
Dense - 30
σ
Dense - 30
Sample - 30
Dense - 120
Dense - 500
Output
Output
μ
[0.1, 1.2, 0.2, 0.8,...]
Output
σ
[0.2, 0.5, 0.8, 1.3,...]
Intermediate
X
$[X_{1\sim N(0.1,\,0.2^2)}, X_{2\sim N(1.2,\,0.5^2)}, X_{3\sim N(0.2,\,0.8^2)}, X_{4\sim N(0.8,\,1.3^2)}, ...]$
sample
Sampled
vector
[0.28, 1.65, 0.92, 1.98,...]

Loss terms

Latent space of MNIST VAE



$$(x - g(z))^2 \qquad \sigma^2 + \mu^2 - \log(\sigma) - 1$$

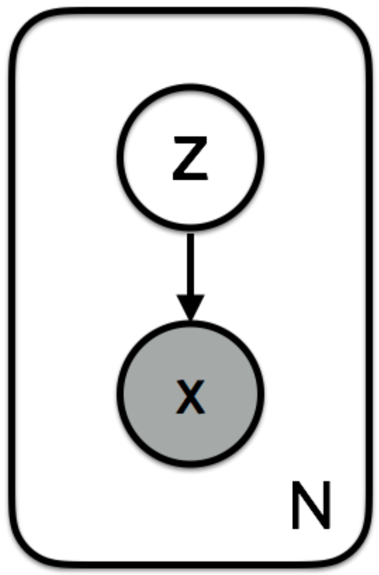
Reconstruction — Regularisation — Both terms

Loss terms

$$L = (x - g(z))^2 + \sigma^2 + \mu^2 - \log(\sigma) - 1$$

How did we get here?

Loss terms



Sample from latent variables z

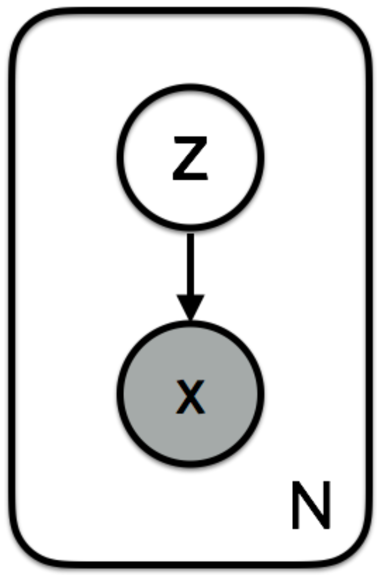
$$z_i \sim p(z)$$

Produce data points x

$$x_i \sim p(x \mid z)$$

https://jaan.io/what-is-variational-autoencoder-vae-tutorial/

Loss terms



Sample from latent variables z

$$z_i \sim p(z)$$

Produce data points x

$$x_i \sim p(x \mid z)$$

To choose correct latent distribution given data, could use Bayes theorem:

Conditional Prior

$$p(z \mid x) = \frac{p(x \mid z) p(z)}{p(x)}$$

Evidence

Difficult due to p(x)

Loss terms

To choose correct latent distribution given data, could use Bayes theorem:

$$p(z \mid x) = \frac{p(x \mid z)p(z)}{p(x)}$$

Instead, approximate with family of posterior distributions (variational inference):

$$\mathbb{KL}(q_\lambda(z \mid x) \mid\mid p(z \mid x)) =$$

$$\mathbf{E}_q[\log q_\lambda(z \mid x)] - \mathbf{E}_q[\log p(x, z)] + \log p(x)$$

And find optimal approximation:

$$q_\lambda^*(z \mid x) = \arg\min_\lambda \mathbb{KL}(q_\lambda(z \mid x) \mid\mid p(z \mid x))$$

Still difficult due to (hidden) p(x) term!

Loss terms

$$\mathbb{KL}(q(x)\,||\,p(x)) = -\int \mathrm{d}x\, q(x) \log \frac{p(x)}{q(x)}$$

Kullback-Leibler definition

To choose correct latent distribution given data, could use Bayes theorem:

$$p(z \mid x) = \frac{p(x \mid z)p(z)}{p(x)}$$

$$p(x, z) = p(z|x)p(x)$$

Reminder

Instead, approximate with family of posterior distributions (variational inference):

$$\mathbb{KL}(q_\lambda(z \mid x) \mid\mid p(z \mid x)) =$$
$$\mathbf{E}_q[\log q_\lambda(z \mid x)] - \mathbf{E}_q[\log p(x, z)] + \log p(x)$$

And find optimal approximation:

$$q_\lambda^*(z \mid x) = \arg\min_\lambda \mathbb{KL}(q_\lambda(z \mid x) \mid\mid p(z \mid x))$$

Still difficult due to p(x) term!

Loss terms

$$\mathbb{KL}(q_\lambda(z \mid x) \mid\mid p(z \mid x)) =$$

$$\mathbf{E}_q[\log q_\lambda(z \mid x)] - \mathbf{E}_q[\log p(x, z)] + \log p(x)$$

Introduce

$$ELBO(\lambda) = \mathbf{E}_q[\log p(x, z)] - \mathbf{E}_q[\log q_\lambda(z \mid x)]$$

Rewrite

$$\log p(x) = ELBO(\lambda) + \mathbb{KL}(q_\lambda(z \mid x) \mid\mid p(z \mid x))$$

As KL is >=0, ELBO is a lower limit for p(X)
ELBO: Evidence Lower Bound

Loss terms

Maximise

$$ELBO(\lambda) = \mathbf{E}_q[\log p(x, z)] - \mathbf{E}_q[\log q_\lambda(z \mid x)]$$

Rewrite for samples, using neural networks:

$$ELBO_i(\theta, \phi) = \mathbb{E}q_\theta(z \mid x_i)[\log p_\phi(x_i \mid z)] - \mathbb{KL}(q_\theta(z \mid x_i) \mid\mid p(z))$$

Reconstruction term

Regularisation term

Assume normal distribution

Difference between normal and standard normal

$$L = (x - g(z))^2 + \sigma^2 + \mu^2 - \log(\sigma) - 1$$

Loss terms

Maximise

$$ELBO(\lambda) = \mathbf{E}_q[\log p(x, z)] - \mathbf{E}_q[\log q_\lambda(z \mid x)]$$

Rewrite for samples, using neural networks:

$$ELBO_i(\theta, \phi) = \mathbb{E}q_\theta(z \mid x_i)[\log p_\phi(x_i \mid z)] - \mathbb{KL}(q_\theta(z \mid x_i) \mid\mid p(z))$$

Reconstruction term

Regularisation term

Assume normal distribution

Difference between normal and standard normal

$$L = (x - g(z))^2 + \sigma^2 + \mu^2 - \log(\sigma) - 1$$

Comments on VAEs

Architecture:

- Low complexity, fast and adaptable
- Target: Maximise lower bound on likelihood

Learning:

- Stable training
- Average prediction → blurrier output
- Interpretable latent space

Maturity:

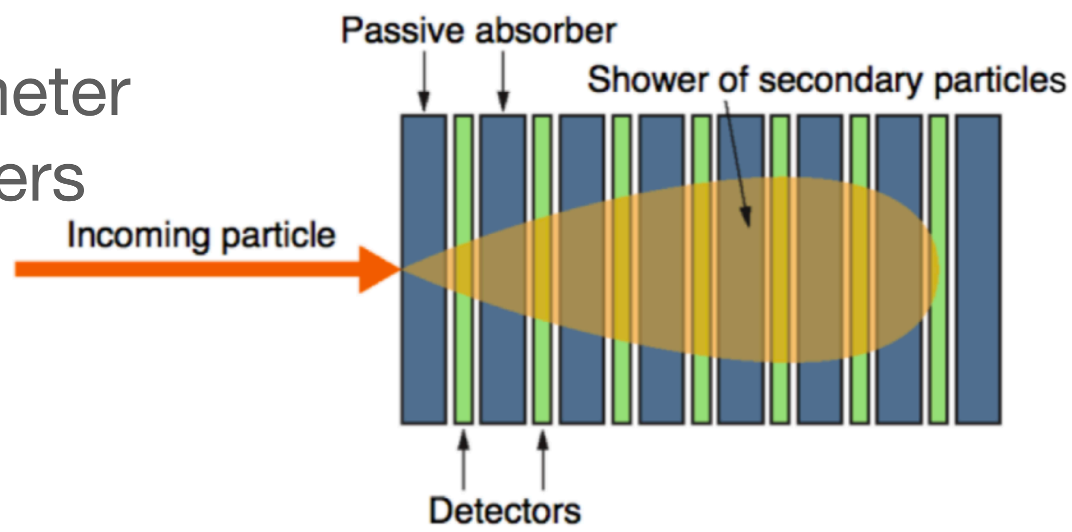
- Well established,
  many variants and extensions

1607.07539

Applications I

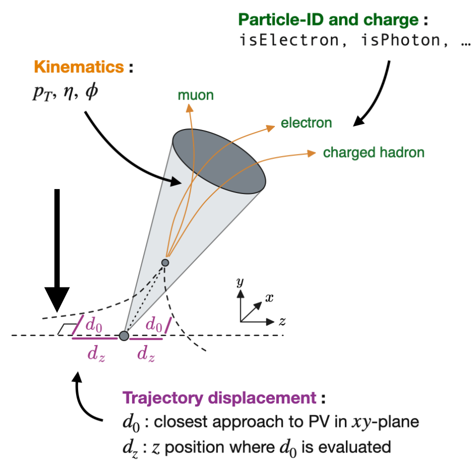
(Some) Simulation targets

Calorimeter Showers



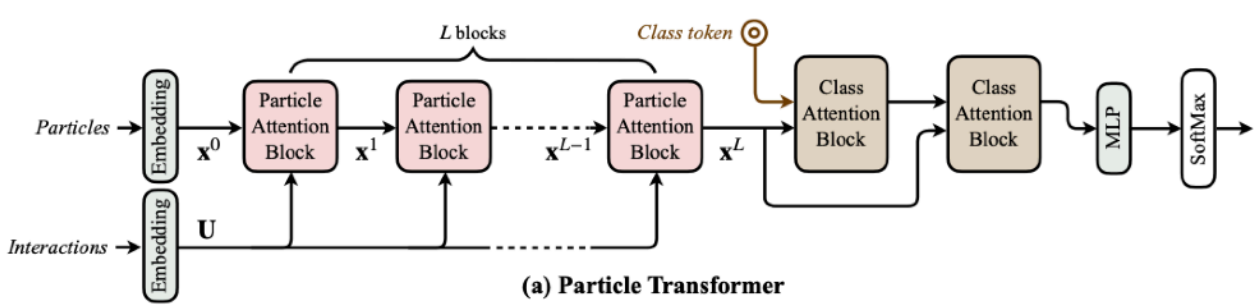
Reduce computational bottleneck

Jet Constituents



Predict background from data

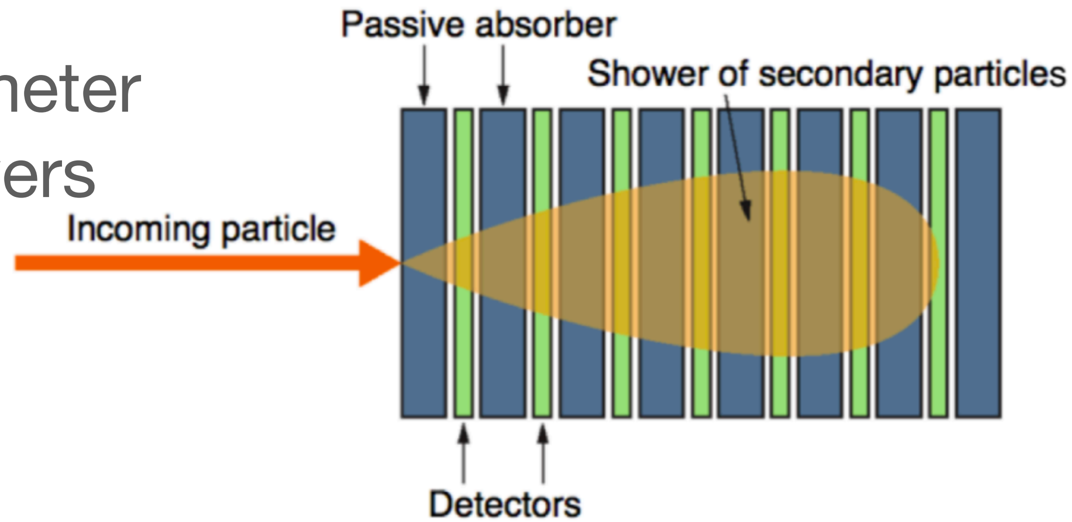
Classification and Reconstruction tasks



Act as surrogate models

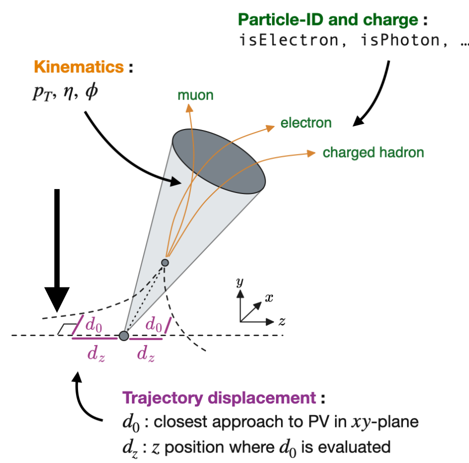
(Some) Simulation targets

Calorimeter Showers



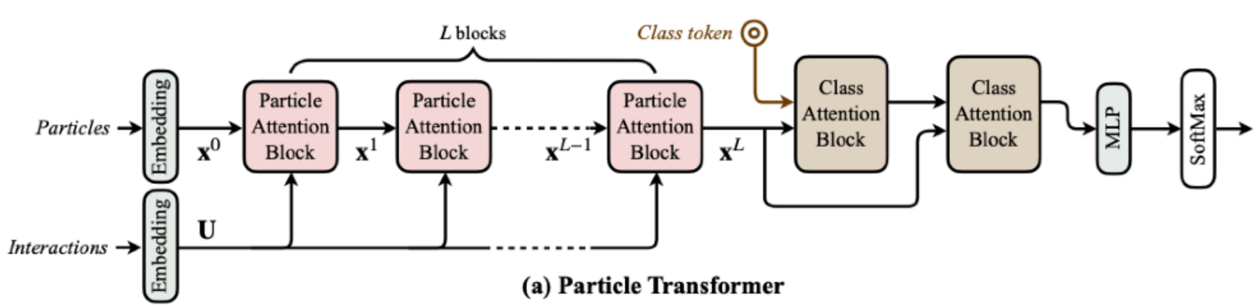
Reduce computational bottleneck

Jet Constituents



Predict background from data

Classification and Reconstruction tasks

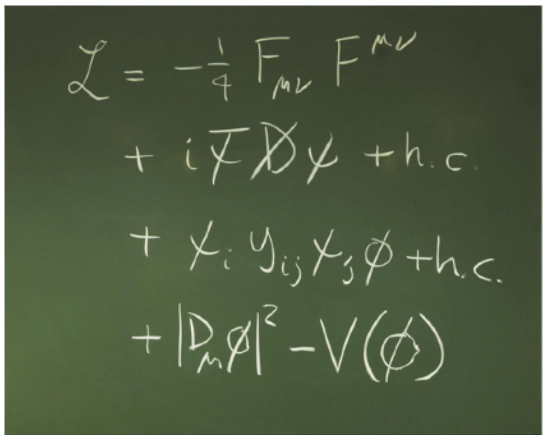


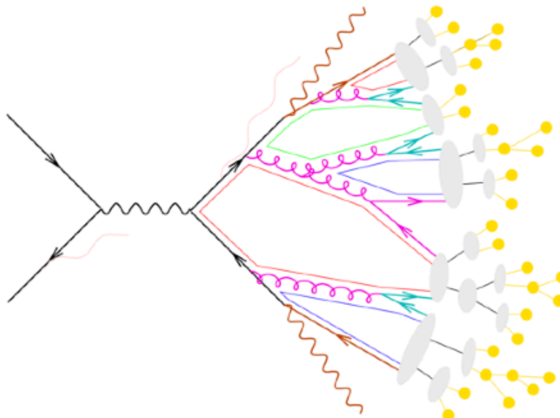
(a) Particle Transformer

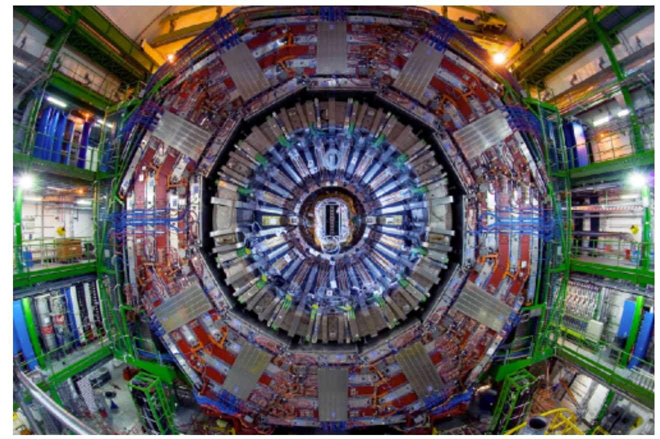
Act as surrogate models

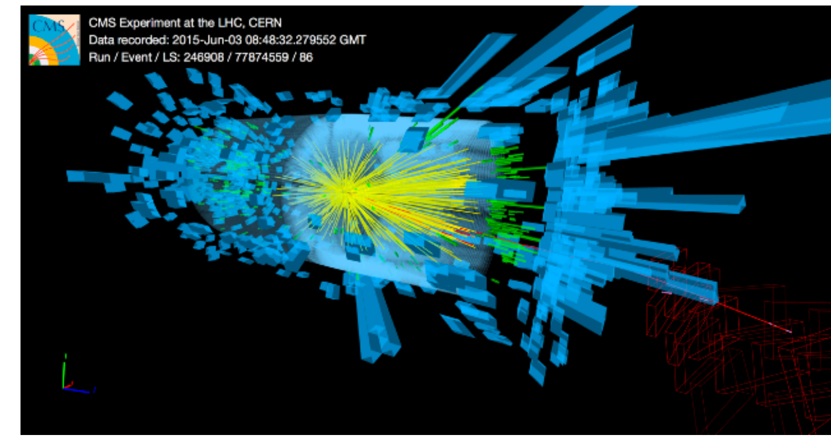
Generative Models

This happens in the experiment







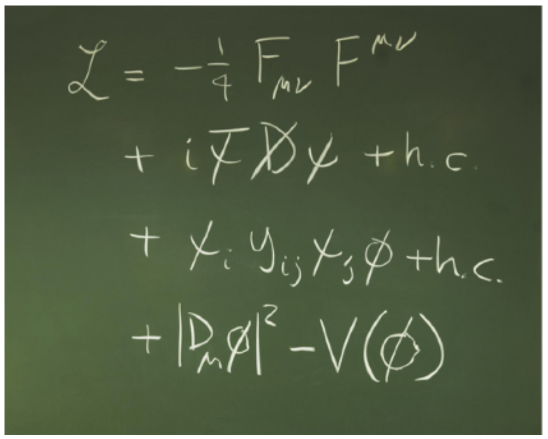


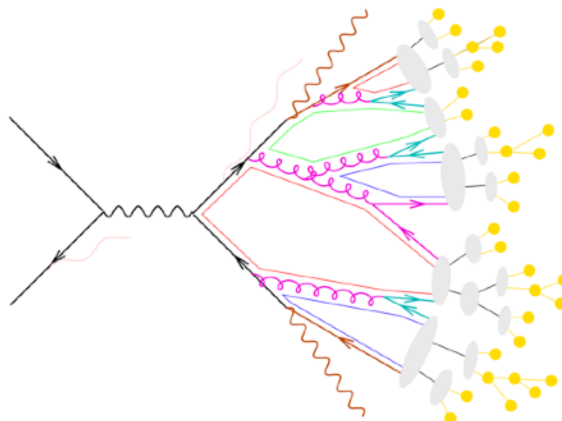
This is what we want to know

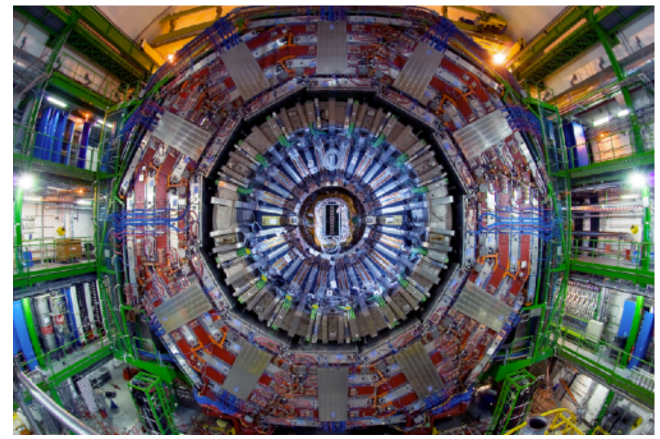
Simulation is crucial to connect experimental data with theory predictions

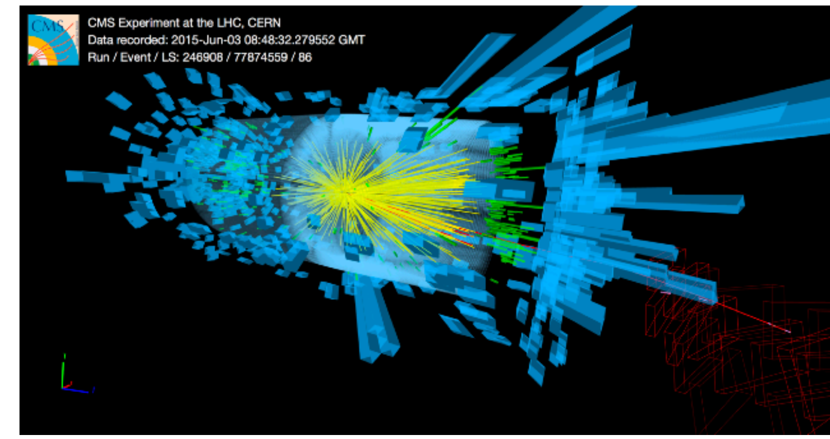
Generative Models

This happens in the experiment



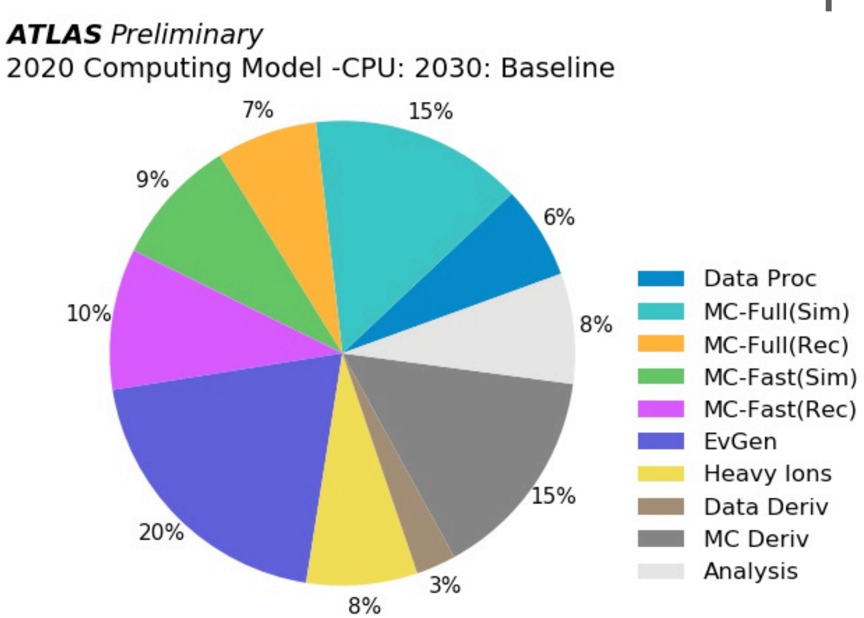


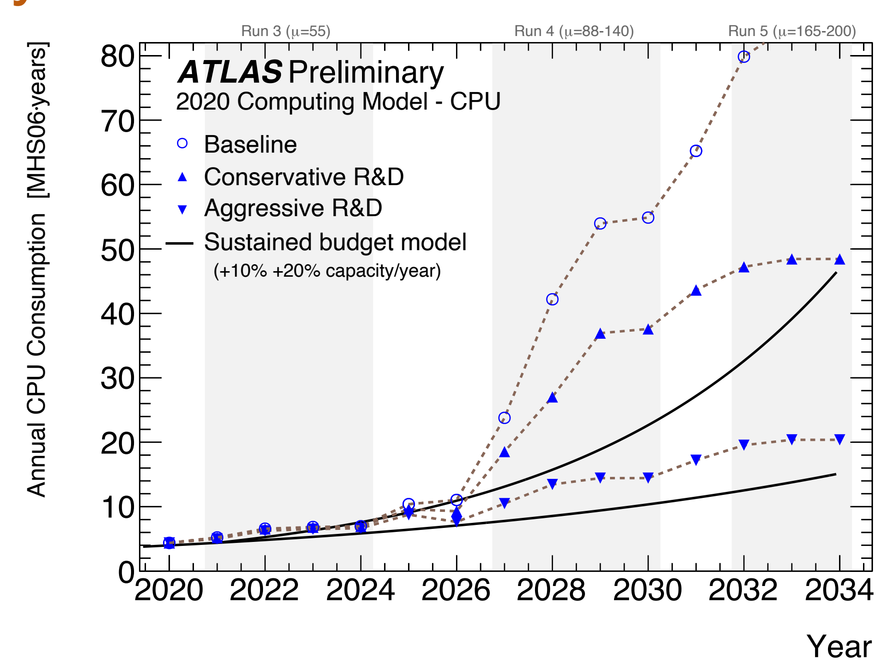




This is what we want to know

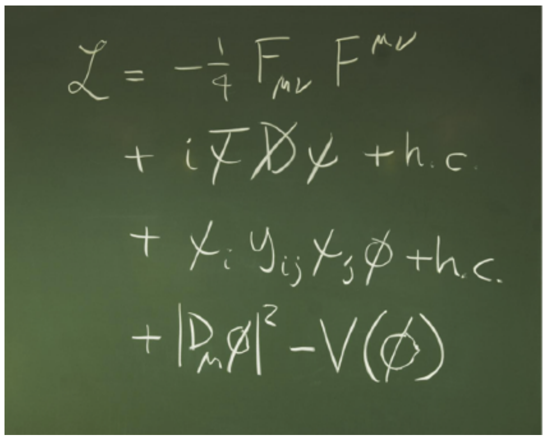
Simulation is crucial to connect experimental data with theory predictions, but computationally very costly

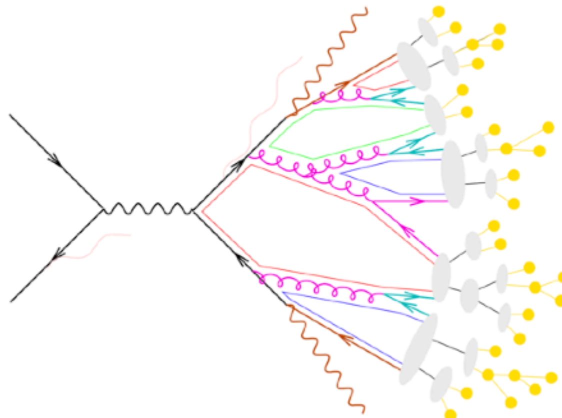


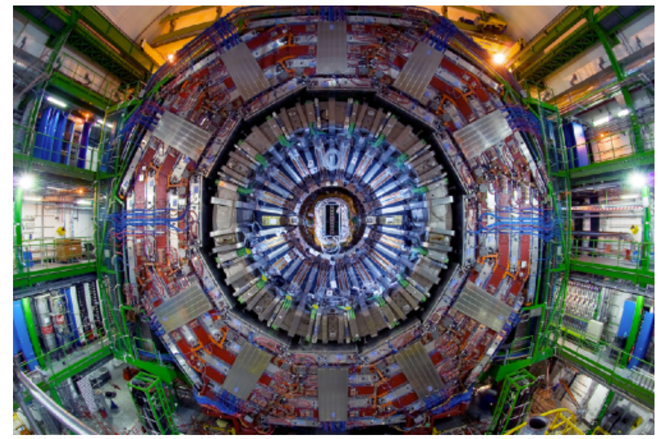


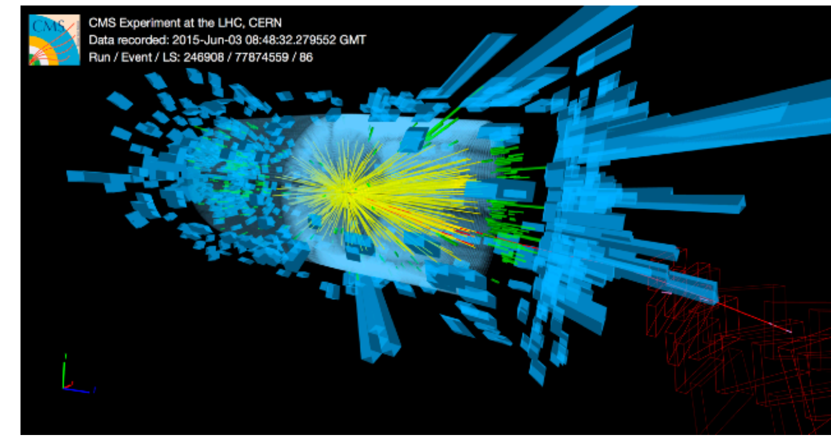
Generative Models

This happens in the experiment



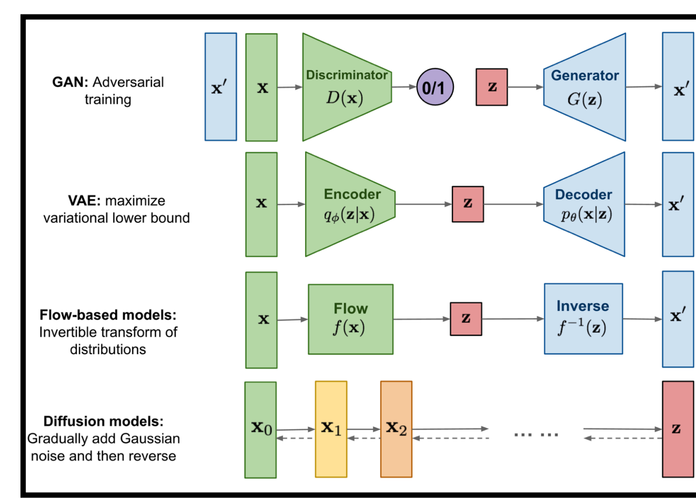






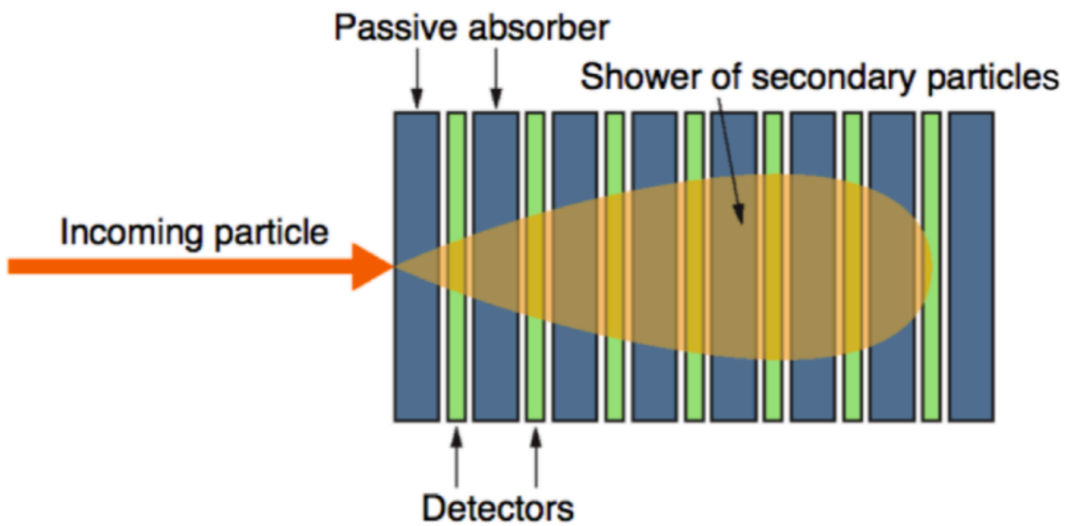
This is what we want to know

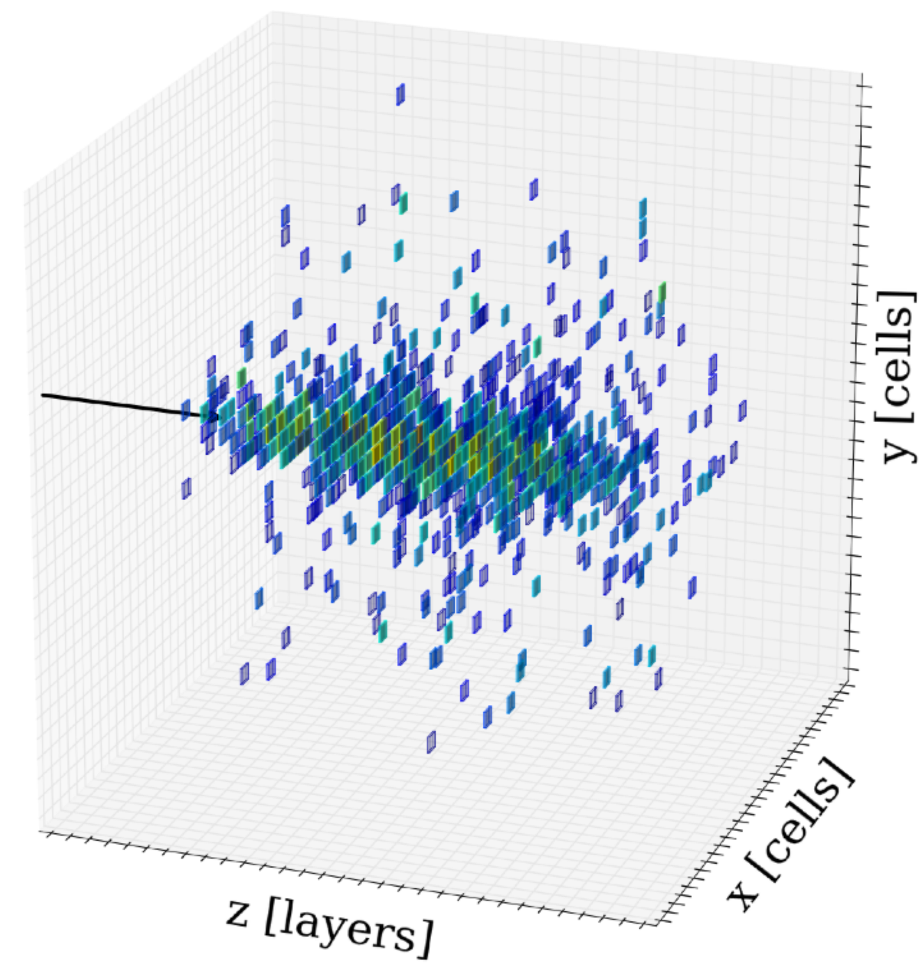
Simulation is crucial to connect experimental data with theory predictions, but computationally very costly



→Use generative models trained on simulation or data to augment simulations

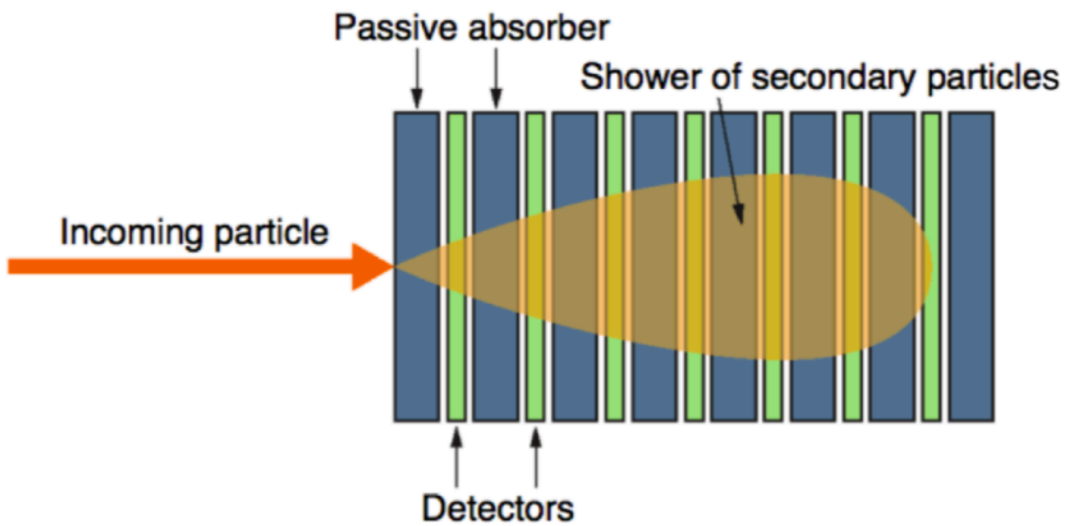
Simulation targets

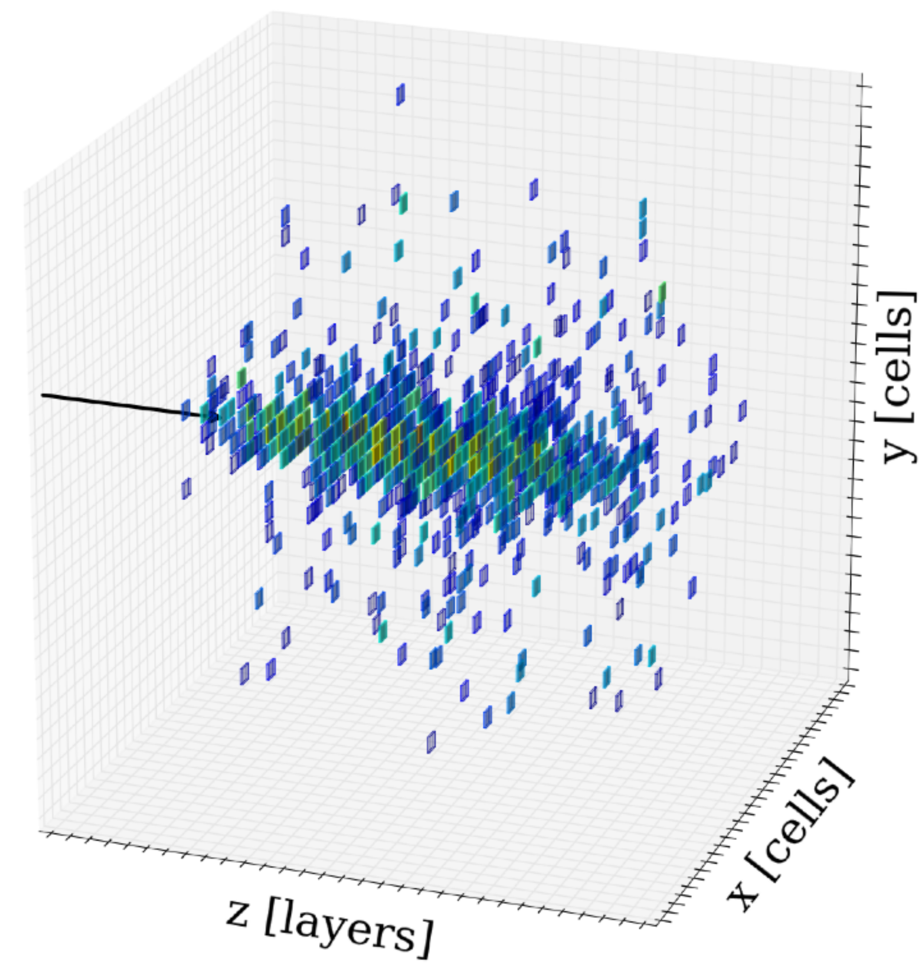




How to represent?

Simulation targets

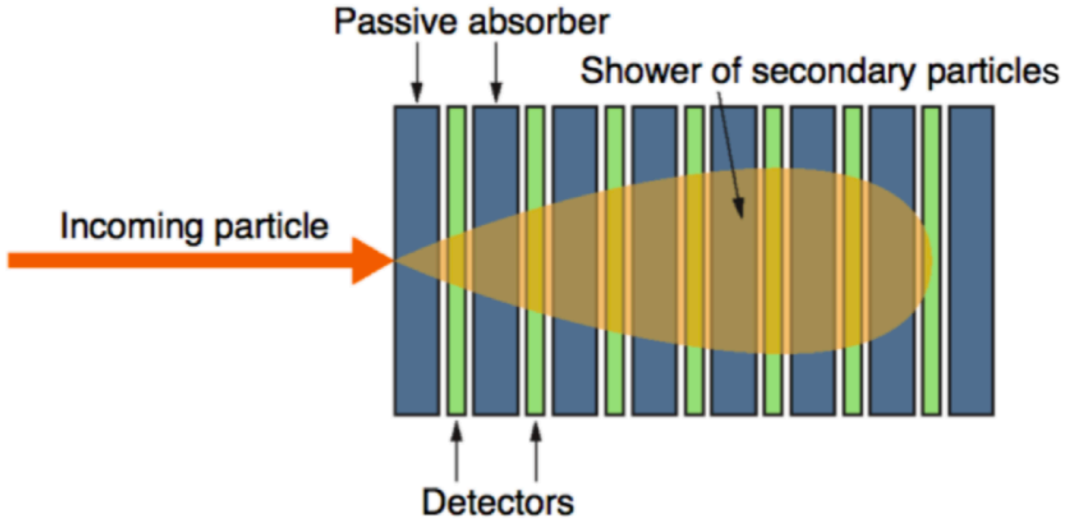


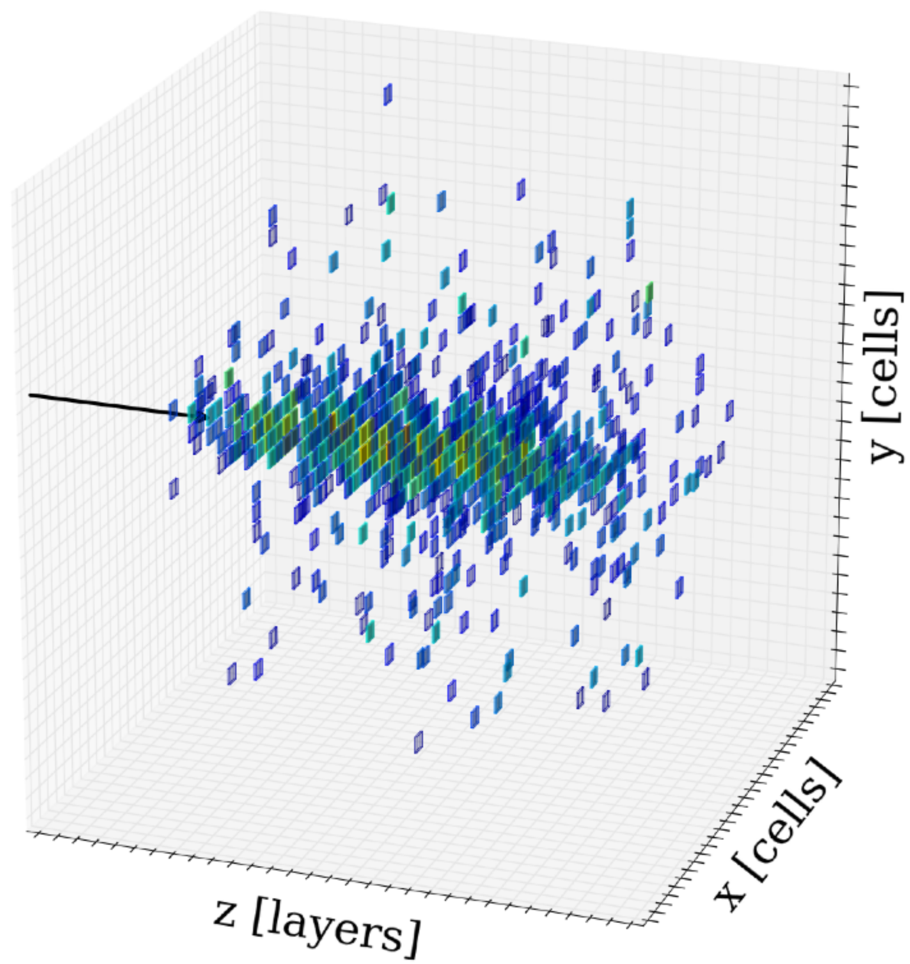


How to represent?

Tabular data:
Easy, insufficient for high-dimensions

Simulation targets



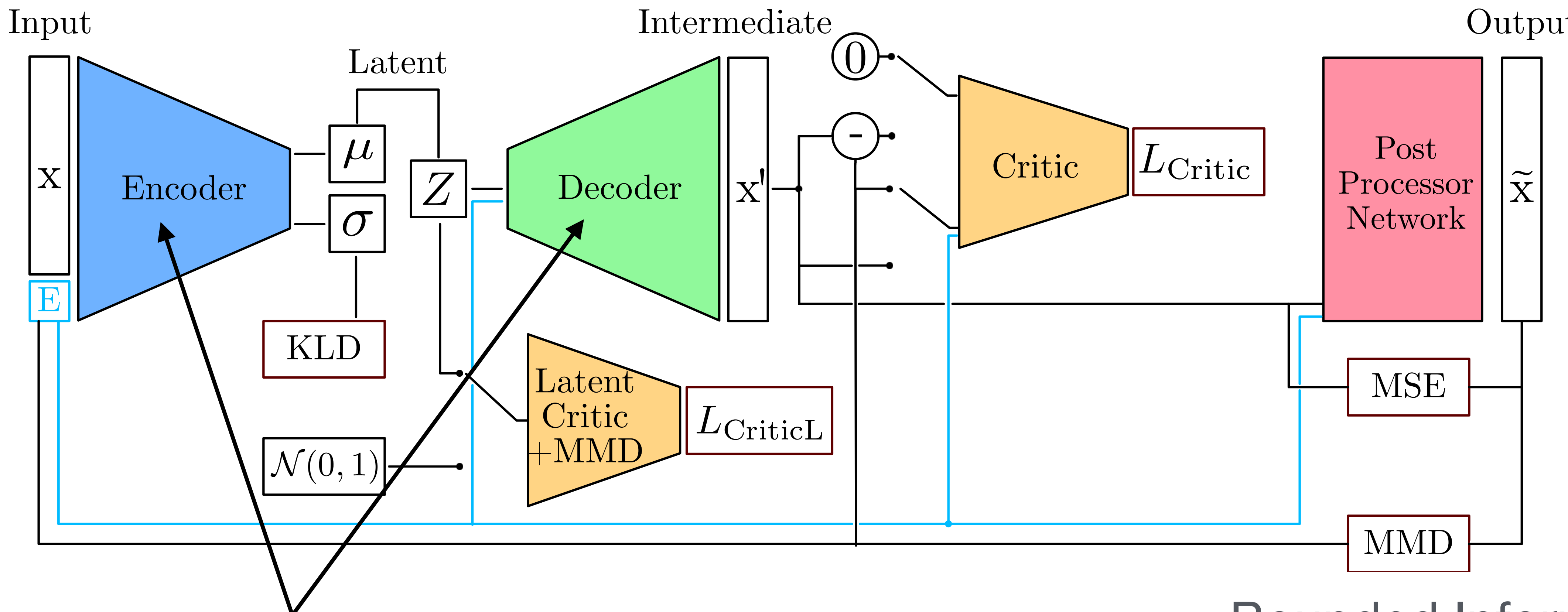


How to represent?

Tabular data

Fixed grid (voxels)

Generative results



(Transposed) Convolution

Bounded Information Bottleneck AE

BIB-AE (GAN + VAE)

$$
\begin{aligned}
L_{\text{BIB-AE}} = & -\beta_{C_L} \cdot \mathbb{E}[C_L(N_E(x))] \\
& -\beta_C \cdot \mathbb{E}[C_E(D_E(N_E(x)))] \\
& -\beta_{C_D} \cdot \mathbb{E}[C_{D,E}(D_E(N_E(x)) - x)] \\
& +\beta_{\text{KLD}} \cdot \text{KLD}(N_E(x)) \\
& +\beta_{\text{MMD}} \cdot \text{MMD}(N_E(x), \mathcal{N}(0,1)).
\end{aligned}
$$

Latent Critic

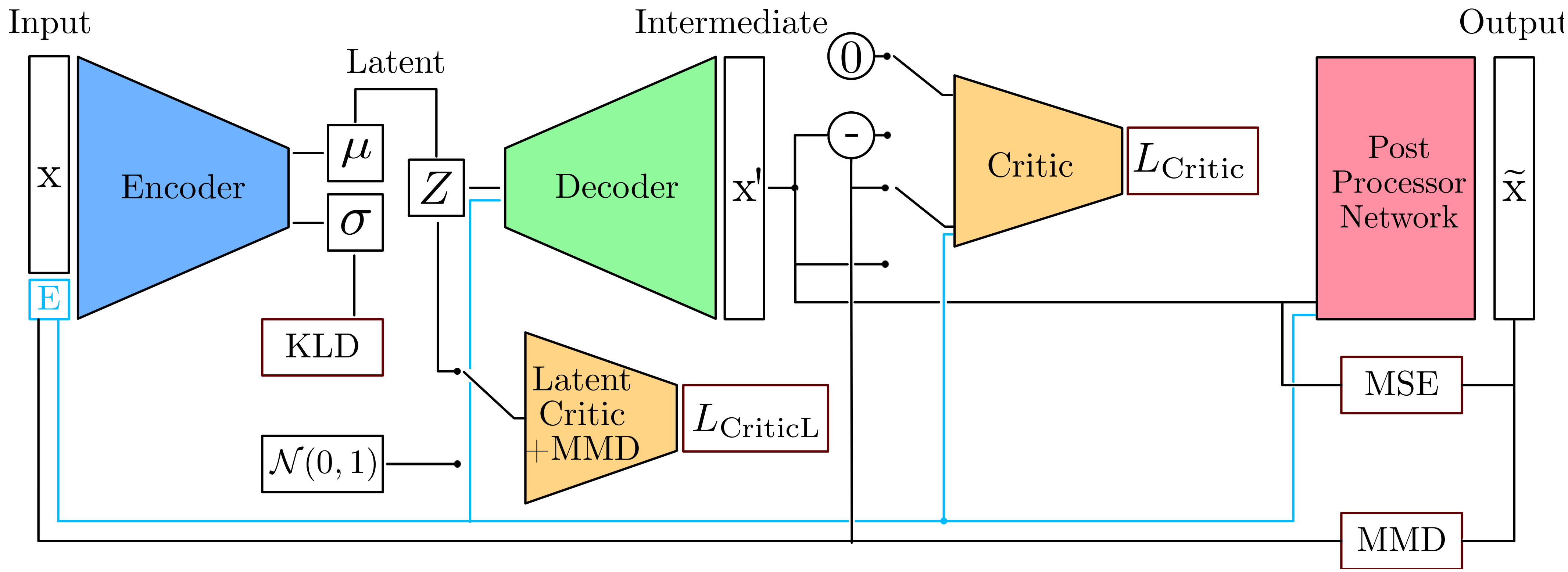
Critic

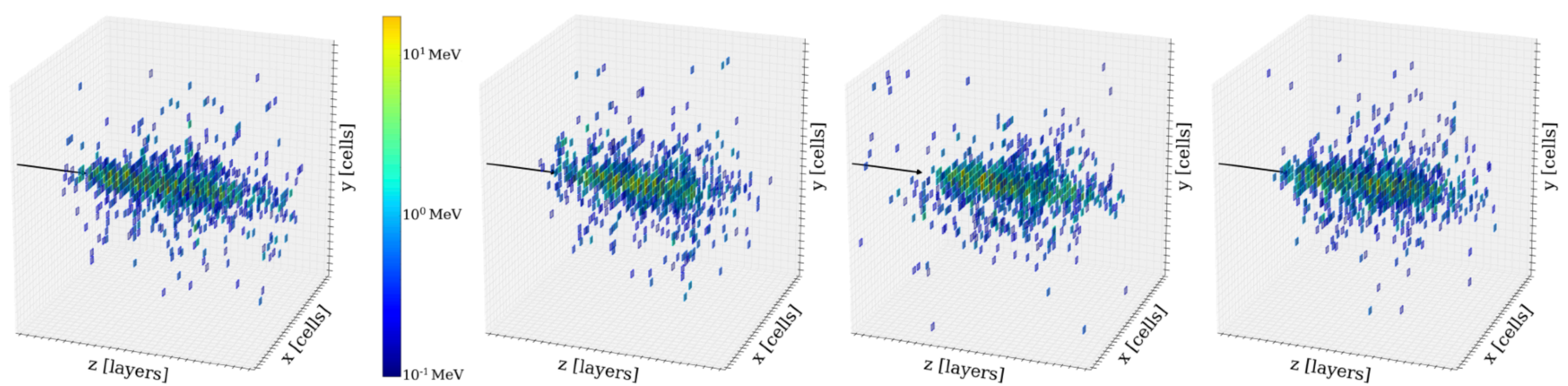
Difference Critic

Latent Regularisation

1912.00830, 2005.05334

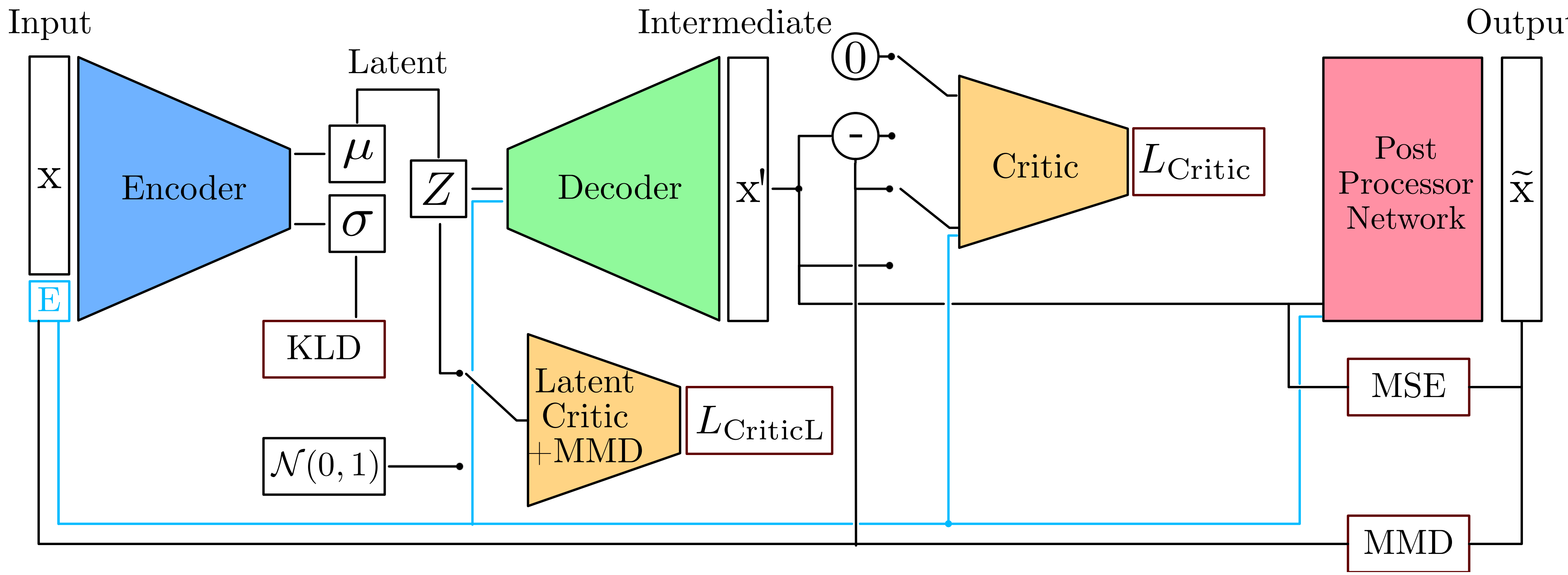
Generative results

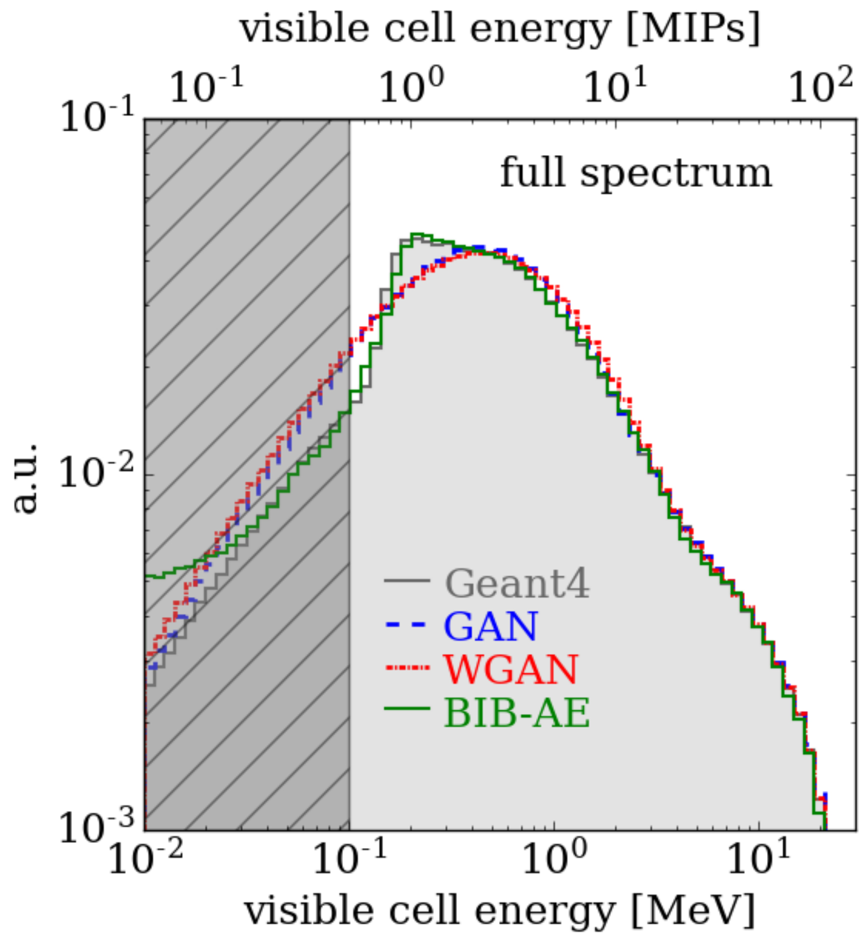




2005.05334

Generative results



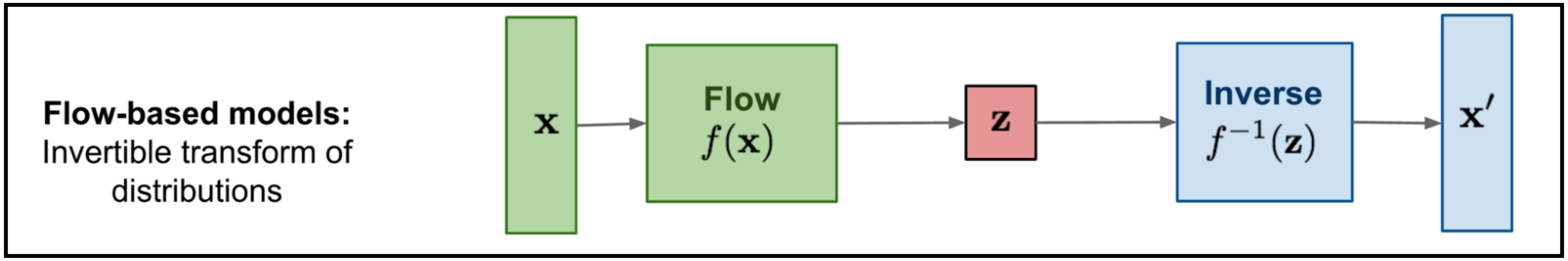


2005.05334

Go with the…

(Normalising) Flows

Generative models

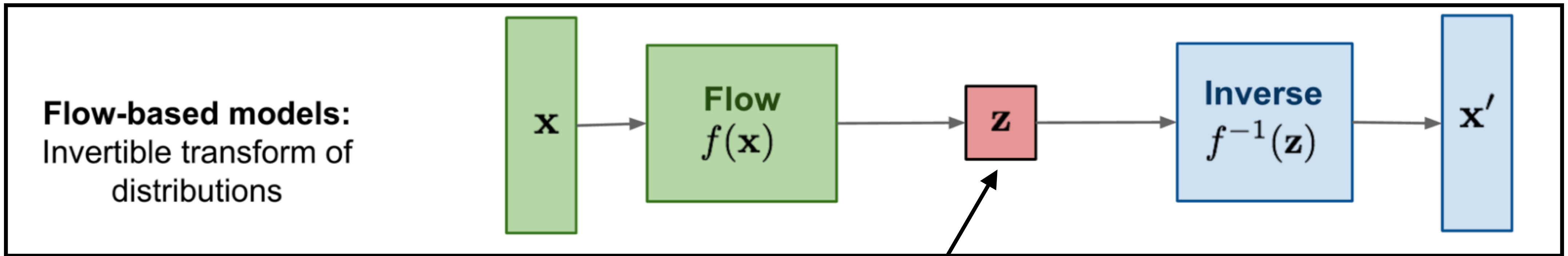


In auto-encoders, the decoder learns to ‘undo’ the encoder

Can we make this exact?

lilianweng.github.io

Generative models

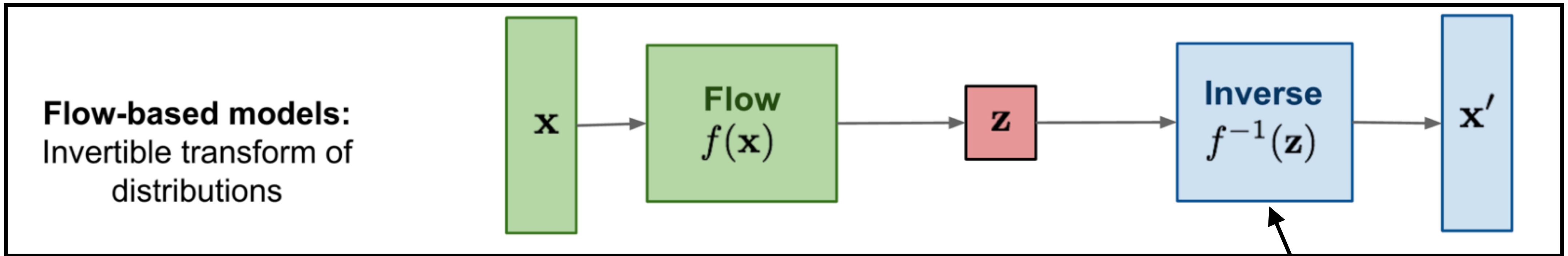


Choose latent space, e.g. standard normal distribution (normalising flow!)
Same dimension as data!

Learn a diffeomorphism between data and latent-space

1505.05770, 1908.09257

Generative models

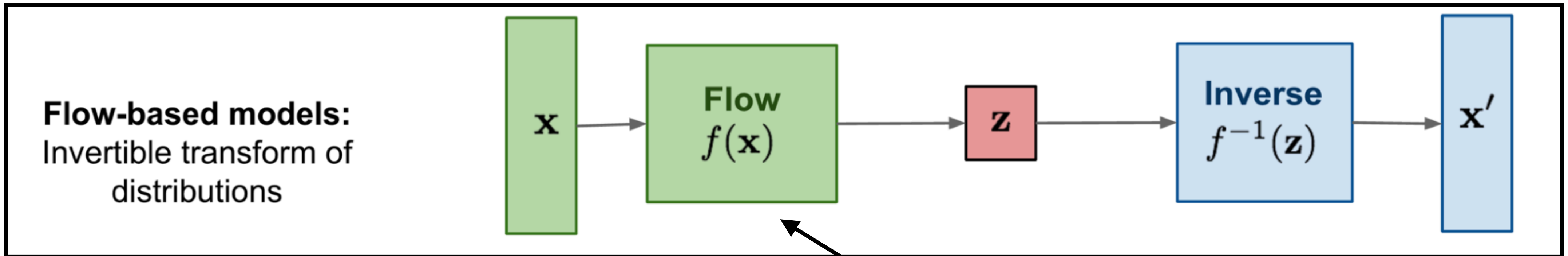


f^{-1} is not a learned inversion, but exact inverse by construction

Learn a diffeomorphism between data and latent-space

Bijective, invertable

Generative models



Take into account Jacobian determinant to evaluate probability density

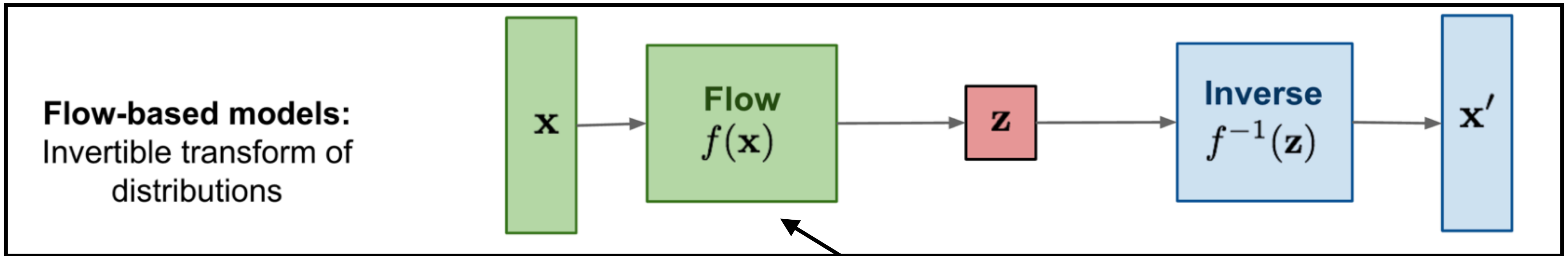
Learn a diffeomorphism between data and latent-space

Bijective, invertable

Learn likelihood of data

1505.05770, 1908.09257

Generative models



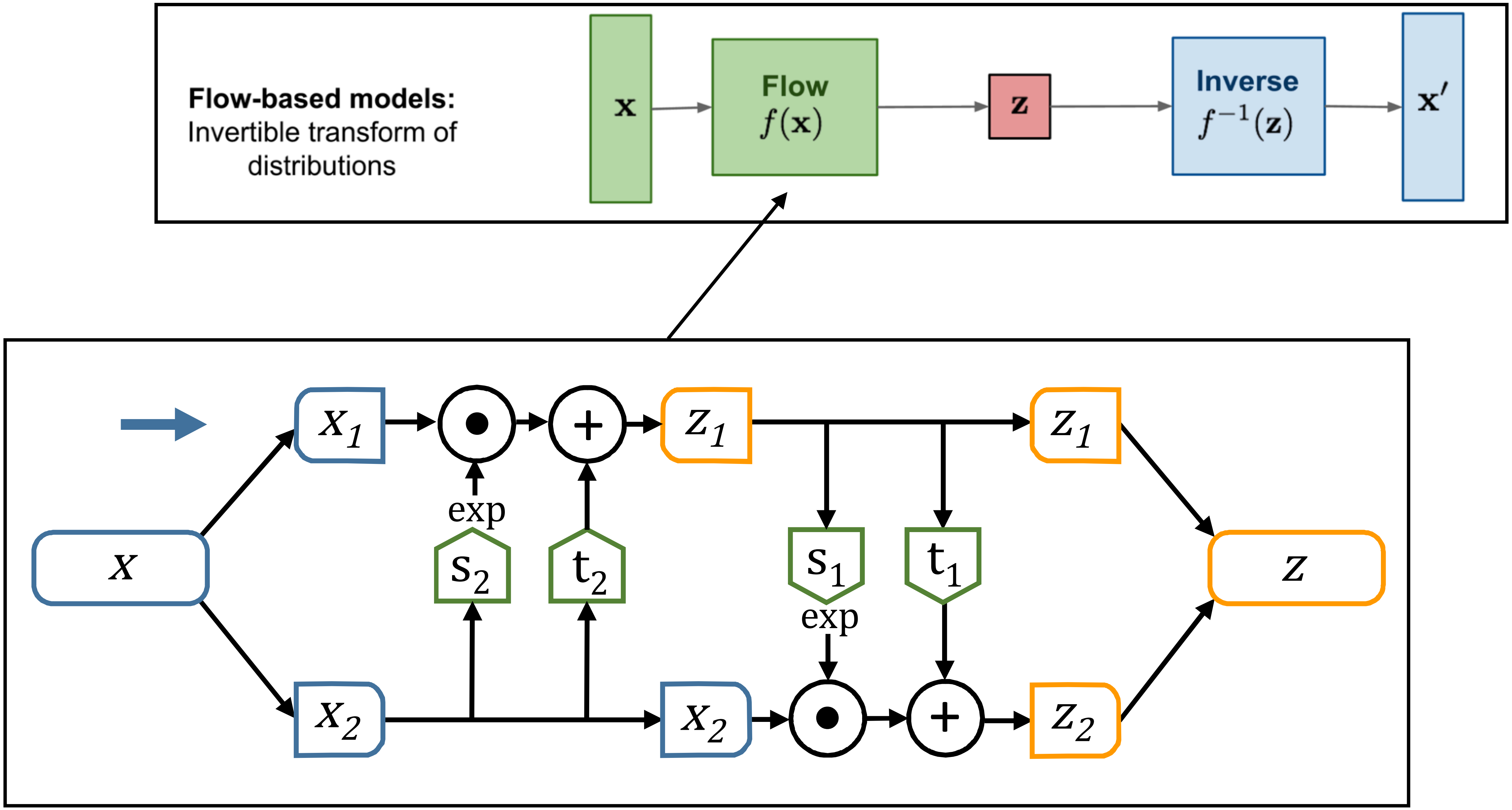
Take into account Jacobian determinant to evaluate probability density

2 challenges:

- Invertible
- Easy-to-calculate Jacobean

1505.05770, 1908.09257

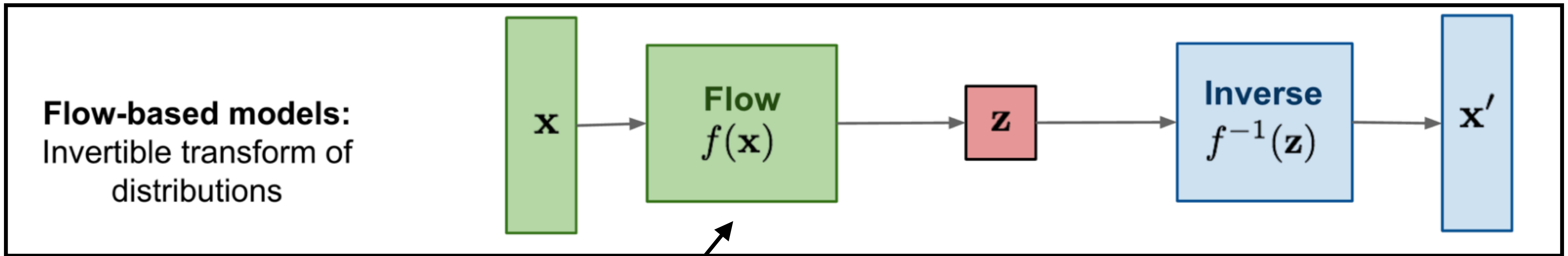
Coupling flows

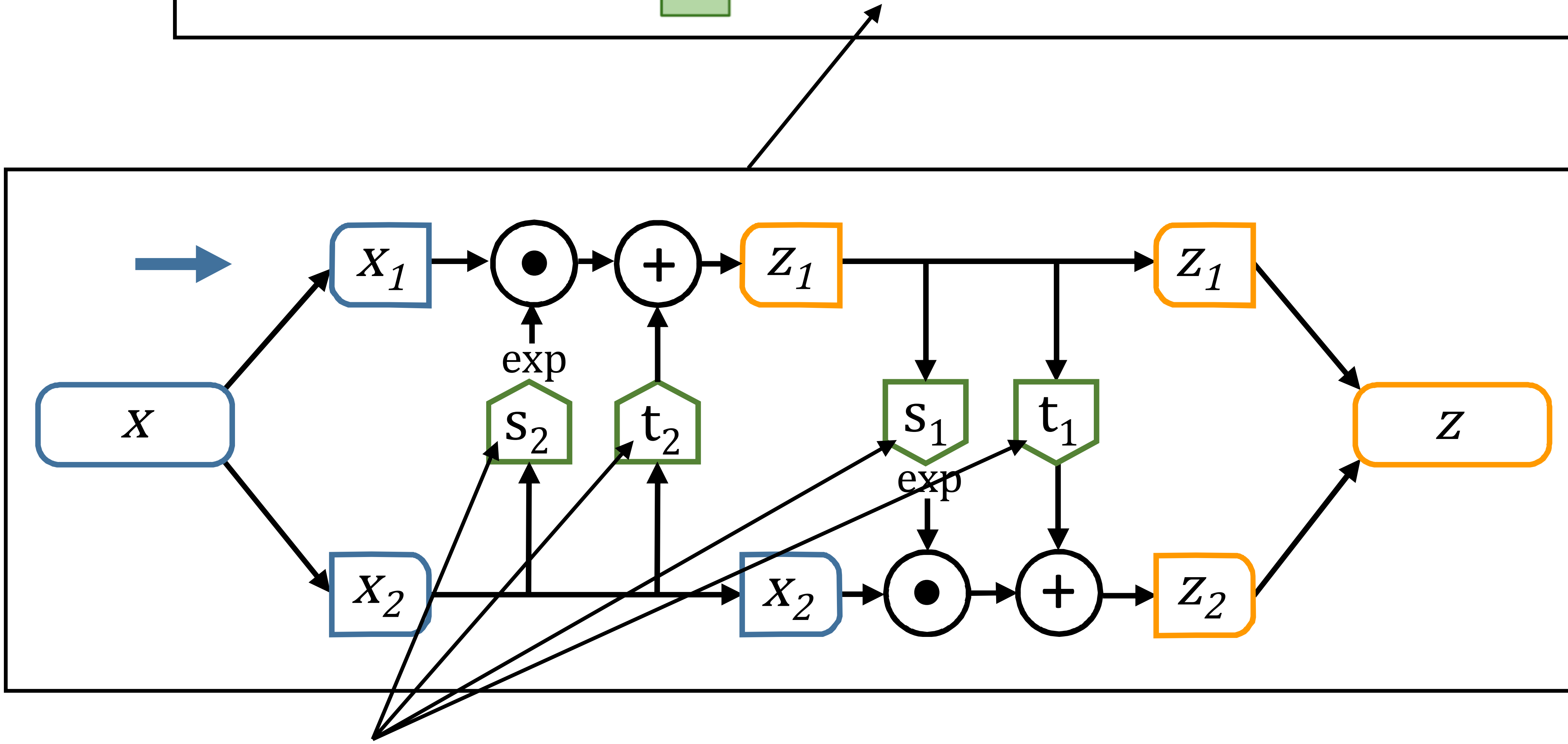


Coupling layers: Not the most expressive, but useful for illustration/understanding

1505.05770, 1908.09257

Coupling flows

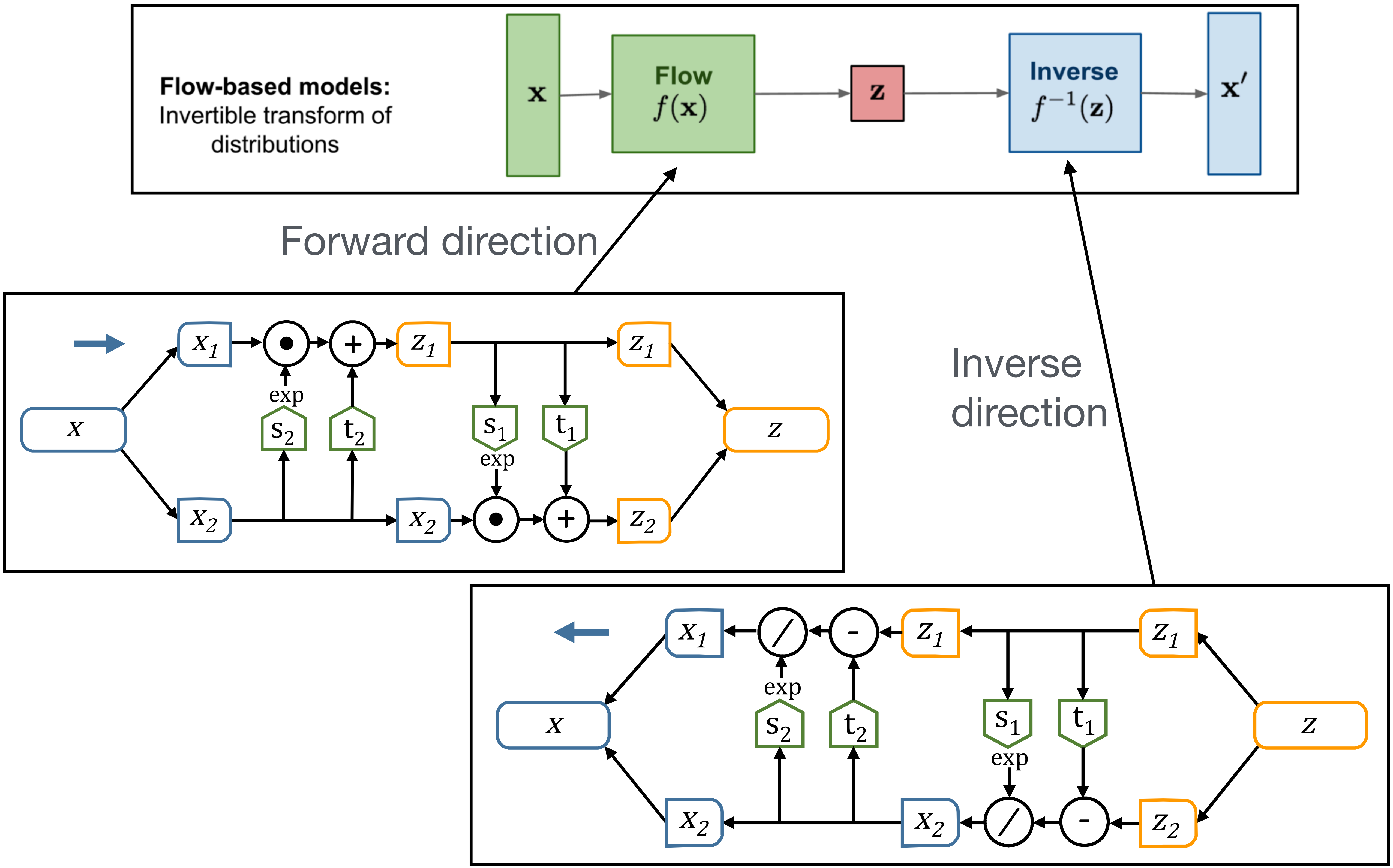




Simple (e.g. dense)
neural networks

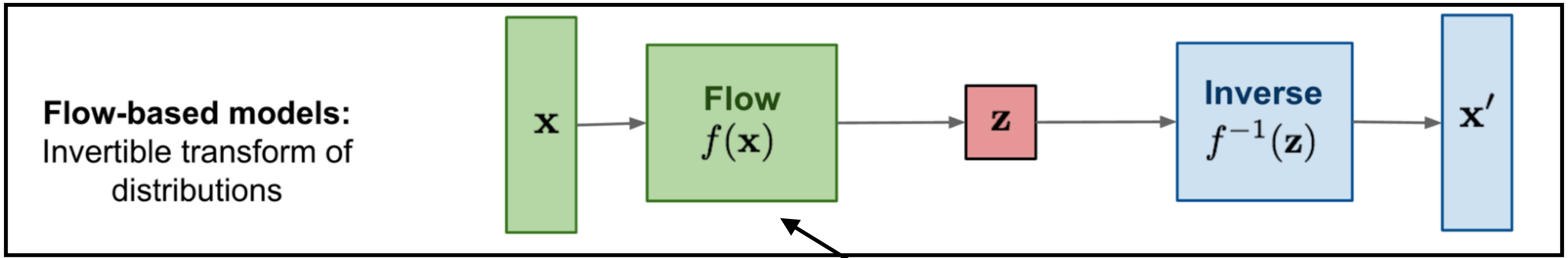
1505.05770, 1908.09257

Coupling flows



1505.05770, 1908.09257

Generative models

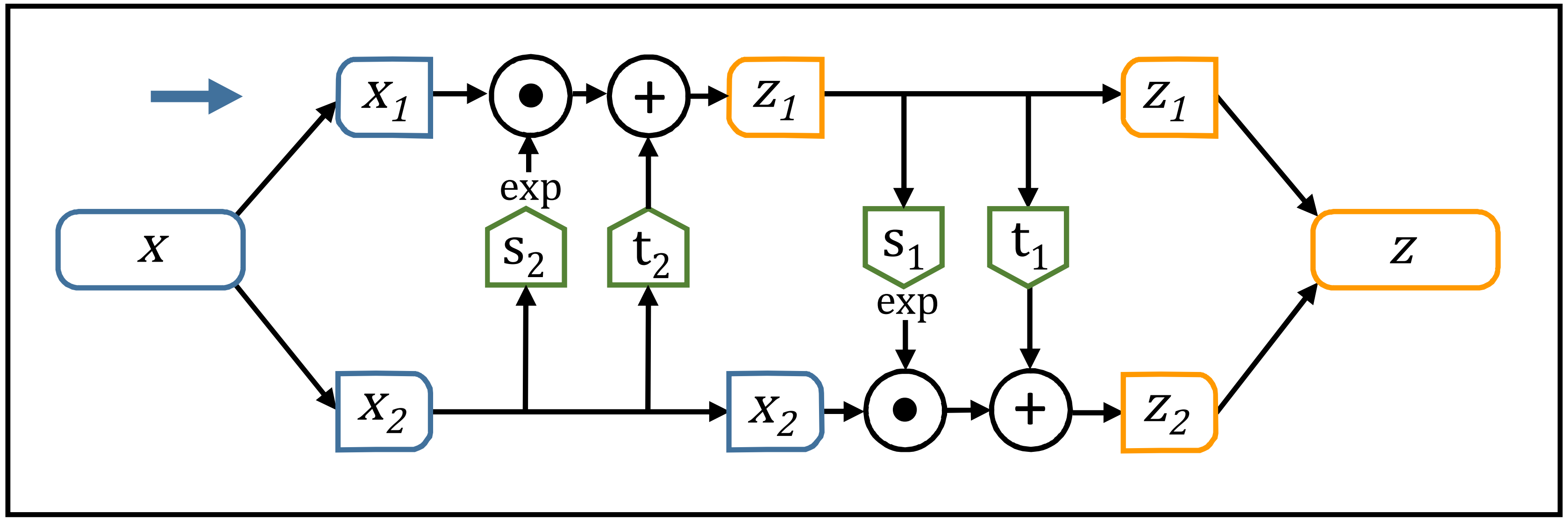


Take into account Jacobian determinant to evaluate probability density

2 challenges:
- Invertible
- Easy-to-calculate Jacobian

1505.05770, 1908.09257

Calculating Jacobian determinant



$$\begin{pmatrix}\mathbf{x}_1\\ \mathbf{x}_2\end{pmatrix} \xrightarrow{f_1} \begin{pmatrix}\mathbf{z}_1\\ \mathbf{x}_2\end{pmatrix} \xrightarrow{f_2} \begin{pmatrix}\mathbf{z}_1\\ \mathbf{z}_2\end{pmatrix}$$ with $$\begin{aligned}\mathbf{x}_1 &\xrightarrow{f_1} \mathbf{z}_1 = \mathbf{x}_1 \odot \exp(s_2(\mathbf{x}_2)) + t_2(\mathbf{x}_2)\\ \mathbf{x}_2 &\xrightarrow{f_1} \mathbf{x}_2.\end{aligned}$$

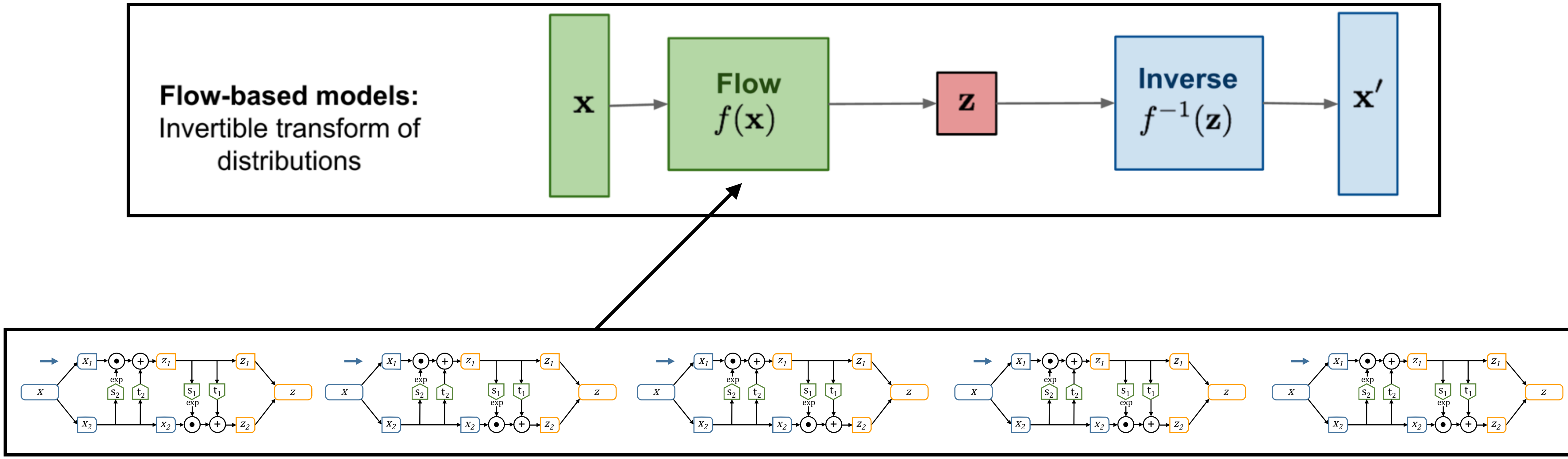
$$\mathbf{J_1} = \begin{pmatrix}\frac{\partial \mathbf{z}_1}{\partial \mathbf{x}_1} & \frac{\partial \mathbf{z}_1}{\partial \mathbf{x}_2}\\ \frac{\partial \mathbf{x}_2}{\partial \mathbf{x}_1} & \frac{\partial \mathbf{x}_2}{\partial \mathbf{x}_2}\end{pmatrix} = \begin{pmatrix}\mathrm{diag}(\exp(s_2(\mathbf{x}_2))) & \frac{\partial \mathbf{z}_1}{\partial \mathbf{x}_2}\\ 0 & \mathbb{1}\end{pmatrix}$$

Triangular by construction

$$\det \mathbf{J_1} = \prod \exp(s_2(\mathbf{x}_2)) = \exp\left(\sum s_2(\mathbf{x}_2)\right)$$

1505.05770, 1908.09257

Composition



Composition of bijective functions remains bijective

Chain rule: Jacobian determinant of composition is product of determinants

1505.05770, 1908.09257

How to train NF?

Training objective: Minimise negative log likelihood of data

Sample points from training data

$$\mathcal{L} = -\mathbb{E}_{\mathbf{x}\sim p_{\mathrm{data}}}\left[-\frac{1}{2}||f(\mathbf{x}))||_2^2 + \sum s(\mathbf{x})\right]$$

1505.05770, 1908.09257

How to train NF?

Training objective: Minimise negative log likelihood of data

$$\mathcal{L} = -\mathbb{E}_{\mathbf{x}\sim p_{\mathrm{data}}}\left[-\frac{1}{2}||f(\mathbf{x}))||_2^2 + \sum s(\mathbf{x})\right]$$

Transform into latent space and evaluate probability there

$$\varphi(\mathbf{z}) = \frac{1}{\sqrt{2\pi}}\mathrm{e}^{-\frac{1}{2}\mathbf{z}^2}$$

$$-\log(\varphi(f(\mathbf{x})) = -\log(\frac{1}{\sqrt{2\pi}}) - \log\left(\mathrm{e}^{-\frac{1}{2}f(\mathbf{x})^2}\right) = -\log(\frac{1}{\sqrt{2\pi}}) + \frac{1}{2}f(\mathbf{x})^2$$

How to train NF?

Training objective: Minimise negative log likelihood of data

$$\mathcal{L} = -\mathbb{E}_{\mathbf{x}\sim p_{\mathrm{data}}}\left[-\frac{1}{2}||f(\mathbf{x}))||_2^2 + \sum s(\mathbf{x})\right]$$

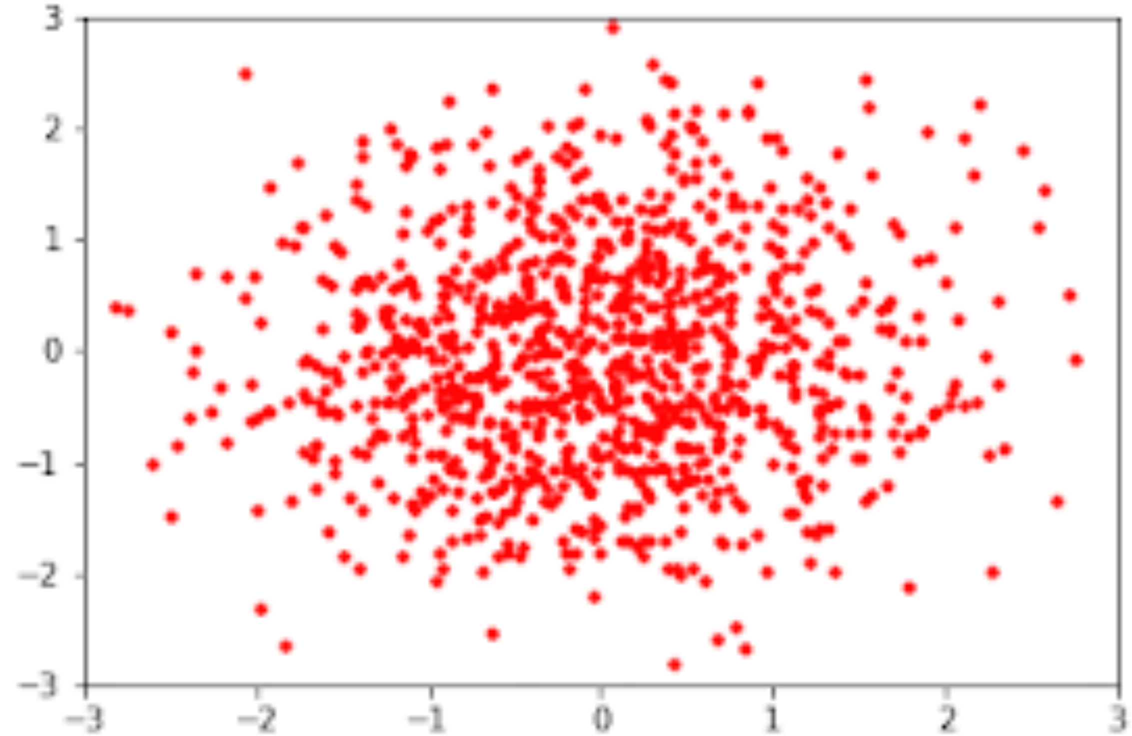
Contribution from Jacobian determinant

$$\det \mathbf{J} = \exp\left(\sum s(\mathbf{x})\right)$$

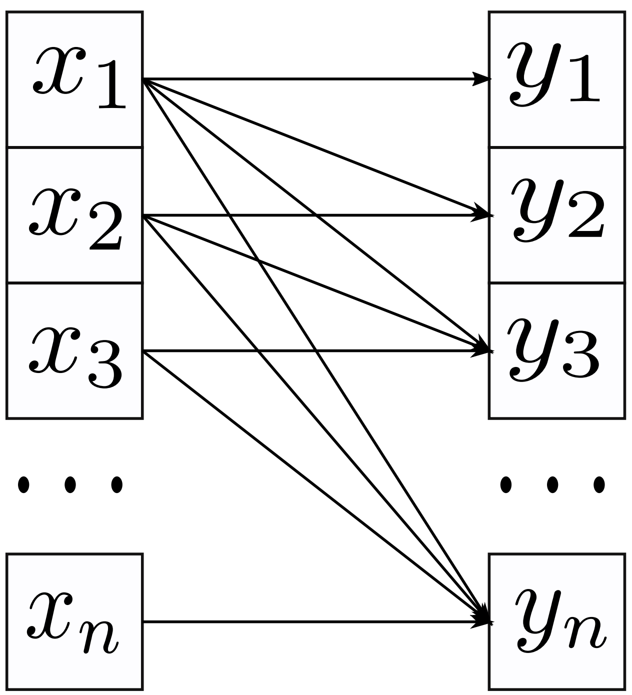
$$-\log(\det \mathbf{J}) = -\sum s(\mathbf{x})$$

1505.05770, 1908.09257

Animation



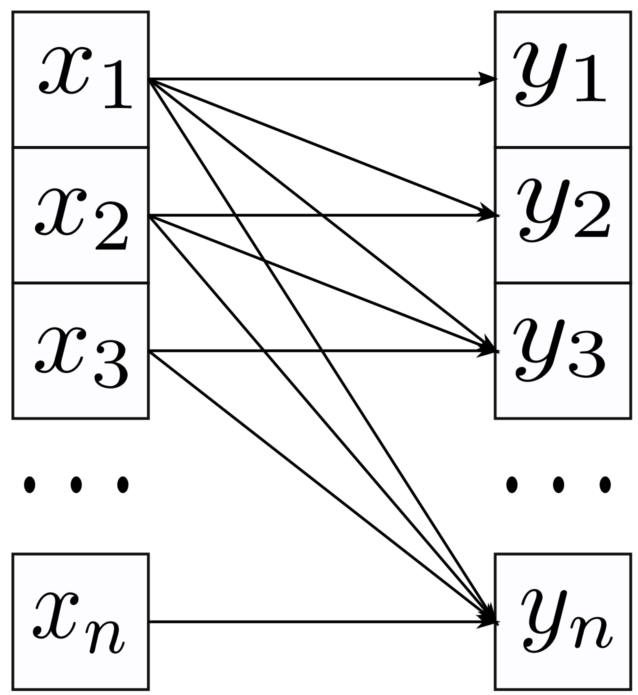
Autoregressive Flows



Alternative to coupling flows:
Outputs conditioned on previous inputs

1505.05770, 1908.09257

Autoregressive Flows



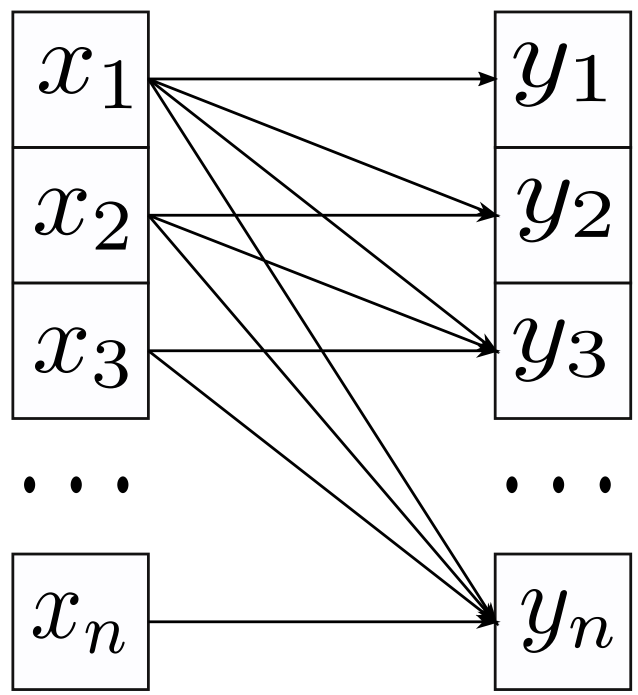
Bijective function Parameters

$$y_t = h(x_t; \Theta_t(\mathbf{x}_{1:t-1}))$$

Alternative to coupling flows:
Outputs conditioned on previous inputs

Again: simple Jacobian and invertible functions

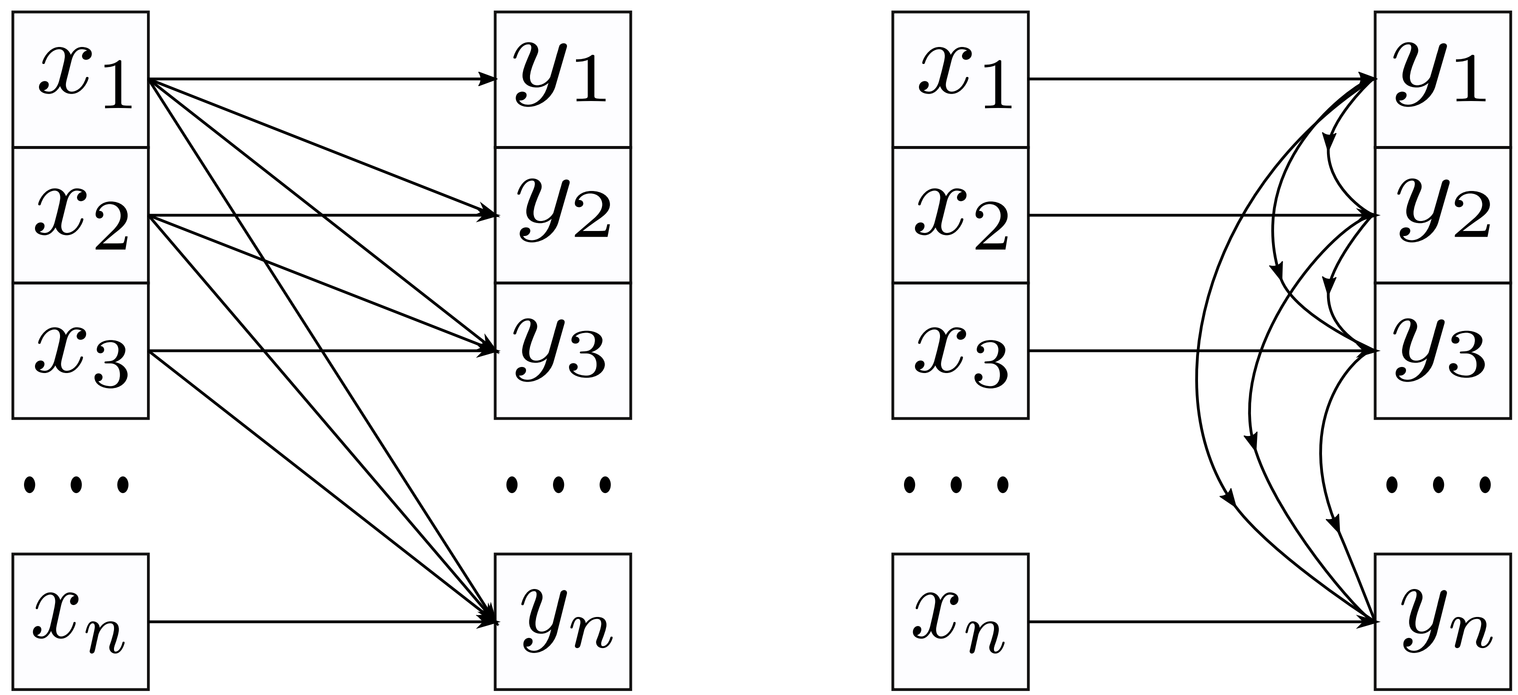
Autoregressive Flows



$$y_t = h(x_t; \Theta_t(\mathbf{x}_{1:t-1}))$$

Masked autoregressive flow (MAF):
Fast: Data → latent space
Slow: Latent space → data

Autoregressive Flows



$$y_t = h(x_t; \Theta_t(\mathbf{x}_{1:t-1}))$$

$$y_t = h(x_t; \theta_t(\mathbf{y}_{1:t-1}))$$

Masked autoregressive flow (MAF):
Fast: Data → latent space
Slow: Latent space → data

Inverse autoregressive flow (IAF):
Slow: Data → latent space
Fast: Latent space → data

Comments on Flows

Only scratched the surface:
more constructions available

§3.1 Elementwise bijections
Non-linear elementwise transform
Problem: no mixing of variables

§3.2 Linear flows
Affine combination of variables
Problem: limited representational power

§3.3 Planar and radial flows
Non-linear transforms
Problem: hard to compute inverse

§3.4.1 Coupling flows
§3.4.2 Autoregressive flows
Architectures that allow invertible non-linear transformations.
Depend on coupling functions

§3.4.4 Coupling functions

- Affine
- Non-linear squared
- Continuous mixture CDFs
- Splines
- Neural autoregressive
- Sum-of-squares polynomial
- Piecewise-bijective

§3.5 Residual flows
Invertible residual networks

§3.6 Infinitesimal flows
Continuous flows depending on ODEs or SDEs

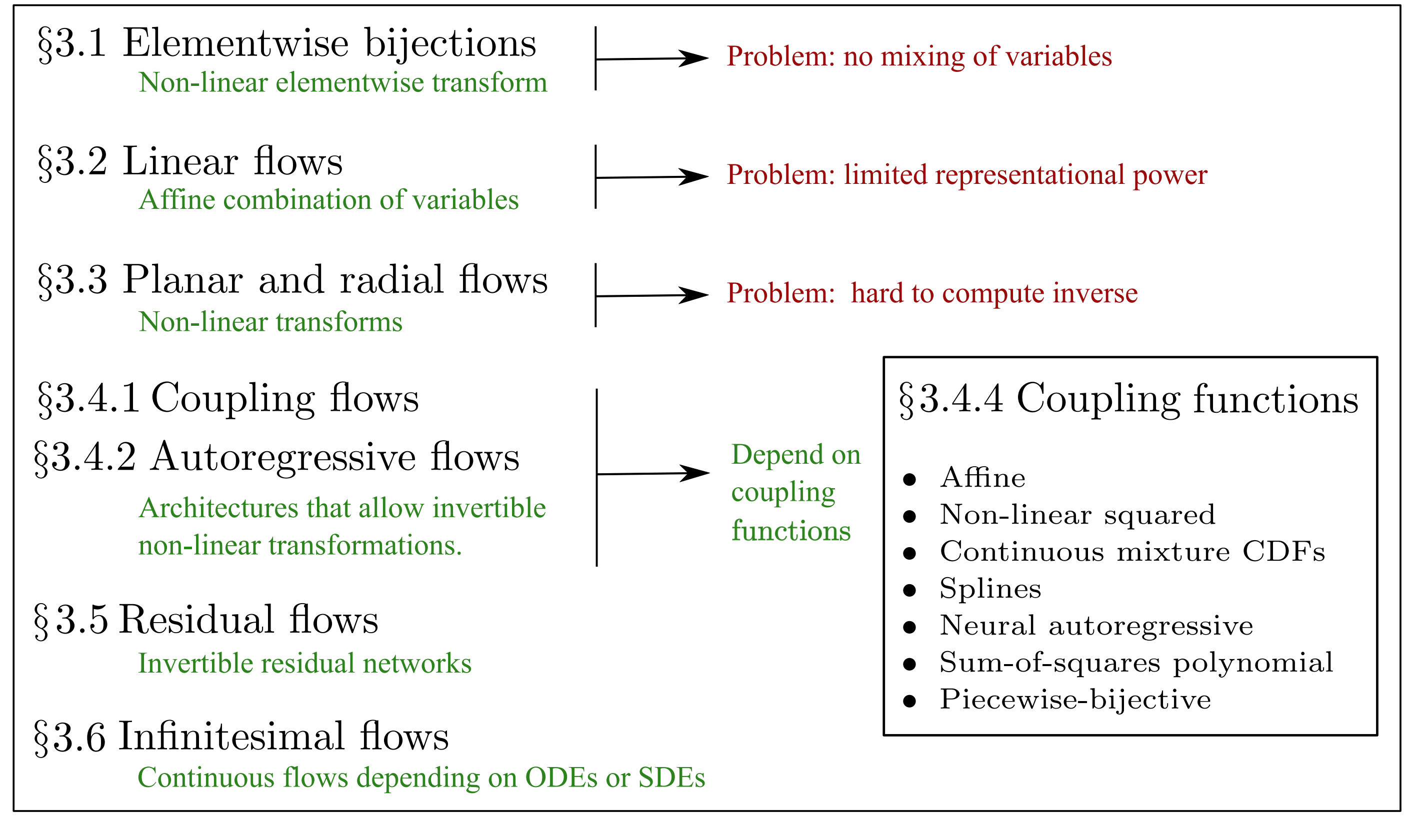
1908.09257

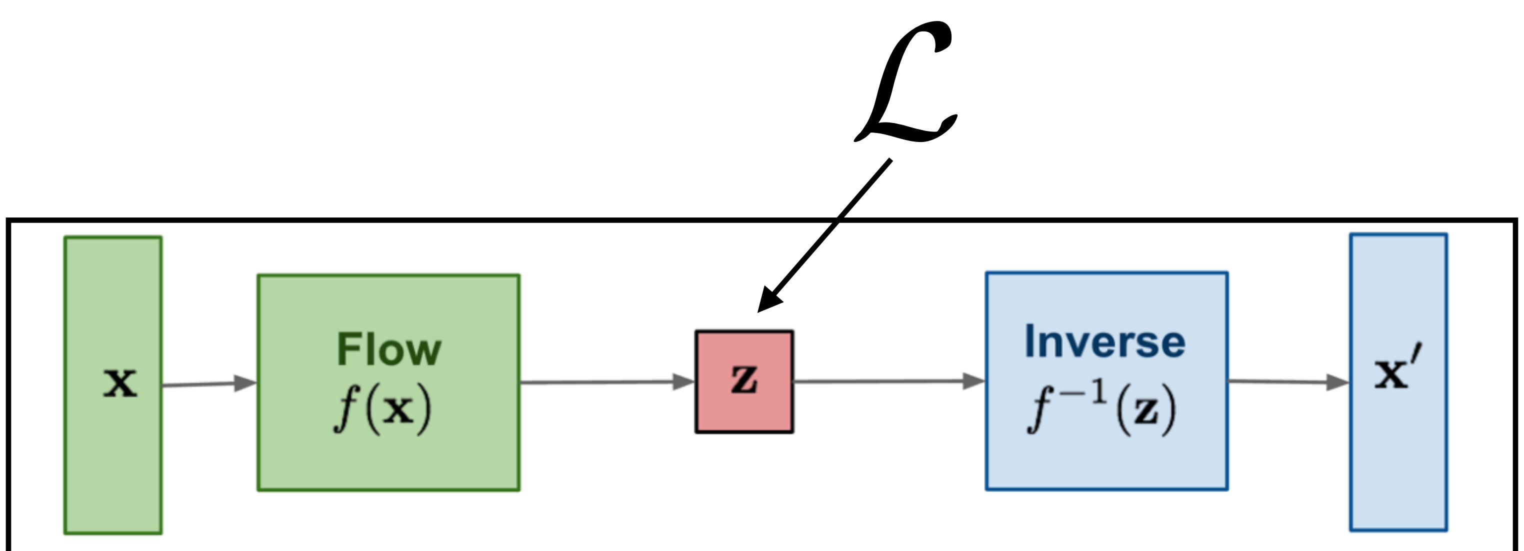
Comments on Flows

Only scratched the surface:
more constructions available

Exact learning of likelihood
→ Better generative fidelity
→ Can evaluate likelihood of data

More complex
→ Slower, choice of fast direction





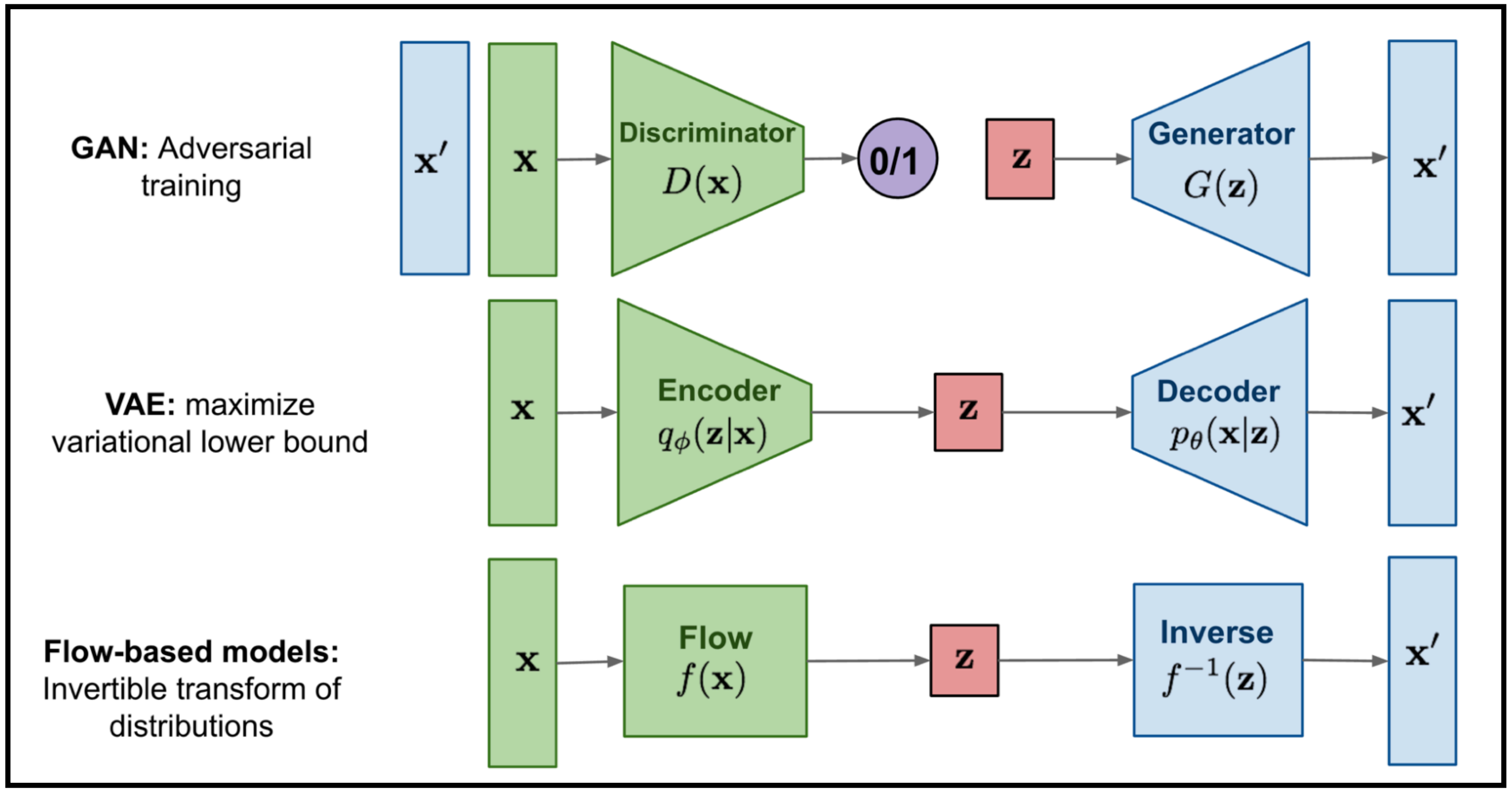
1908.09257

Simulation & Generative Models Part II

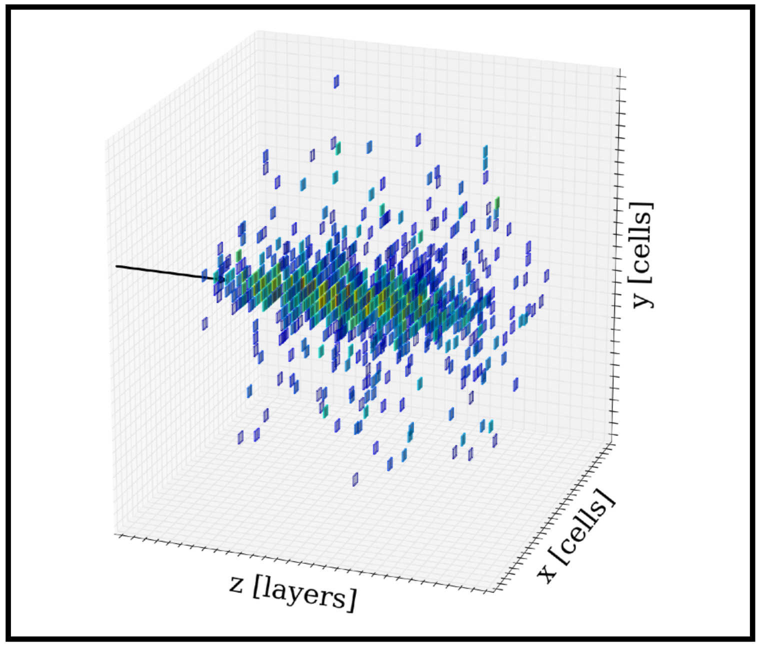
Gregor Kasieczka
Email: gregor.kasieczka@uni-hamburg.de
Twitter/X: @GregorKasieczka
COFI Winter School 2023

CLUSTER OF EXCELLENCE
QUANTUM UNIVERSE
KISS
CDCS
CENTER FOR DATA AND COMPUTING
IN NATURAL SCIENCES
FSP CMS
PUNCH 4NFDI
DASHH
GEFÖRDERT VOM
Bundesministerium für Bildung und Forschung
Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG
PIER
Partnership of Universität Hamburg and DESY
Emmy Noether-Programm
Deutsche Forschungsgemeinschaft
DFG

Yesterday

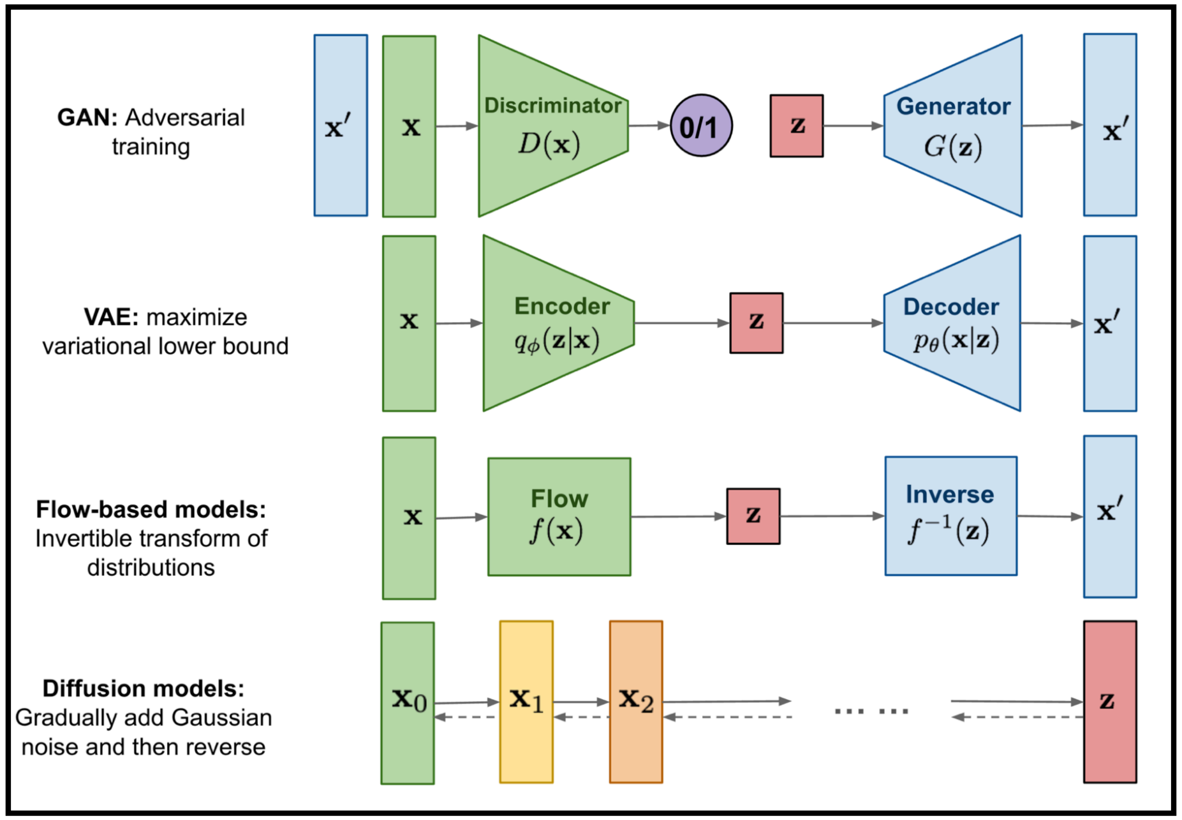


Understand three basic generative architectures



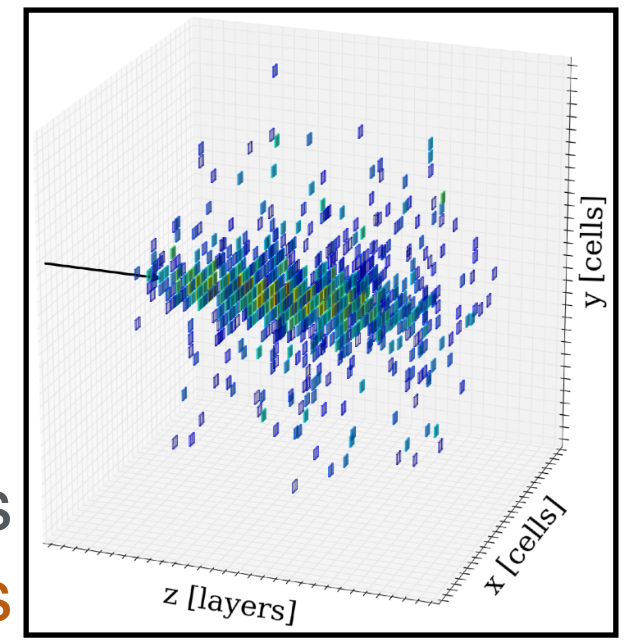
First look at simulating fixed grid data

Overview

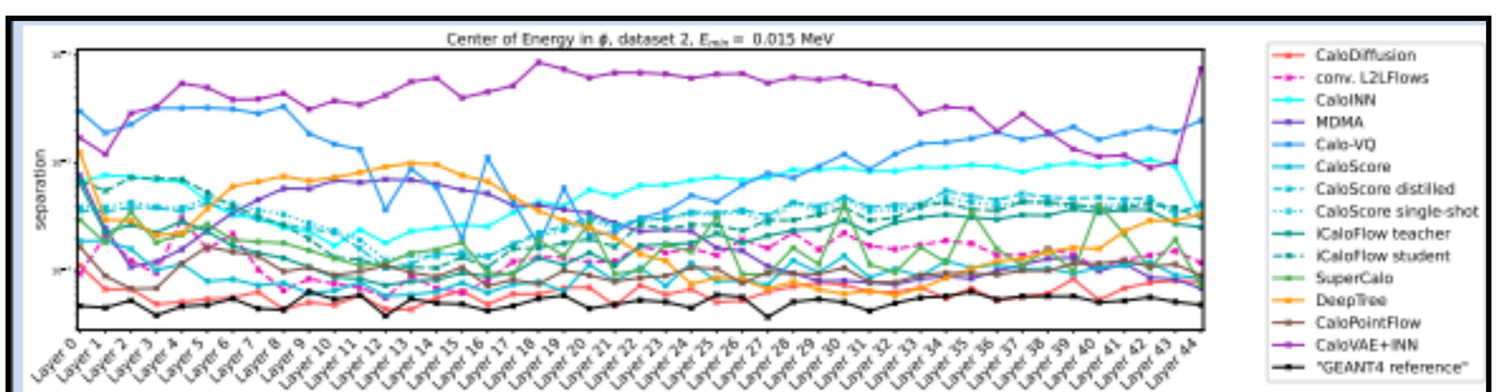


1. Common architectures
   -> GANs, VAEs, NF yesterday
   -> Diffusion & CNF today

2. Physics applications

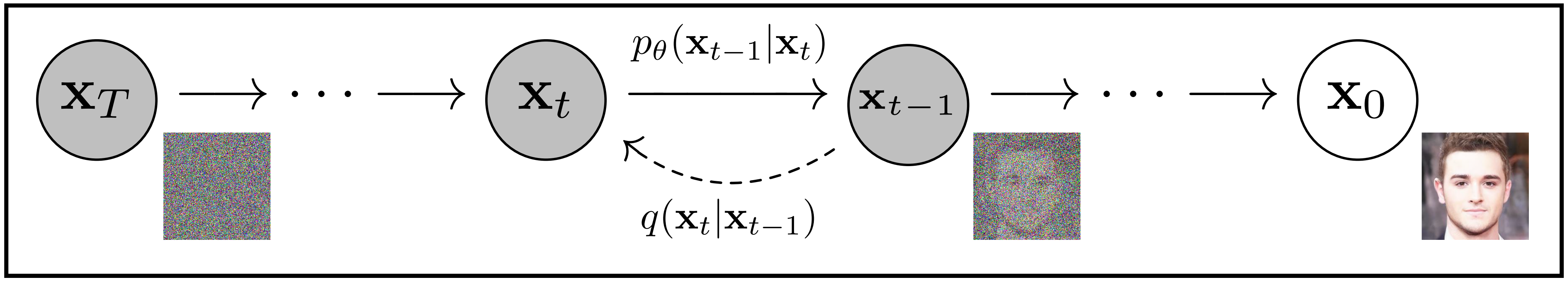


3. Quality metrics



Diffusion Models

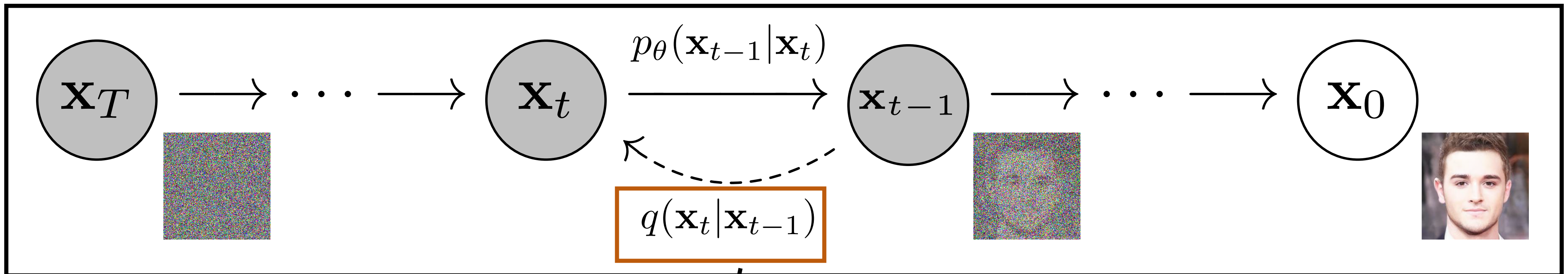
Diffusion Model



Core idea: Stepwise transition from pure noise to data

Markov chain

Diffusion Model



Core idea: Stepwise transition from pure noise to data

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

Individual step

Noise schedule (hyper-parameter)

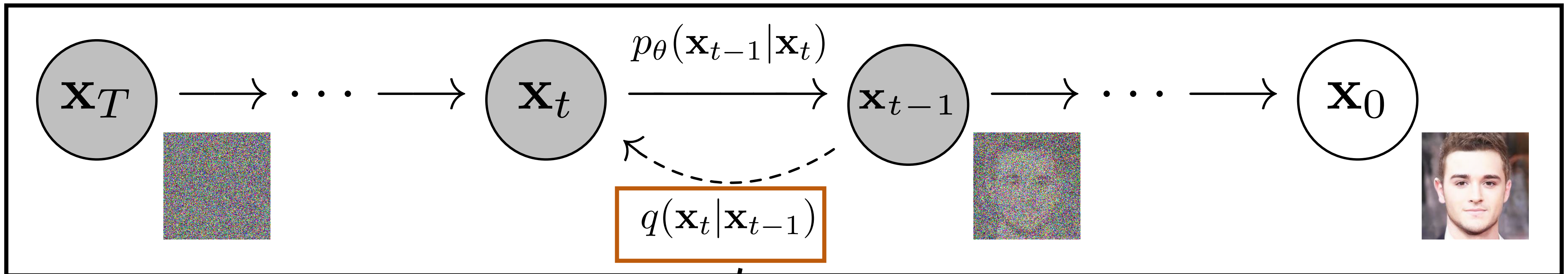
Forward
(Data → Noise)

$$q(\mathbf{x}_{1:T}|\mathbf{x}_0) := \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1})$$

Full chain

This is added in the training process

Diffusion Model



Core idea: Stepwise transition from pure noise to data

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

Forward
(Data → Noise)

$$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I})$$

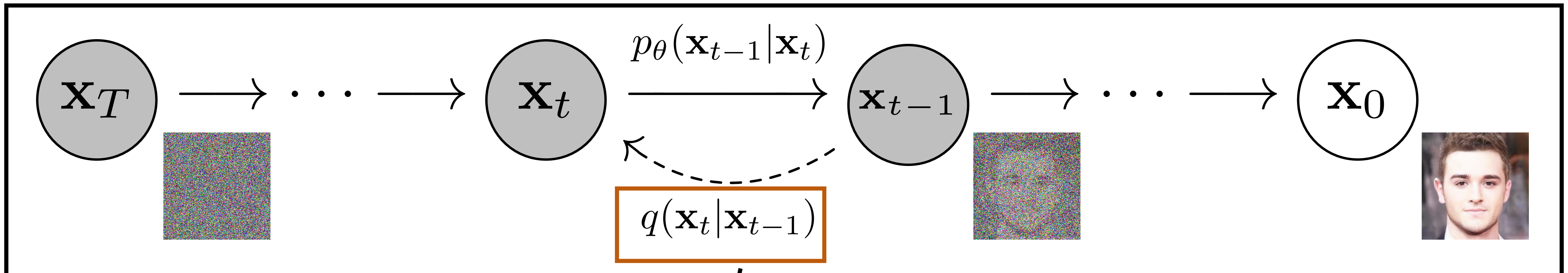
Shortcut:
transition to any time

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

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

This is added in the training process

Diffusion Model



Core idea: Stepwise transition from pure noise to data

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

Forward
(Data → Noise)

$$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I})$$

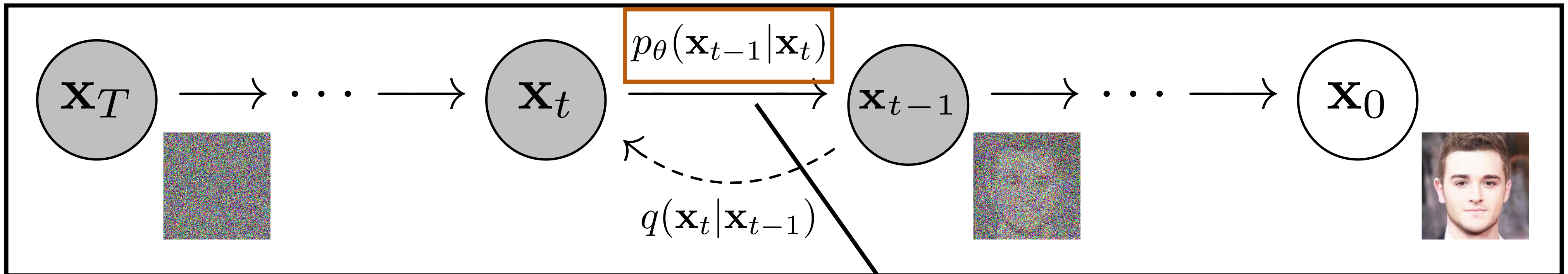
$$\mathbf{x}_t(\mathbf{x}_0, \boldsymbol{\epsilon}) = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boxed{\boldsymbol{\epsilon}} \text{ for } \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

Rewrite: State at any time

Will try to predict

This is added in the training process

Diffusion Model



Core idea: Stepwise transition from pure noise to data

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) := \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t))$$

To be learned (in principle)

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

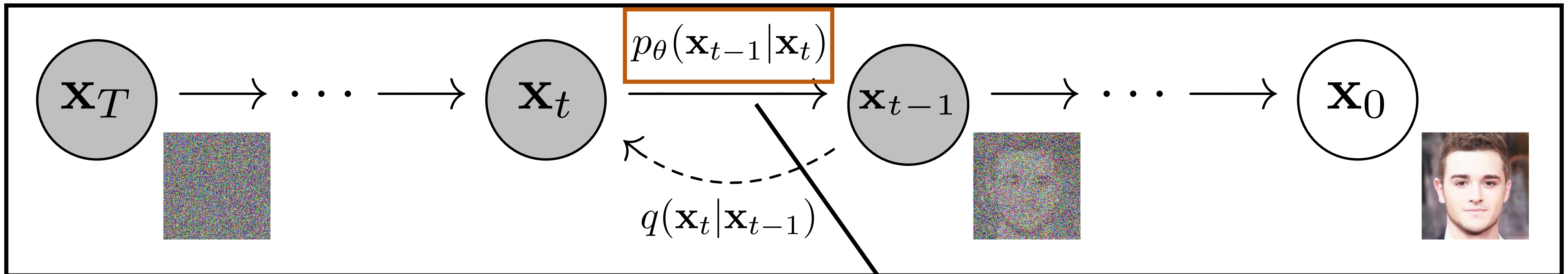
(hyper-parameter)

Reverse (Noise → data)

$$p_\theta(\mathbf{x}_{0:T}) := p(\mathbf{x}_T)\prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$$

$$p(\mathbf{x}_T) = \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I})$$

Diffusion Model



Core idea: Stepwise transition from pure noise to data

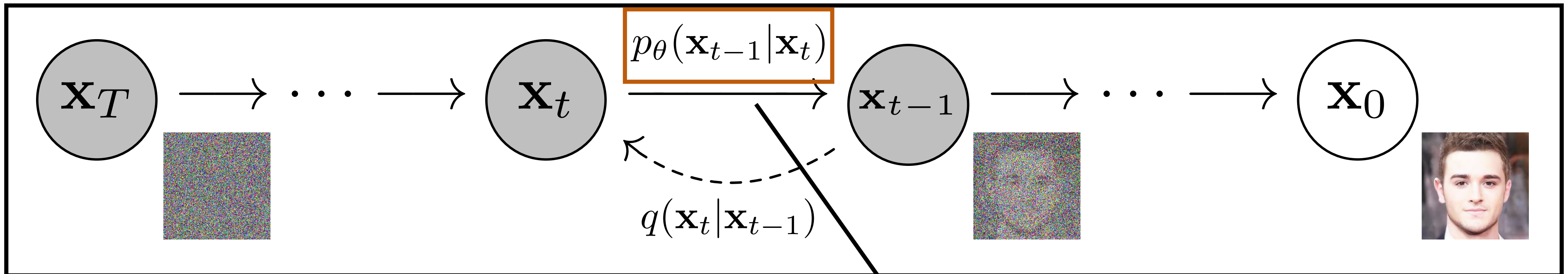
$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) := \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t))$$

Reverse
(Noise → data)

$$\frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon\right)$$

Optimal expected mean (take into account known noise schedule)

Diffusion Model



Core idea: Stepwise transition from pure noise to data

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) := \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t))$$

Reverse
(Noise → data)

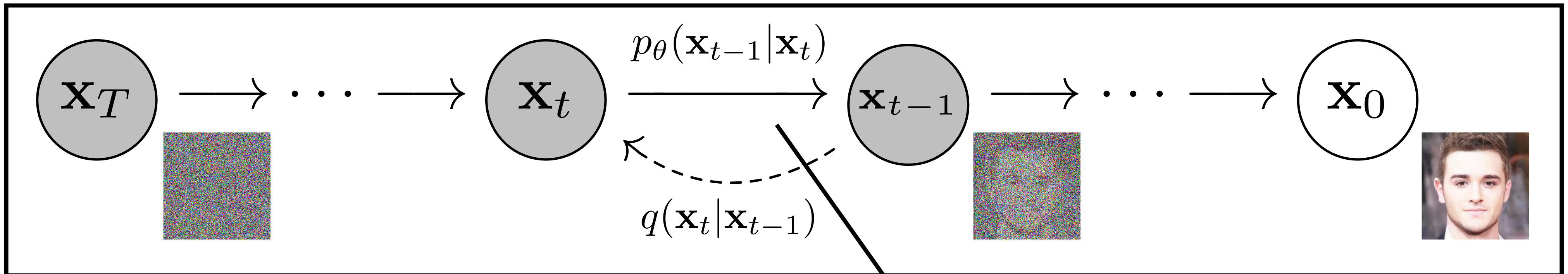
$$\frac{1}{\sqrt{\alpha_t}}(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon)$$

Use to parametrise function:

$$\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}}(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\boxed{\epsilon_\theta(x_t, t)}))$$

and learn to predict noise

Diffusion Model



Core idea: Stepwise transition from pure noise to data

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) := \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t))$$

Reverse (Noise → data)

Resulting learning objective

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

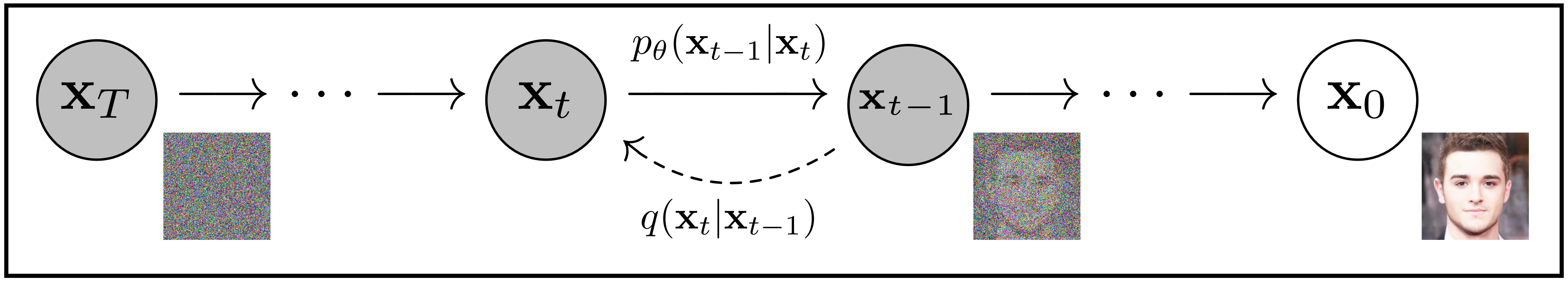
Noisy image

Reminder: Forward diffusion to time t

$$\mathbf{x}_t(\mathbf{x}_0, \boldsymbol{\epsilon}) = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}$$

Timestep

Diffusion Model

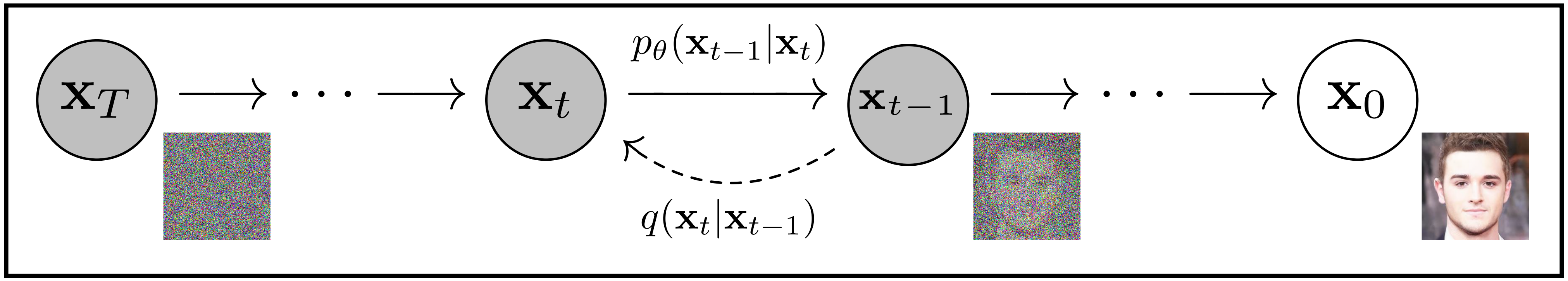


Core idea: Stepwise transition
from pure noise to data

Algorithm 1 Training

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

Diffusion Model



Core idea: Stepwise transition from pure noise to data

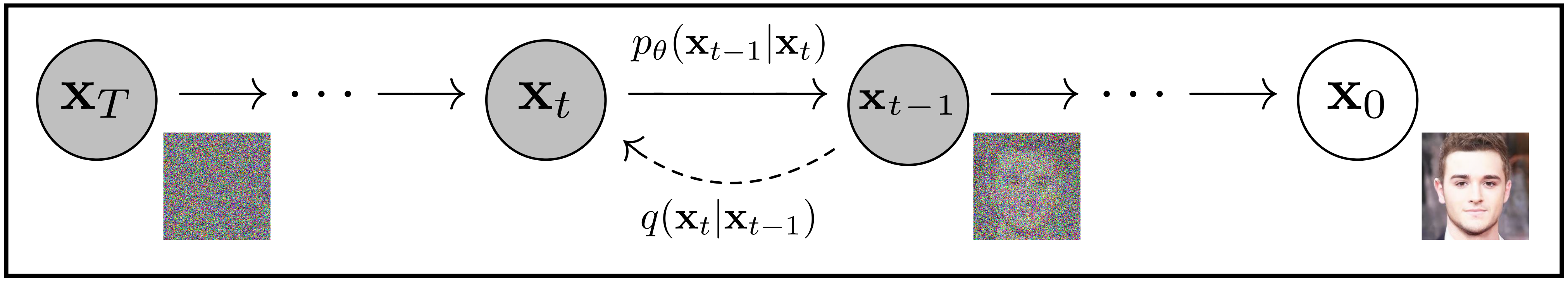
Algorithm 1 Training

repeat
  $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
  $t \sim \mathrm{Uniform}(\{1, \ldots, T\})$
  $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  Take gradient descent step on
  $$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$
until converged

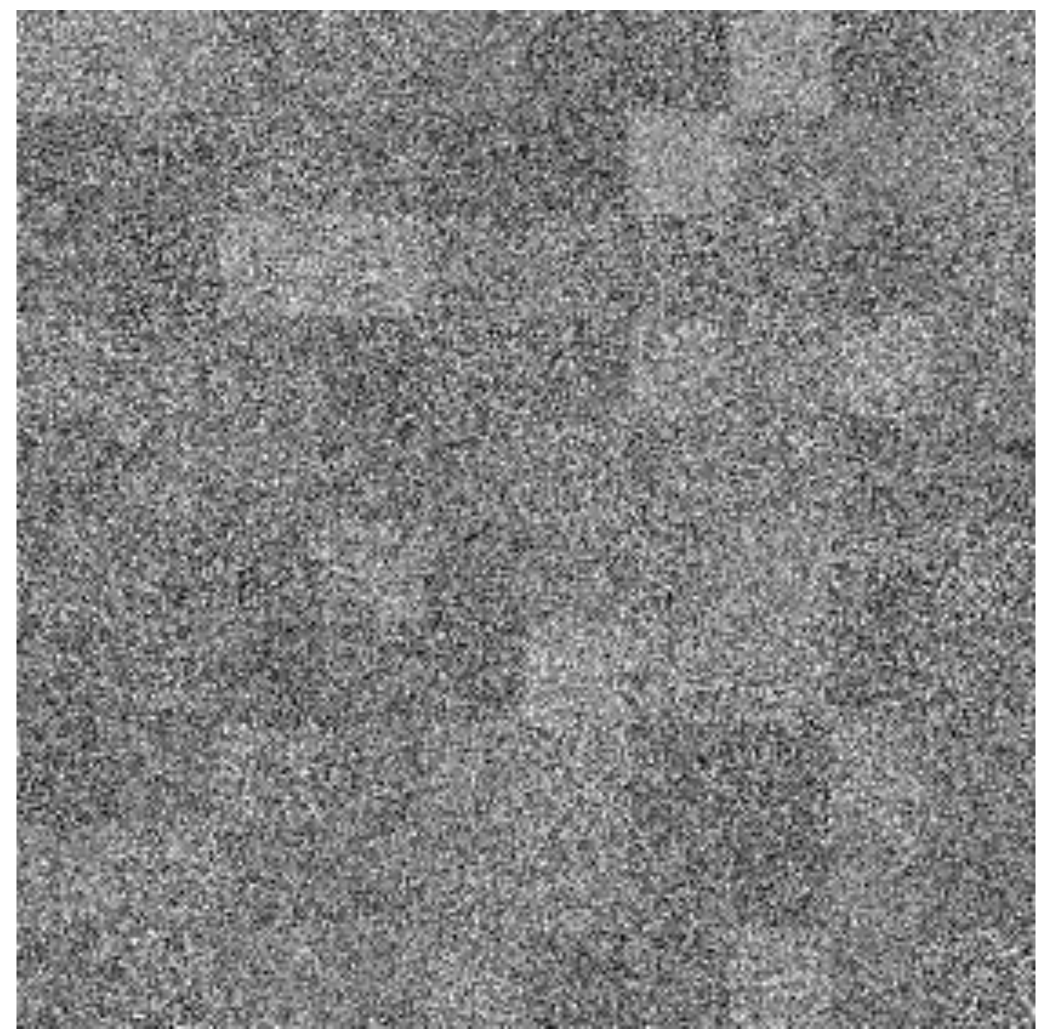
Algorithm 2 Sampling

$\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
for $t = T, \ldots, 1$ **do**
  $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
  $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right) + \sigma_t \mathbf{z}$
end for
return $\mathbf{x}_0$

Diffusion Model

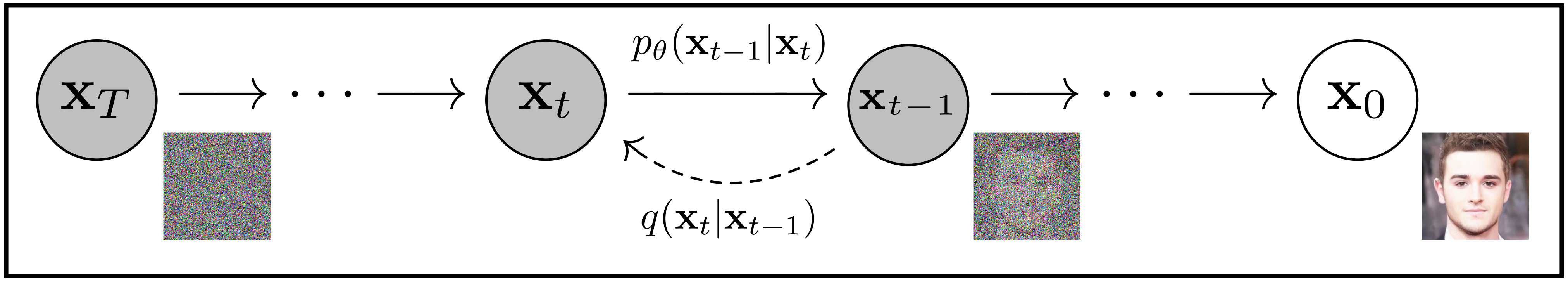


Core idea: Stepwise transition from pure noise to data

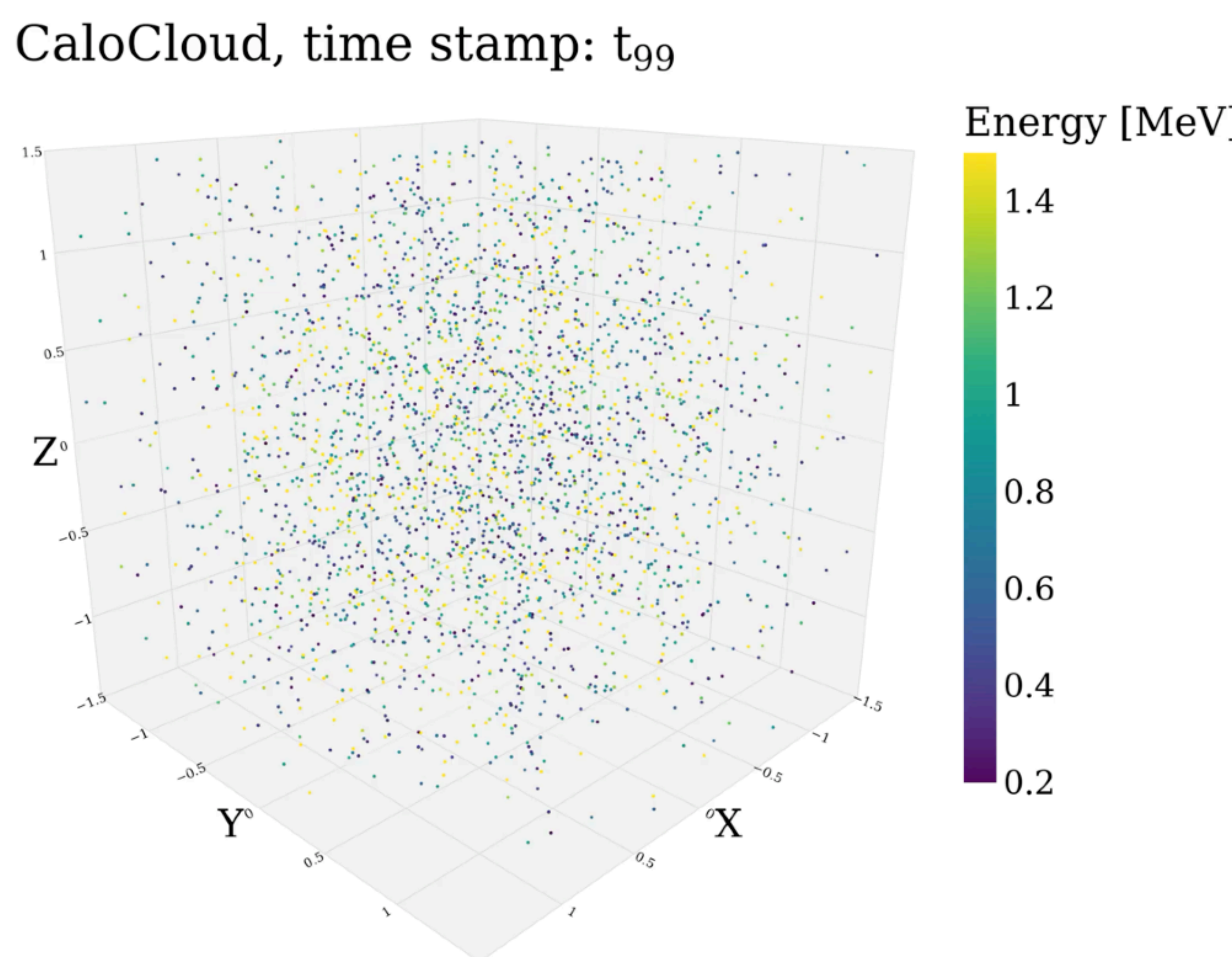


https://medium.com/mlearning-ai/enerating-images-with-ddpms-a-pytorch-implementation-cef5a2ba8cb1

Diffusion Model

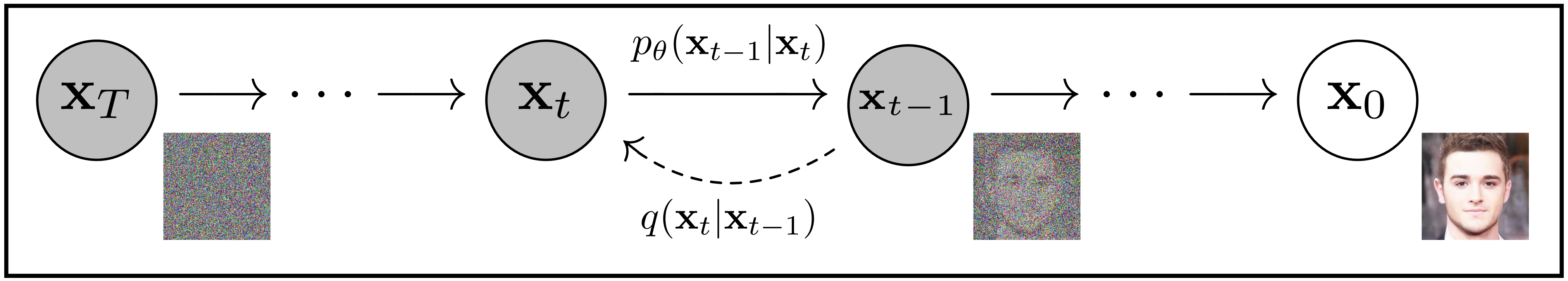


Core idea: Stepwise transition from pure noise to data

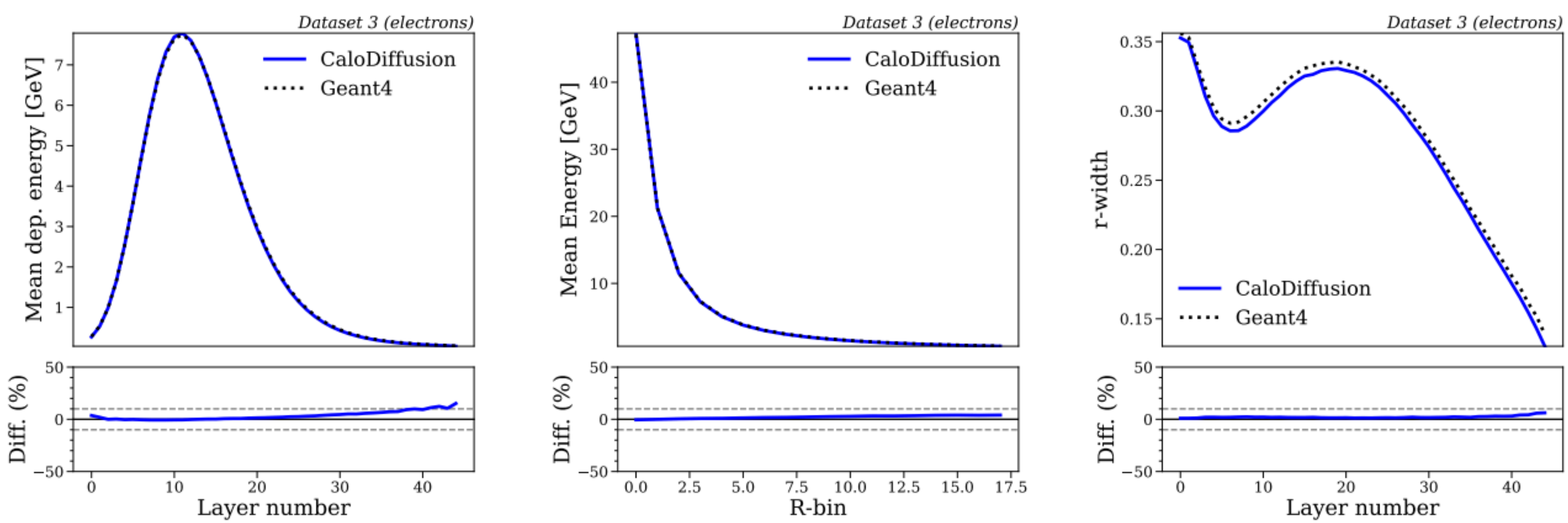


2305.04847

Diffusion Model

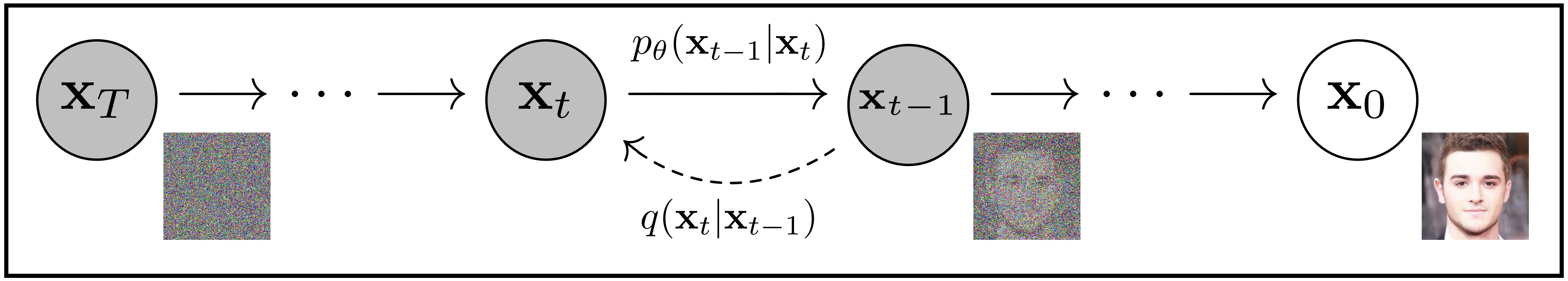


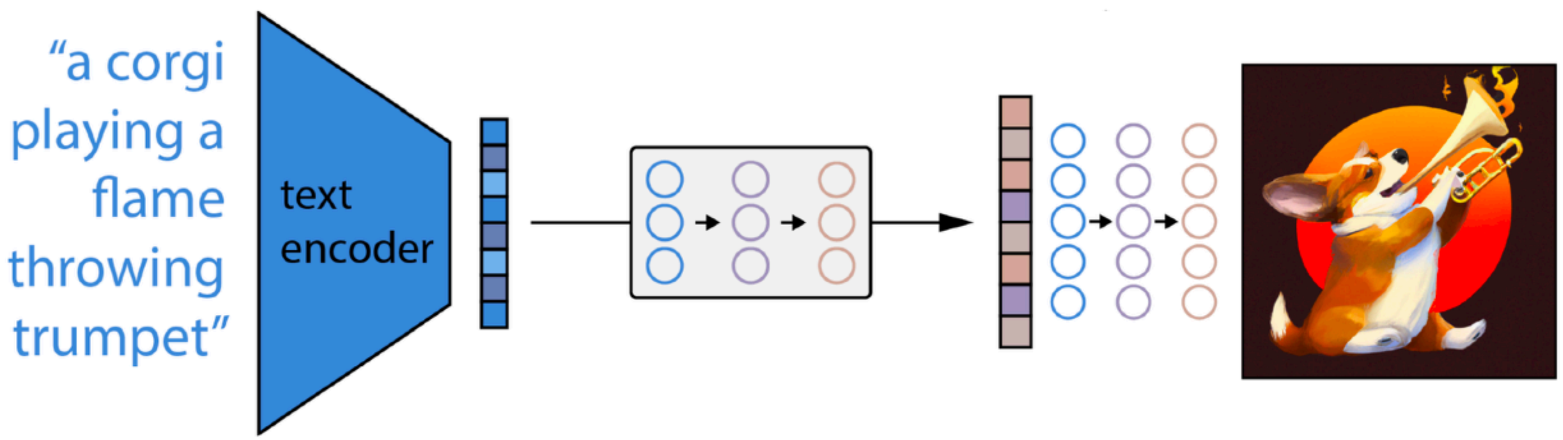
Core idea: Stepwise transition
from pure noise to data



2308.03876

Diffusion Model



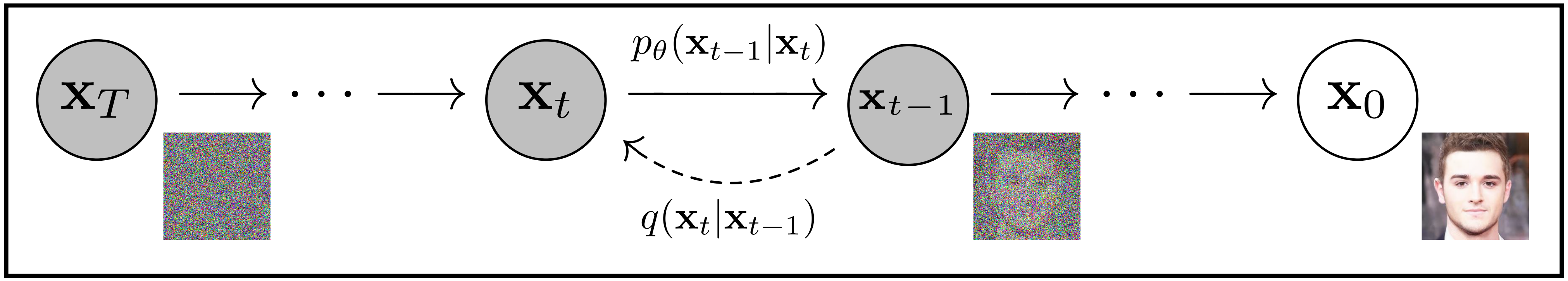


DALLE-2 (Roughly)

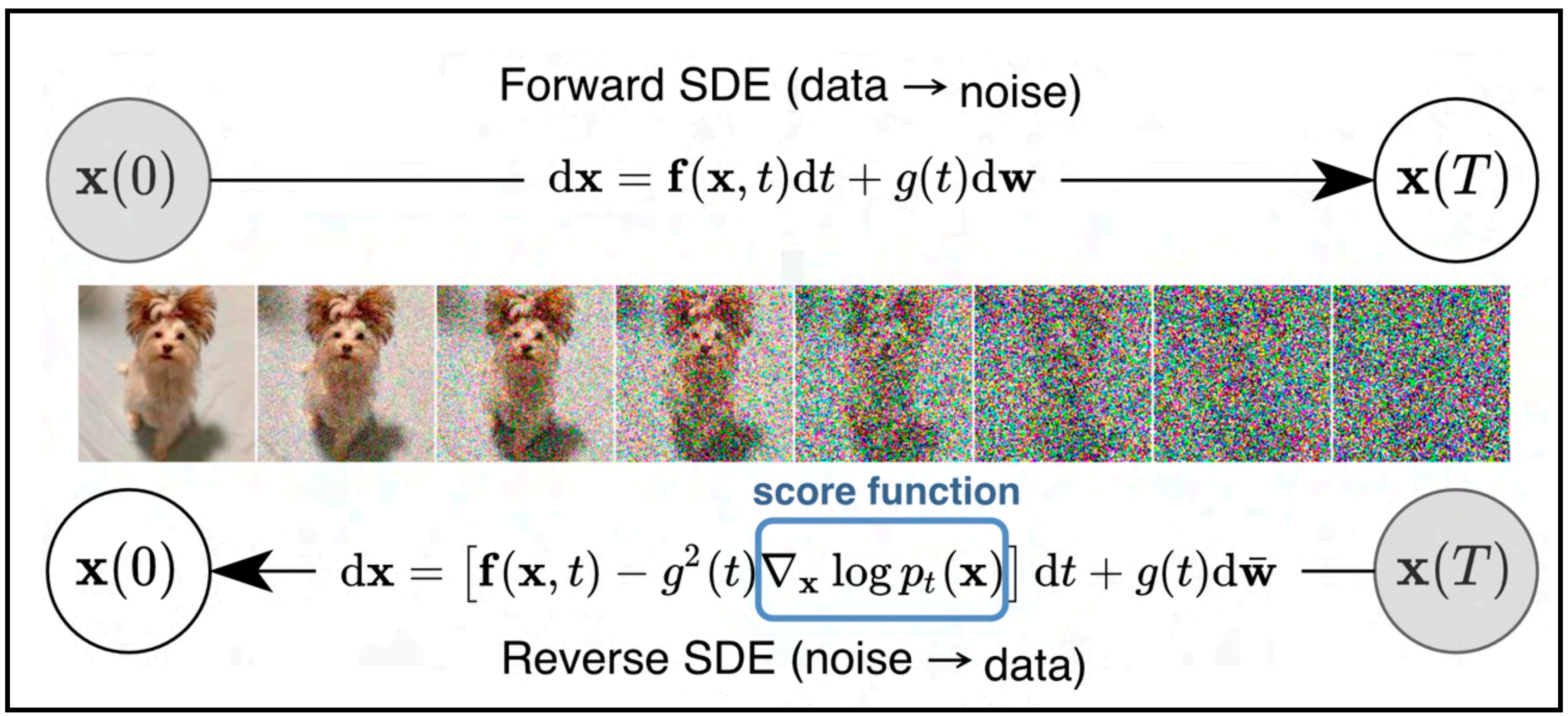
Contrastive learning of joint image/text embedding +
translation of embedding +
conditioned diffusion

https://openai.com/dall-e-3; https://www.assemblyai.com/blog/how-dall-e-2-actually-works/

Continuos Diffusion Model



Replace discrete noise steps



with stochastic differential equations (SDEs)

Continuos Diffusion Model

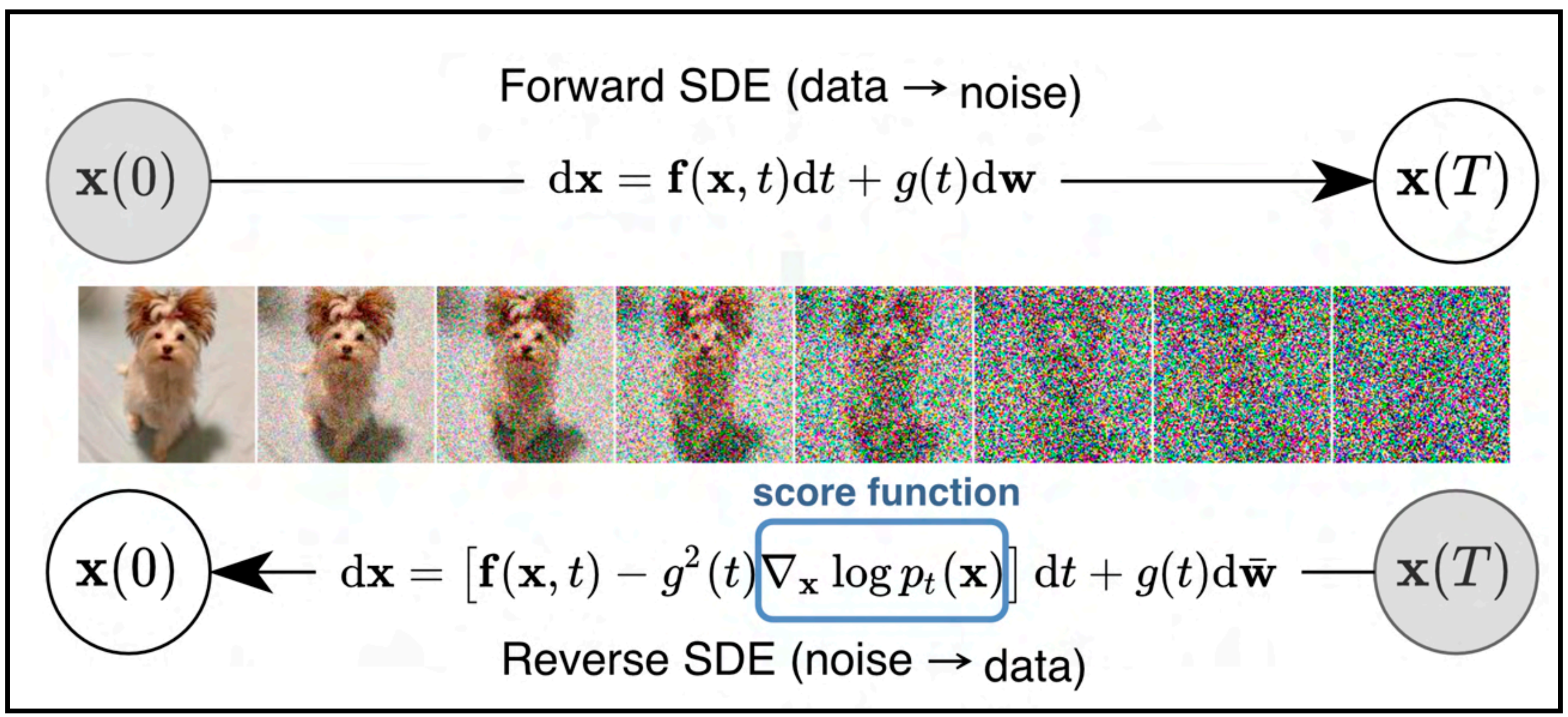
Forward SDE: $$\mathrm{d}\mathbf{x} = \mathbf{f}(\mathbf{x}, t)\mathrm{d}t + g(t)\mathrm{d}\mathbf{w}$$

Drift term

Diffusion term

Wiener process/
Brownian motion
(independent
Gaussian
increments)

(Correspond to the noise schedule
in discrete case)



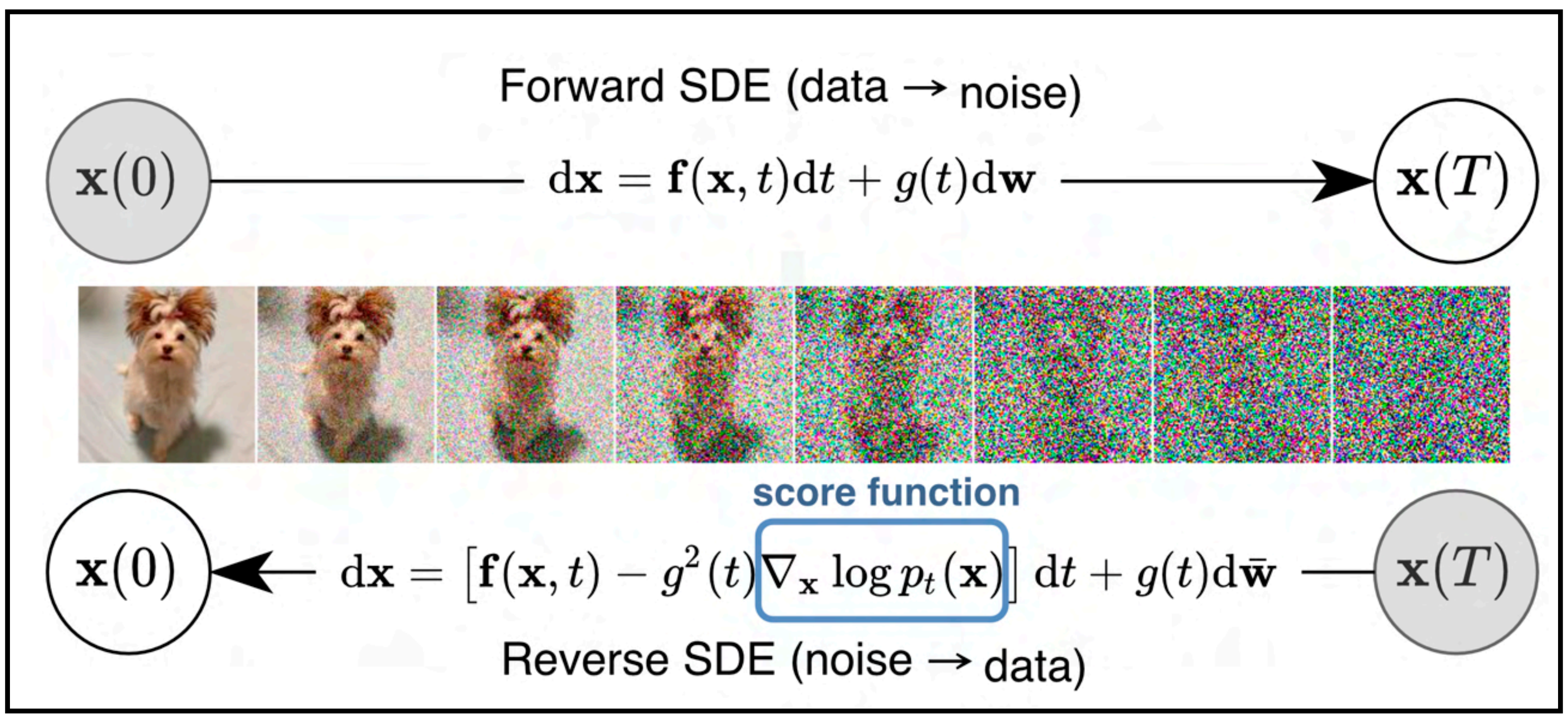
Continuos Diffusion Model

Probability density of x(t)

Reverse SDE: $$\mathrm{d}\mathbf{x} = [\mathbf{f}(\mathbf{x}, t) - g(t)^2 \nabla_{\mathbf{x}} \log p_t(\mathbf{x})]\mathrm{d}t + g(t)\mathrm{d}\bar{\mathbf{w}}$$

Score function

Reverse of a diffusion process is also a diffusion

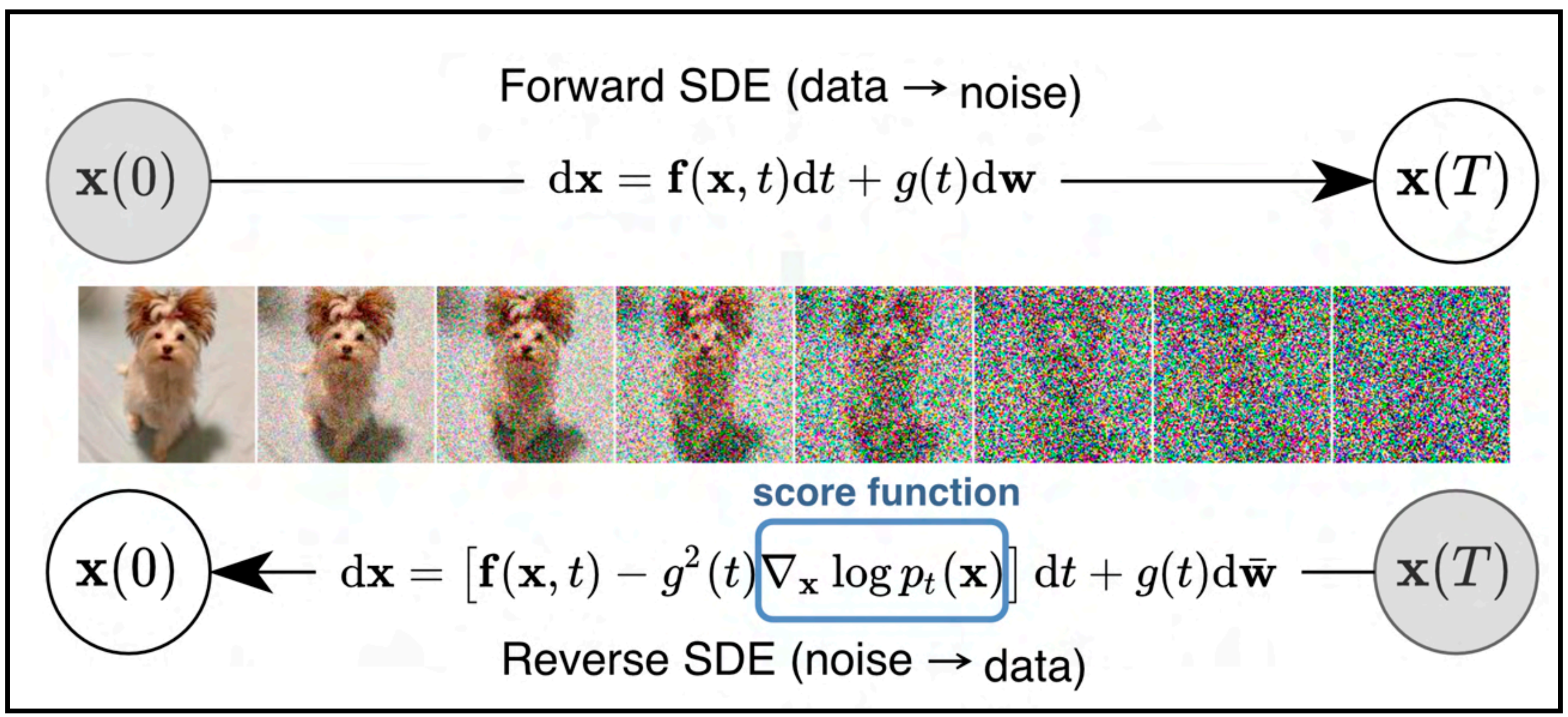


Continuos Diffusion Model

Reverse SDE: $$\mathrm{d}\mathbf{x} = [\mathbf{f}(\mathbf{x},t) - g(t)^2 \boxed{\nabla_{\mathbf{x}} \log p_t(\mathbf{x})}]\mathrm{d}t + g(t)\mathrm{d}\bar{\mathbf{w}}$$

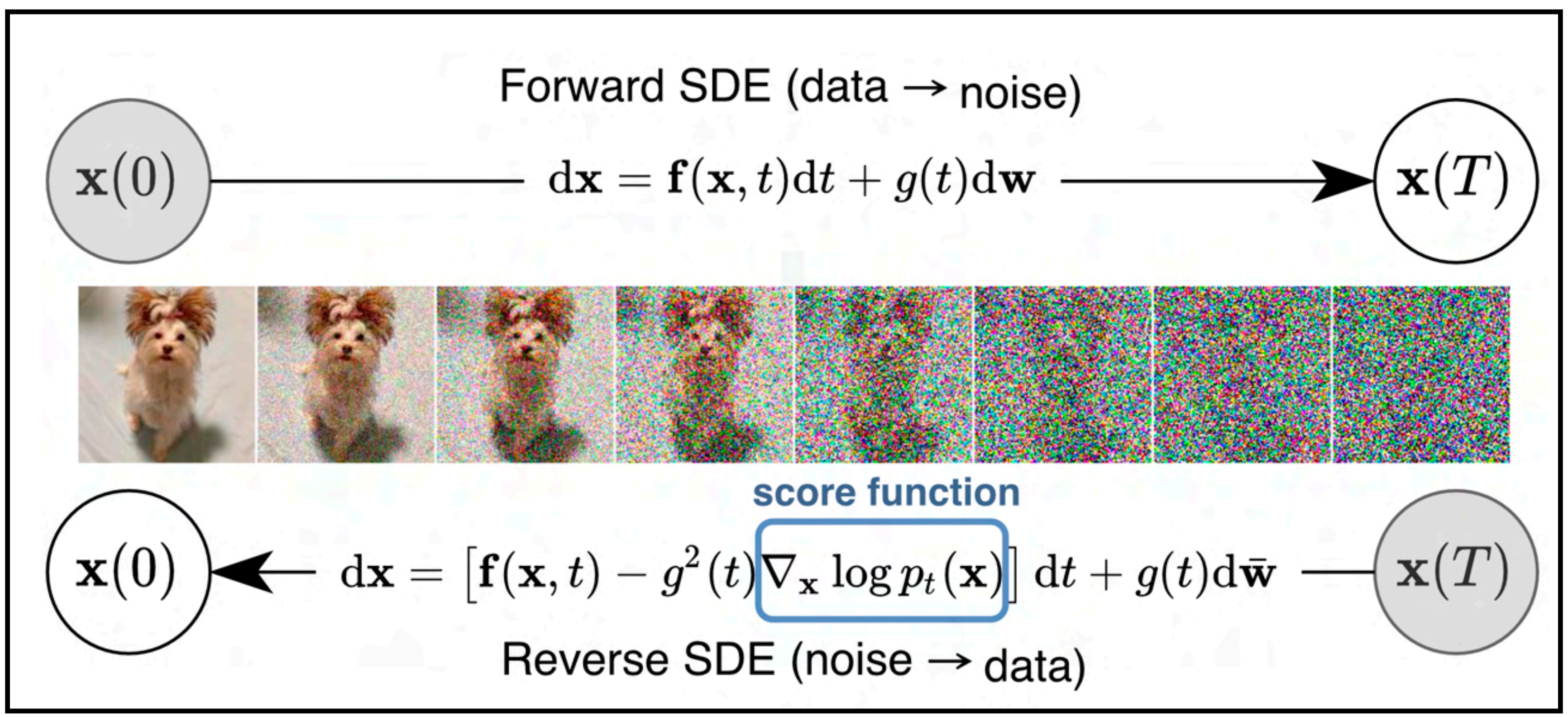
$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathbb{E}_t \Big\{ \lambda(t) \mathbb{E}_{\mathbf{x}(0)} \mathbb{E}_{\mathbf{x}(t)|\mathbf{x}(0)} \big[\big\| \mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}(t),t) - \boxed{\nabla_{\mathbf{x}(t)} \log p_{0t}(\mathbf{x}(t) \mid \mathbf{x}(0))} \big\|_2^2 \big] \Big\}$$

Learn to approximate score function with neural network

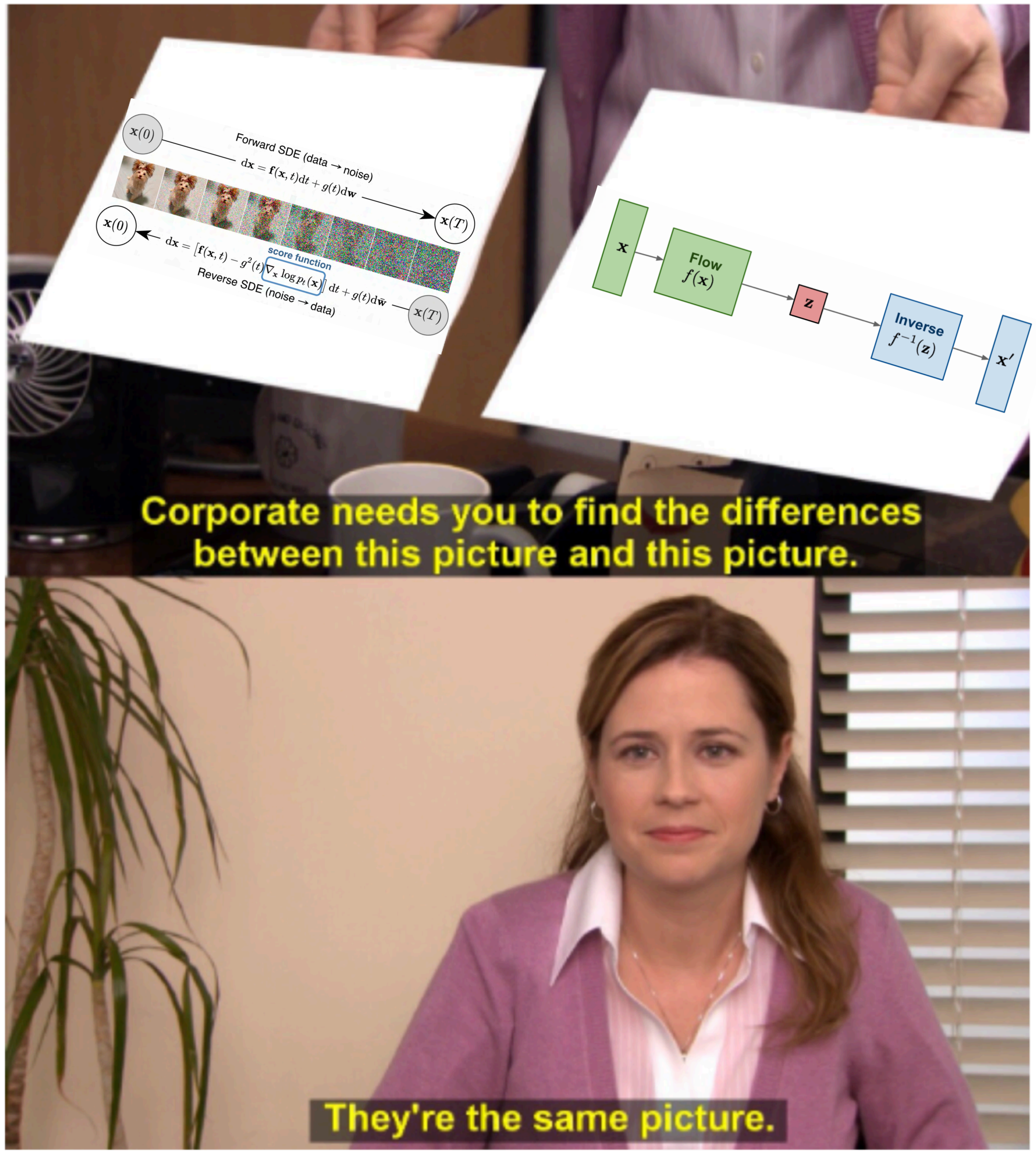


Continuos Diffusion Model

Once trained: Sample latent space and numerically solve SDE to transport to data space



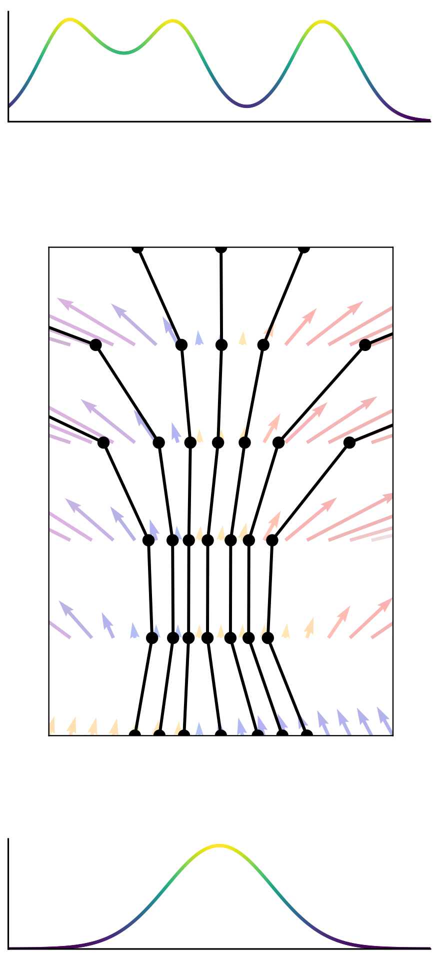
Aside

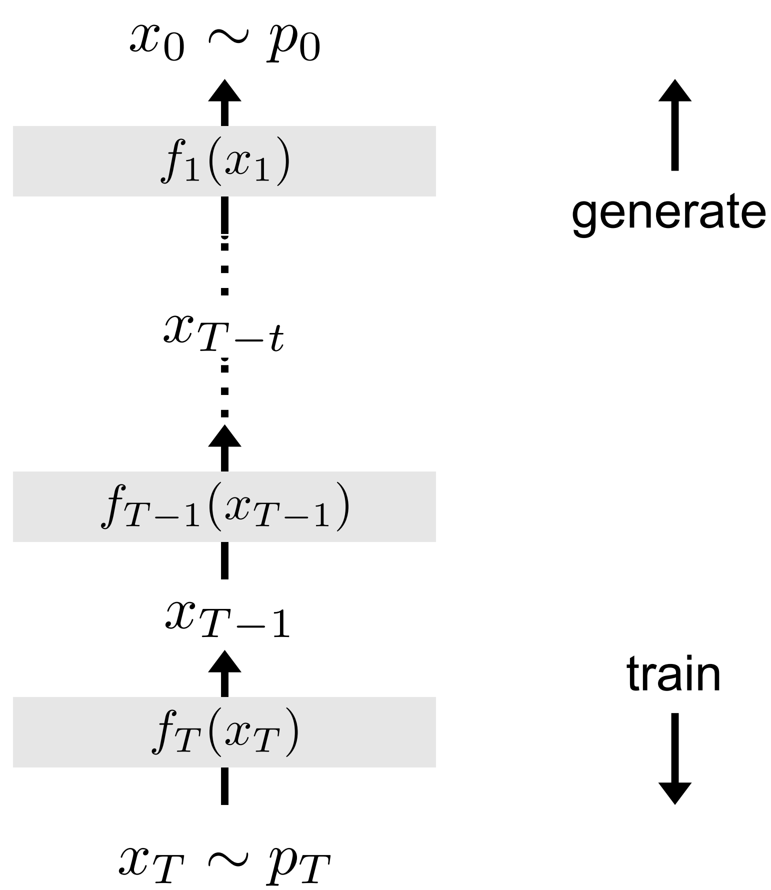


Well, almost

Flow Matching

Remember: Normalising Flows



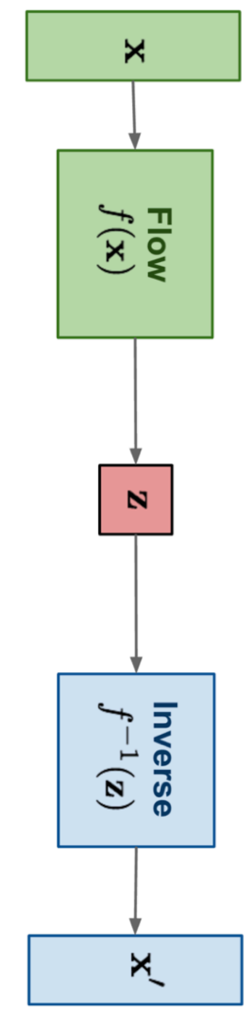


discrete normalizing flow

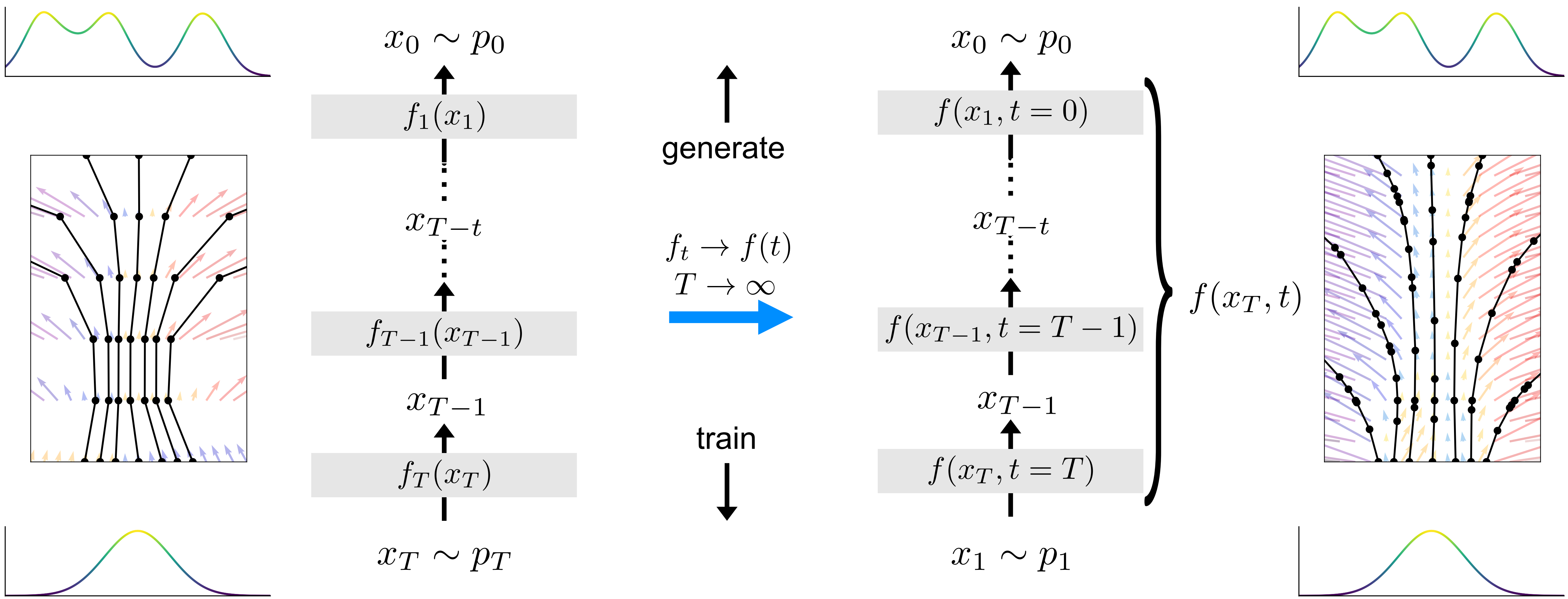
$$x_0 = f_1^{\theta_1} \ldots \circ f_{T-1}^{\theta_{T-1}} \circ f_T^{\theta_T}(x_T)$$

$$\log p_0(x_0) = \log p_T(x_T) - \sum_t \log |\det J_t|$$

Can view each layer in the flow as a discrete time step



Remember: Normalising Flows



discrete normalizing flow

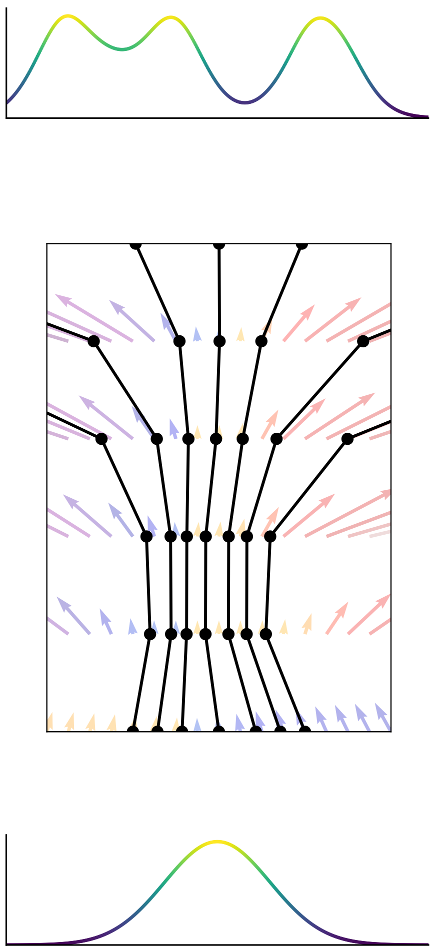
$$x_0 = f_1^{\theta_1} \ldots \circ f_{T-1}^{\theta_{T-1}} \circ f_T^{\theta_T}(x_T)$$

continuous normalizing flow

$$\boxed{\frac{dx_t}{dt} = v_t^\theta(x_t)}, \quad x_t := f(x_T, t)$$

Take continuous limit:
Approximate vector field with network

Remember: Normalising Flows



$x_0 \sim p_0$

$f_1(x_1)$

x_{T-t}

$f_{T-1}(x_{T-1})$

x_{T-1}

$f_T(x_T)$

$x_T \sim p_T$

generate

$f_t \to f(t)$
$T \to \infty$

train

$x_0 \sim p_0$

$f(x_1, t = 0)$

x_{T-t}

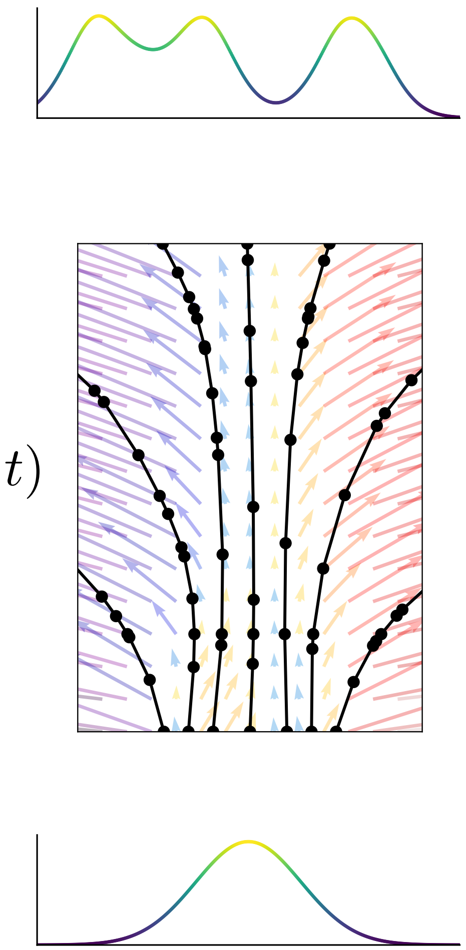
$f(x_{T-1}, t = T-1)$

x_{T-1}

$f(x_T, t = T)$

$x_1 \sim p_1$

$f(x_T, t)$



discrete normalizing flow

$$x_0 = f_1^{\theta_1} \ldots \circ f_{T-1}^{\theta_{T-1}} \circ f_T^{\theta_T}(x_T)$$

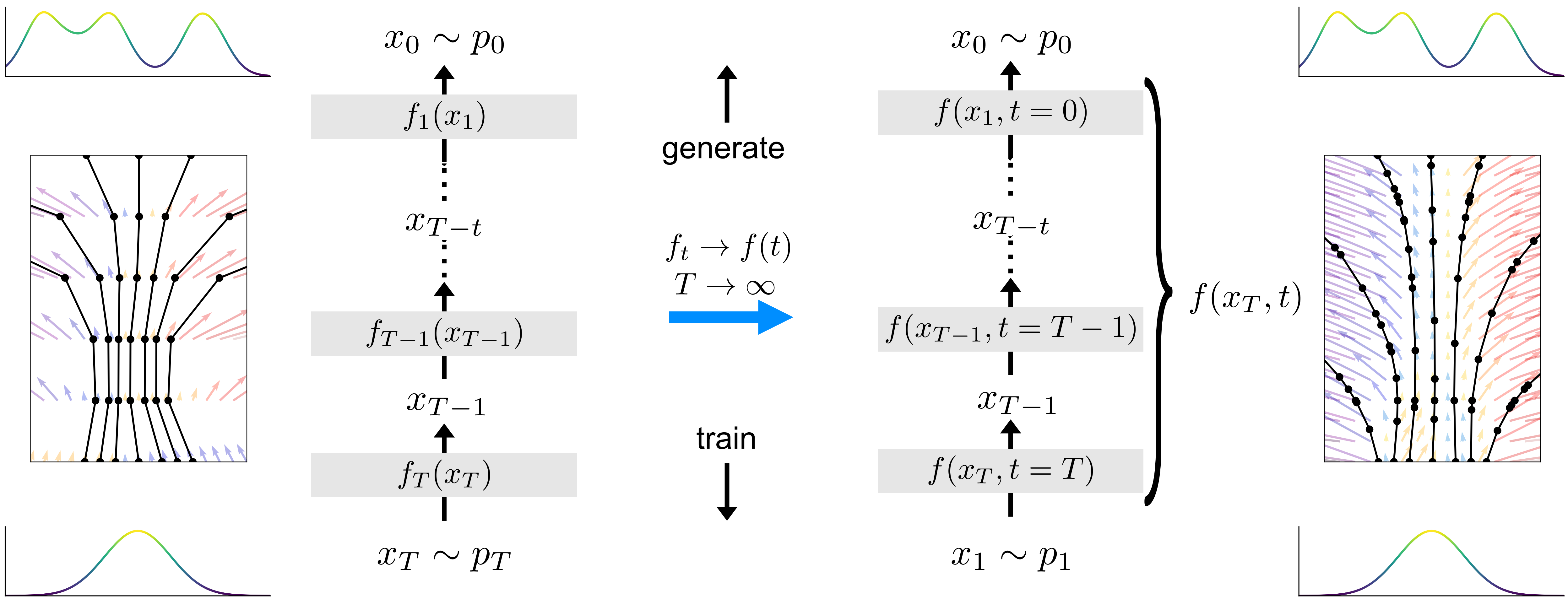
continuous normalizing flow

$$\frac{dx_t}{dt} = v_t^{\theta}(x_t), \quad x_t := f(x_T, t)$$

Change in loss:

$$\log p_0(x_0) = \log p_T(x_T) - \sum_t \log |\det J_t| \longrightarrow \log p_0(x_0) = \log p_1(x_1) - \int_{t_1}^{t_0} \mathrm{tr}\left(\frac{\partial u_t}{\partial x_t}\right) dt$$

Remember: Normalising Flows



discrete normalizing flow

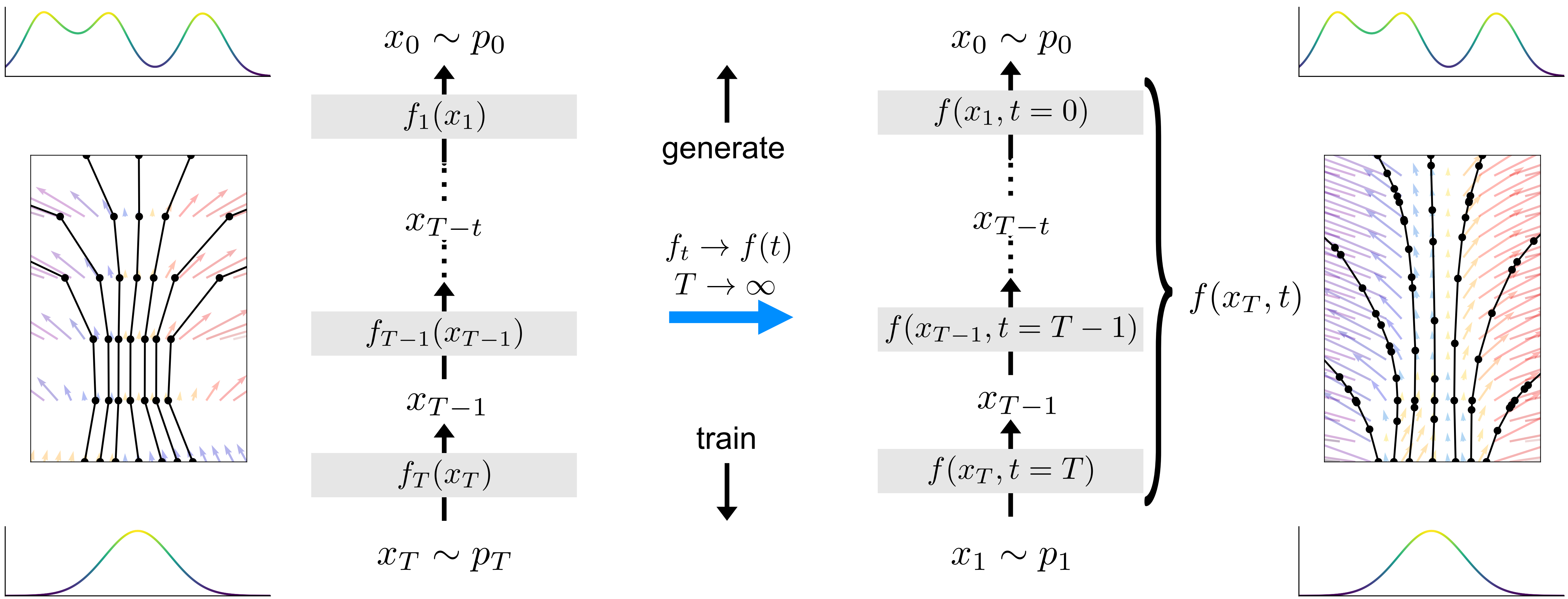
$$x_0 = f_1^{\theta_1} \ldots \circ f_{T-1}^{\theta_{T-1}} \circ f_T^{\theta_T}(x_T)$$

continuous normalizing flow

$$\frac{dx_t}{dt} = v_t^\theta(x_t), \quad x_t := f(x_T, t)$$

For sampling:
Solve differential equation (ODE)

Remember: Normalising Flows



discrete normalizing flow

$$x_0 = f_1^{\theta_1} \ldots \circ f_{T-1}^{\theta_{T-1}} \circ f_T^{\theta_T}(x_T)$$

continuous normalizing flow

$$\frac{dx_t}{dt} = v_t^\theta(x_t), \quad x_t := f(x_T, t)$$

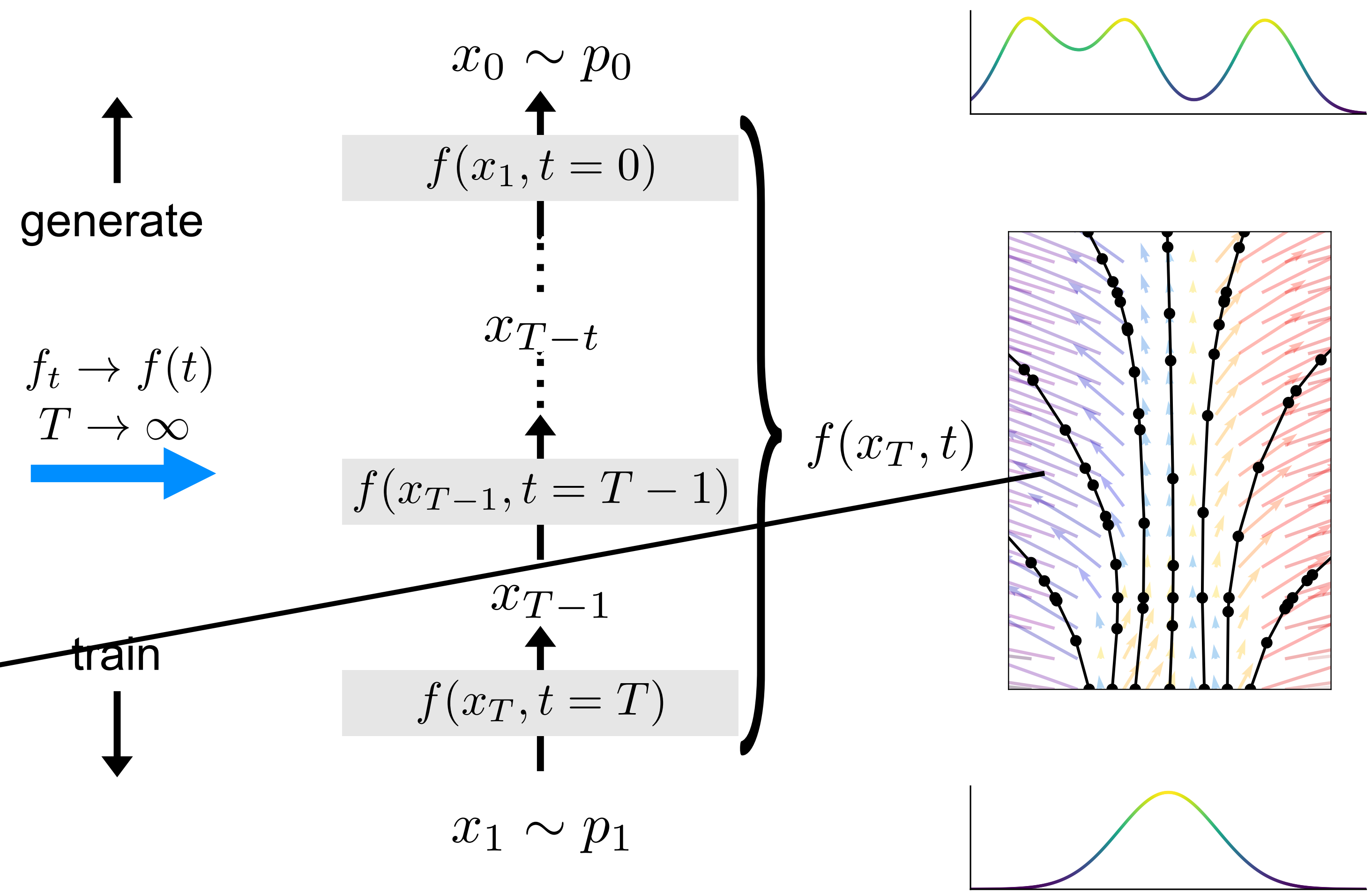
Training costly:
Solve ODE to evaluate p(x)
Calculate trace for change in probability volume

Flow Matching

$$\mathcal{L}_{\text{FM}} = \left\| v_t(x_t) - u_t(x_t) \right\|^2$$

Neural network

Vector field describing morphing



Training costly:
Solve ODE to evaluate p(x)
Calculate trace for change in probability volume
Instead, learn to approximate target vector field

Flow Matching

How to get the vector field describing morphing?

Constrains:
At t=0: Input data
At t=1: Target latent distribution

Use mixture of conditional morphings

$$\underbrace{p_t(x)}_{\text{Marginal path}} = \int dx_0 \, \underbrace{p_t(x|x_0)}_{\text{conditional path}} p_0(x_0)$$

$$u_t(x) = \int dx_0 \, u_t(x|x_0) \frac{p_t(x|x_0)p_0(x_0)}{p_t(x)}$$

Would allow calculating corresponding vector field

$$\mathcal{L}_{\text{FM}} = \left\| v_\theta(x_t|t) - u_t(x_t|x_0) \right\|^2$$

Still expensive, let network only approximate conditional vector field

2210.02747

Flow Matching

How to get the vector field describing morphing?

Constrains:
At t=0: Input data
At t=1: Target latent distribution

$$p_t(x|x_0) = \mathcal{N}(x|\gamma_t, \sigma_t)$$

Use mixture of conditional morphings

$$\gamma_t^{\text{FM}} = (1-t)x_0,$$

$$\sigma_t^{\text{FM}} = \sigma_{\min} + (1-\sigma_{\min})t$$

But what are the actual paths?
Can use Gaussians with linear interpolation

Vector field:

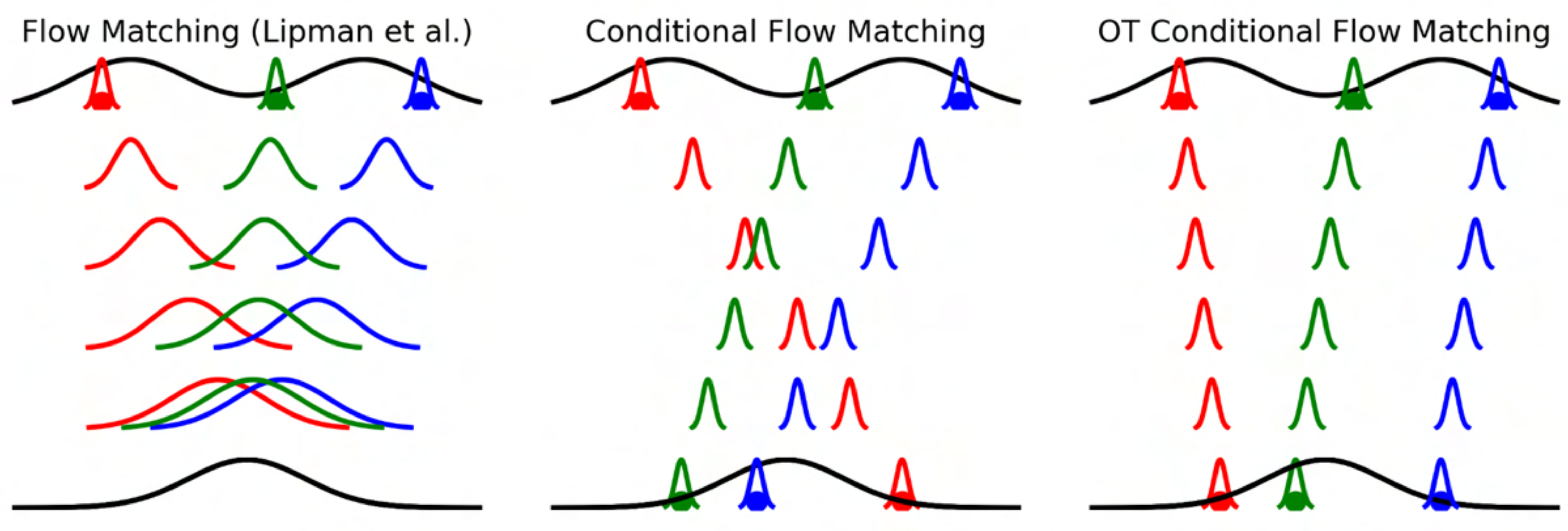
$$u_t(x_t|x_0) = \dot{\gamma}_t + \dot{\sigma}_t\epsilon = (1-\sigma_{\min})\epsilon - x_0$$

Loss:

$$\mathcal{L}_{\text{FM}} = \|(v_\theta(x_t, t) - (1-\sigma_{\min})\epsilon - x_0)\|^2$$

2210.02747

Optimal Transport



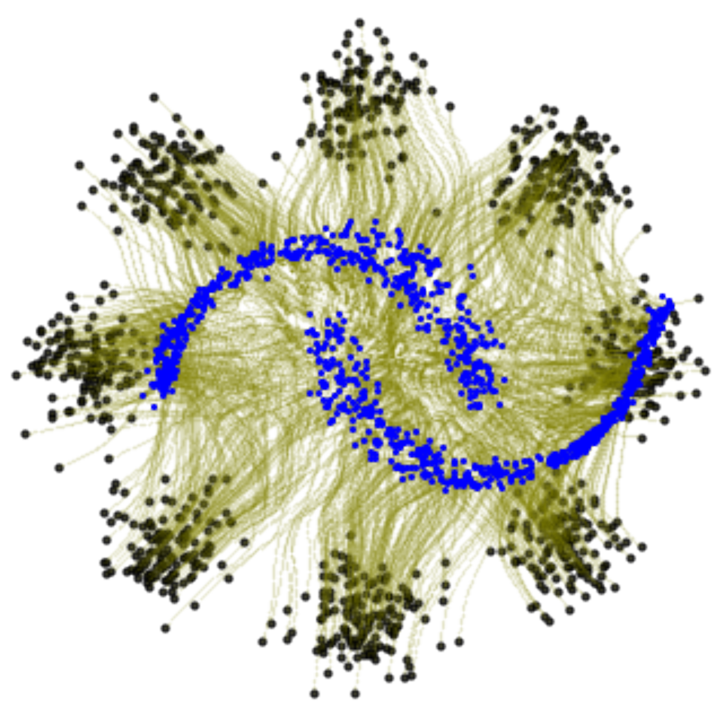
Can further improve paths by:

Conditioning on start- and endpoint

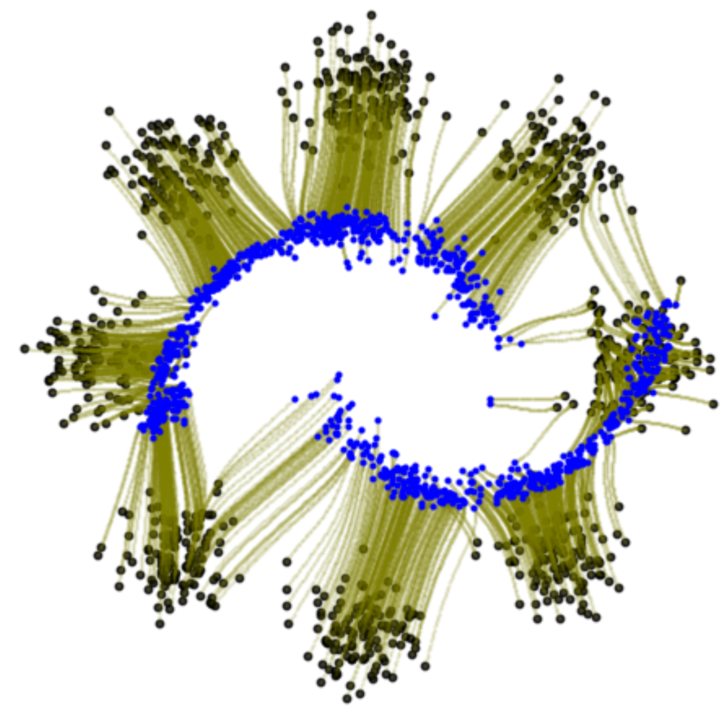
Adding optimal transport condition

2302.00482

Optimal Transport



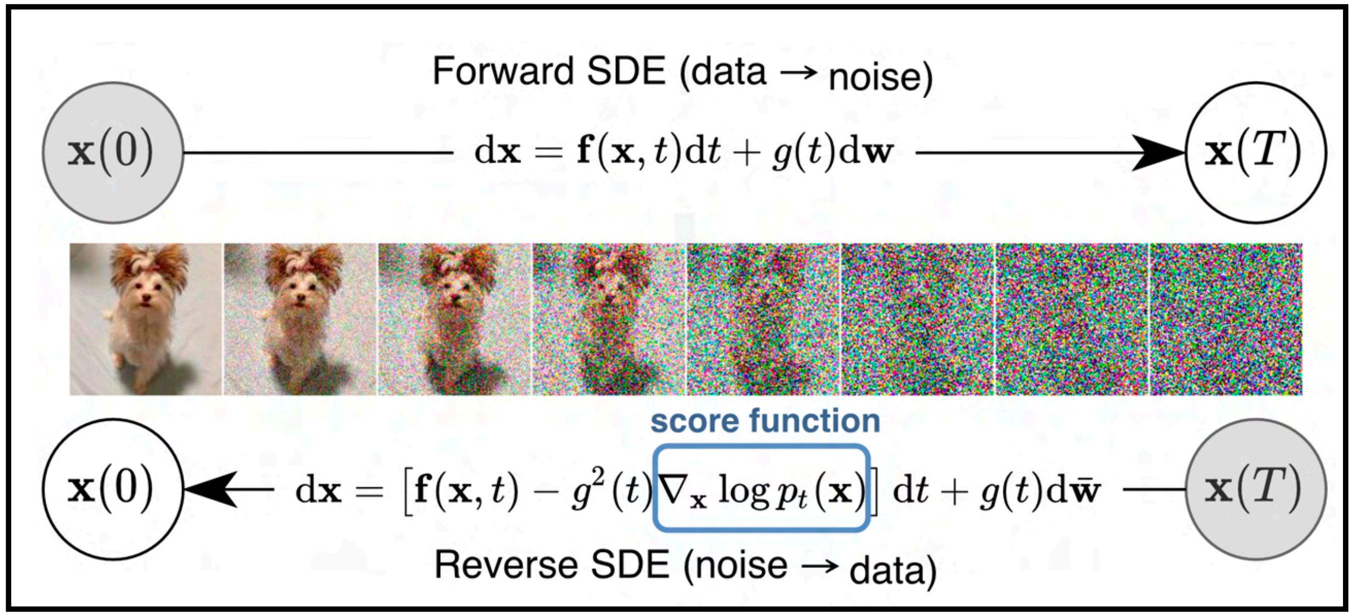
Conditioning on
start- and endpoint

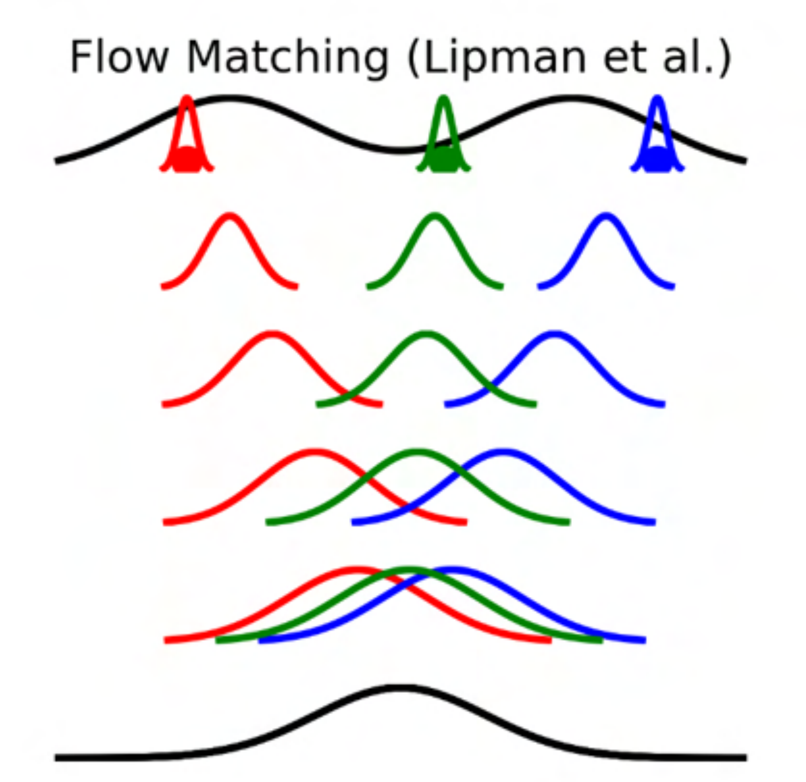


Adding optimal
transport condition

2302.00482

Comments





Score matching objective

Flow matching objective

Can show equivalence for Gaussian probability paths

Continuous Normalising Flow more general framework (other definition of paths)

2310.00049

Comments

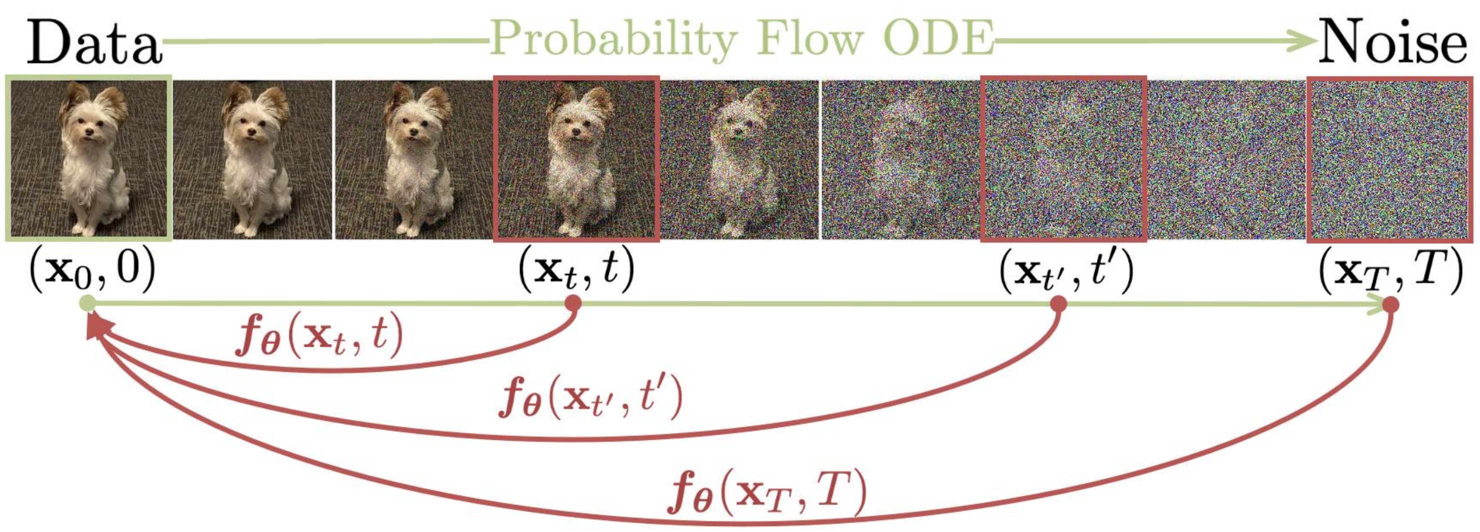
Close relation of CNF and diffusion models

Maturity:
Very recent, much in flux

Sample quality:
Very high

Training:
Stable

Sampling:
Expensive
(however: consistency distillation)



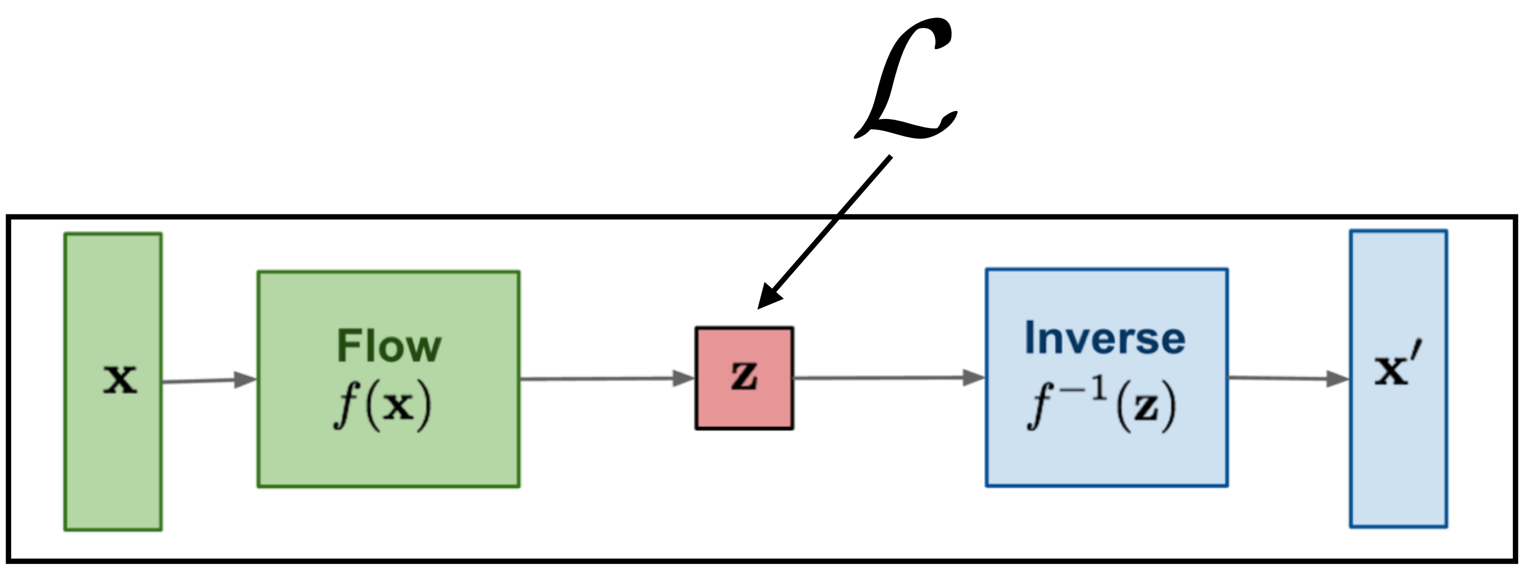
2303.01469

Applications II

Comments on Flows

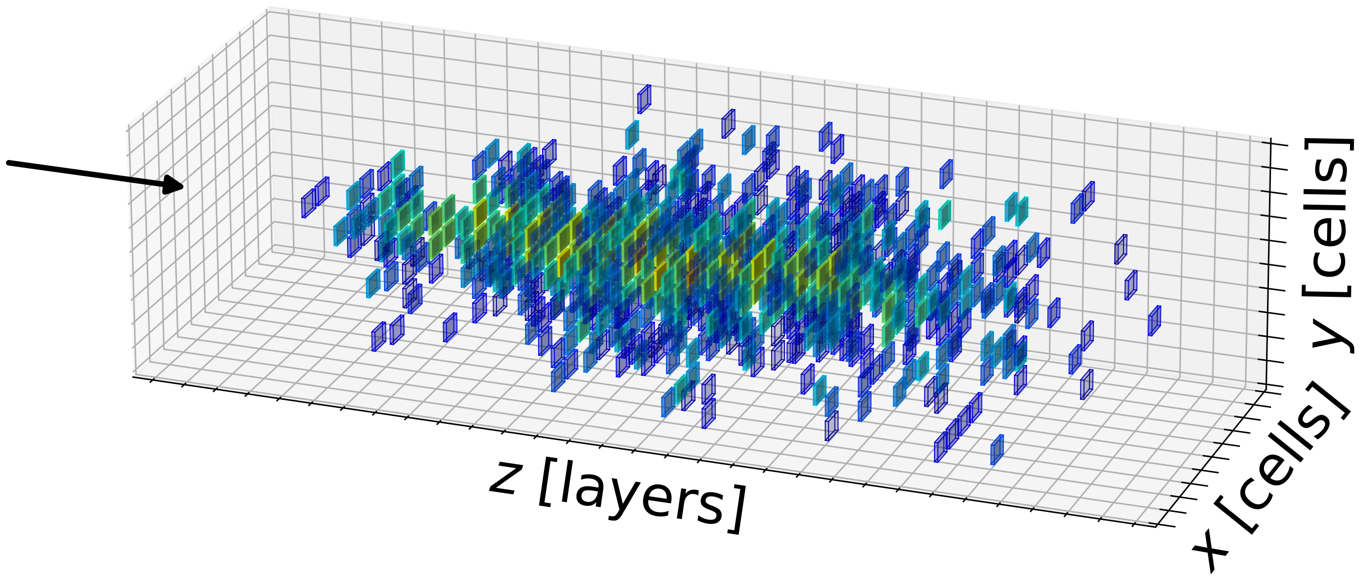
Exact learning of likelihood
→ Better generative fidelity
→ Can evaluate likelihood of data

More complex
→ Slower, choice of fast direction



1908.09257

Generative results II

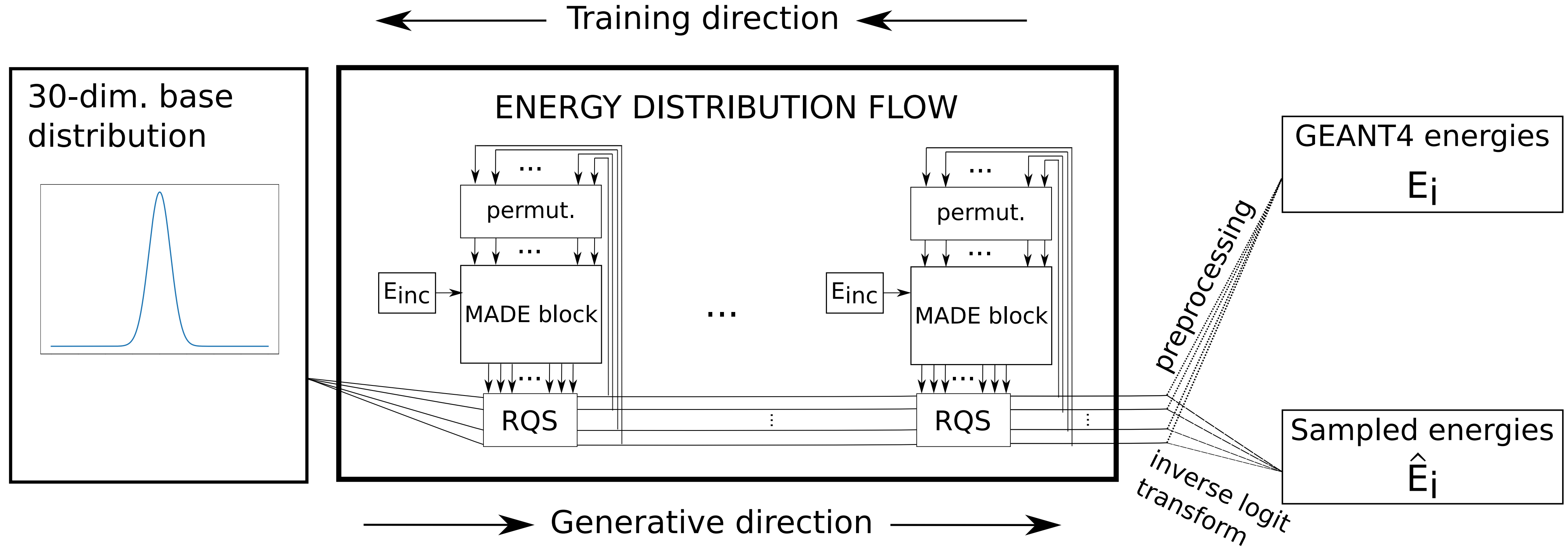


10x10 cells / layer
30 layers

How to flows for high-dimensional data?

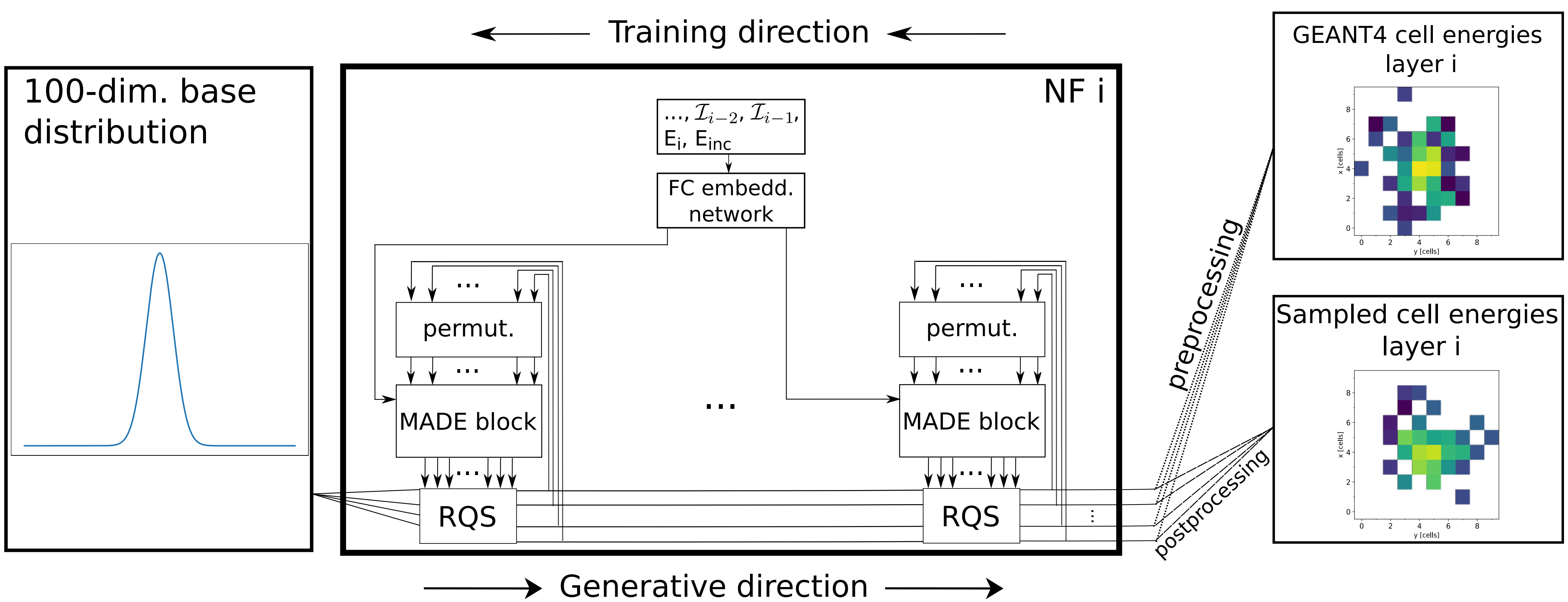
2302.11594

Generative results II



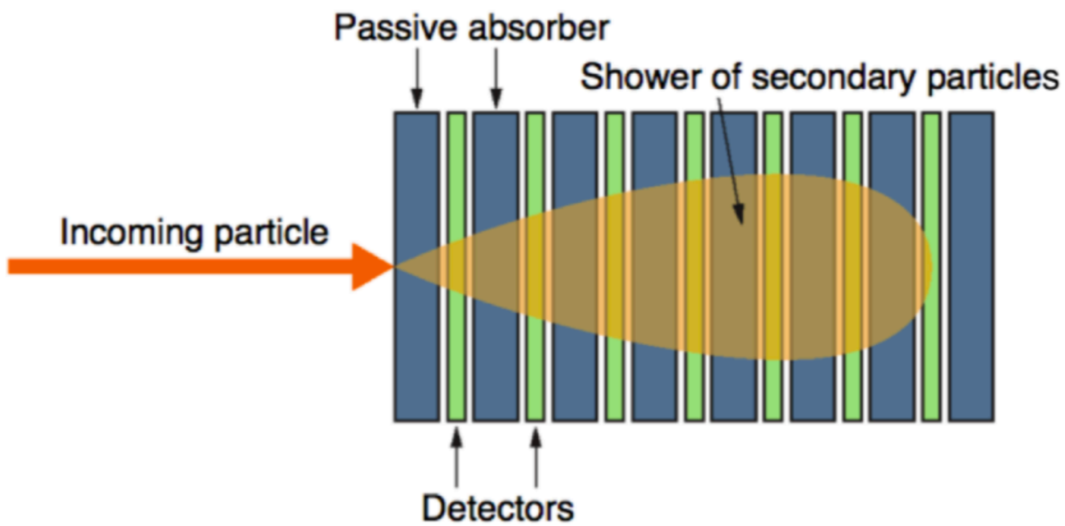
How to flows for high-dimensional data?

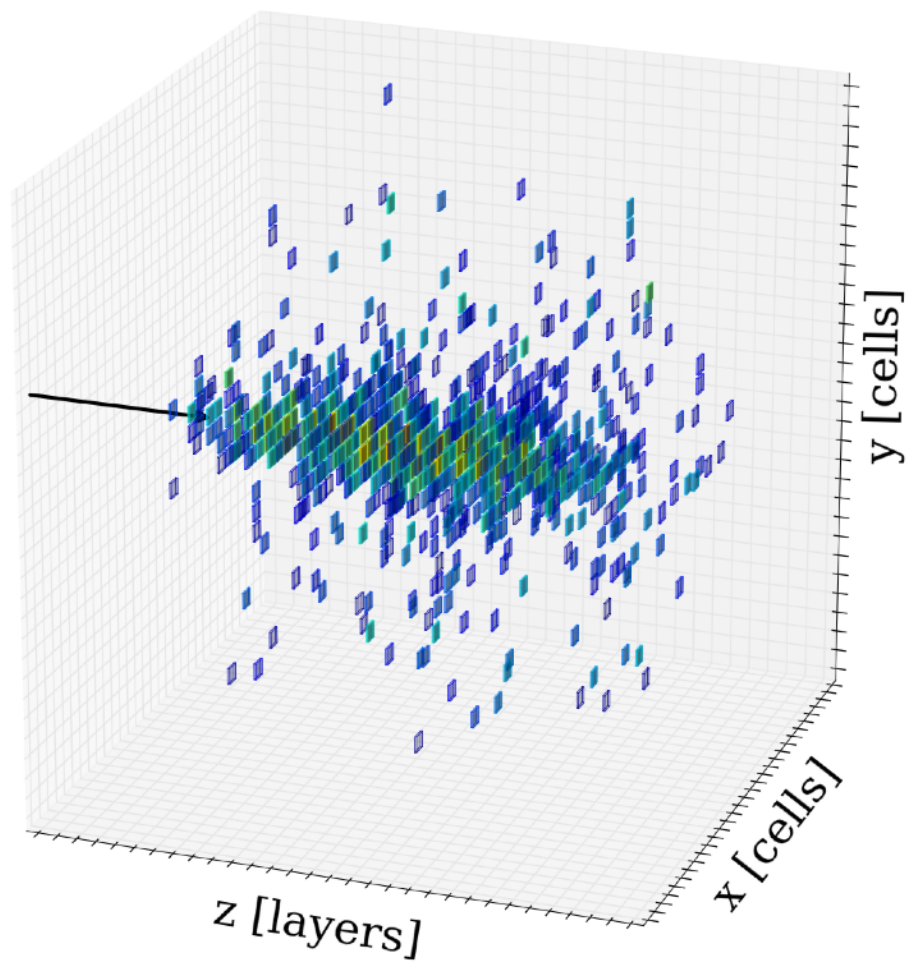
Split!



2302.11594

Simulation targets





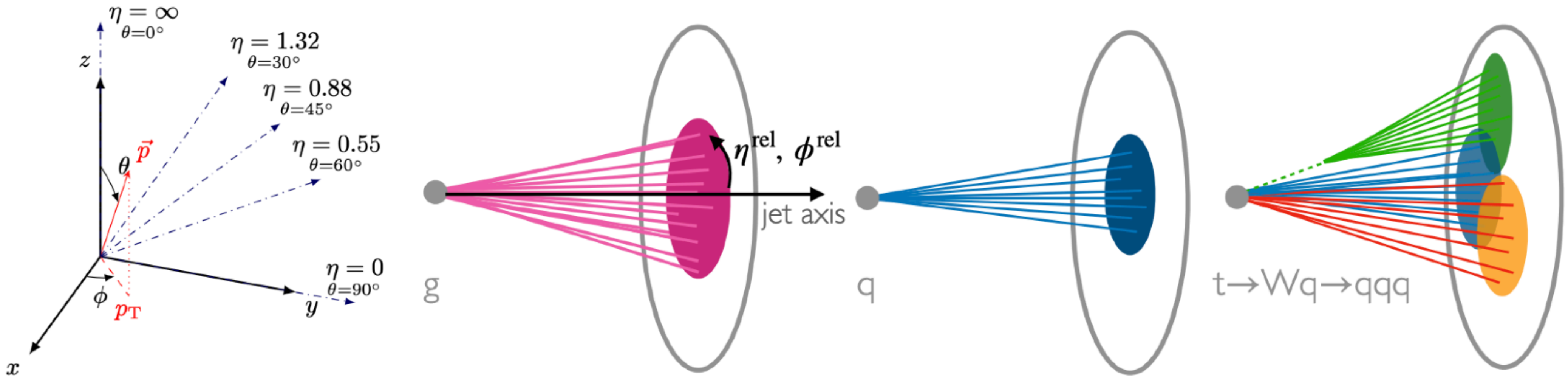
How to represent?

Tabular data

Fixed grid (voxels)
Limiting for high-dimensions (sparse data)

Point clouds / graphs

Simulation targets



Before tackling showers in calorimeters:
Look at jet constituents (JetNet data):
3 features per constituents
up to 30/150 constituents/jet

How to represent?

Tabular data

Fixed grid (voxels)
Limiting for high-dimensions (sparse data)

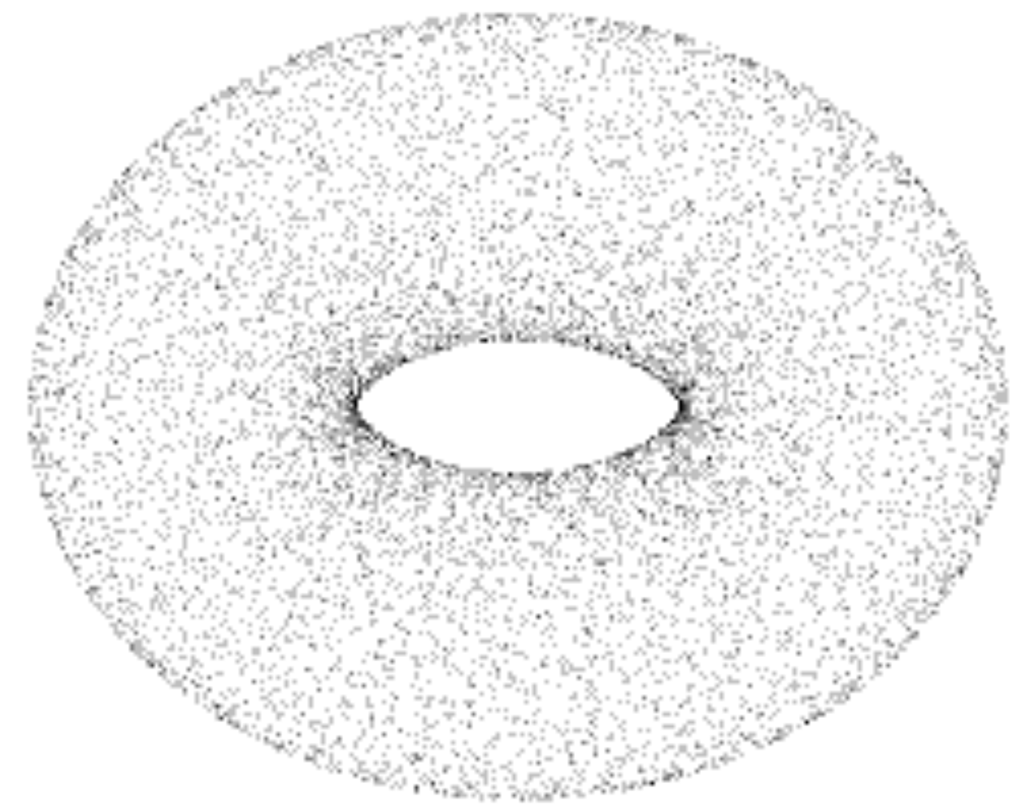
Point clouds / graphs

Why?
Useful stepping stone
In-situ background

2106.11535

Point Clouds

- Example:
  Sensors in a space
  - Fixed grid vs arbitrary positions
  - Potential sparsity of data
- Permutation symmetry
- Can view as trivial graph



Examples

Total data $\{x^j\}_{j=1\ldots N}$

with $x^j = \{\vec{p}^{\,j}_1, \ldots \vec{p}^{\,j}_i, \ldots, \vec{P}^{\,j}_{L(j)}\}$

and $\vec{p}_i \in \mathbb{R}^D$

Deep Sets

Theorem 7 *Let $f : [0,1]^M \to \mathbb{R}$ be a permutation invariant continuous function iff it has the representation*

$$f(x_1, ..., x_M) = \rho\left(\sum_{m=1}^{M} \phi(x_m)\right) \tag{18}$$

for some continuous outer and inner function $\rho : \mathbb{R}^{M+1} \to \mathbb{R}$ and $\phi : \mathbb{R} \to \mathbb{R}^{M+1}$ respectively. The inner function ϕ is independent of the function f.

$$X = \left.\begin{matrix} \vec{p}_1 = \{r_1, \theta_1, \varphi_1, T_1\} \\ \vdots \qquad\qquad \vdots \\ \vec{p}_L = \{r_L, \theta_L, \varphi_L, T_L\} \end{matrix}\right\}$$

$$f(x) = \rho\left(\Sigma_i \, \phi(\vec{p}_i)\right)$$

$$\Uparrow$$

$$f : \mathbb{R}^{4\times L} \to \mathbb{R}^1$$

$$\phi : \mathbb{R}^4 \to \mathbb{R}^S$$

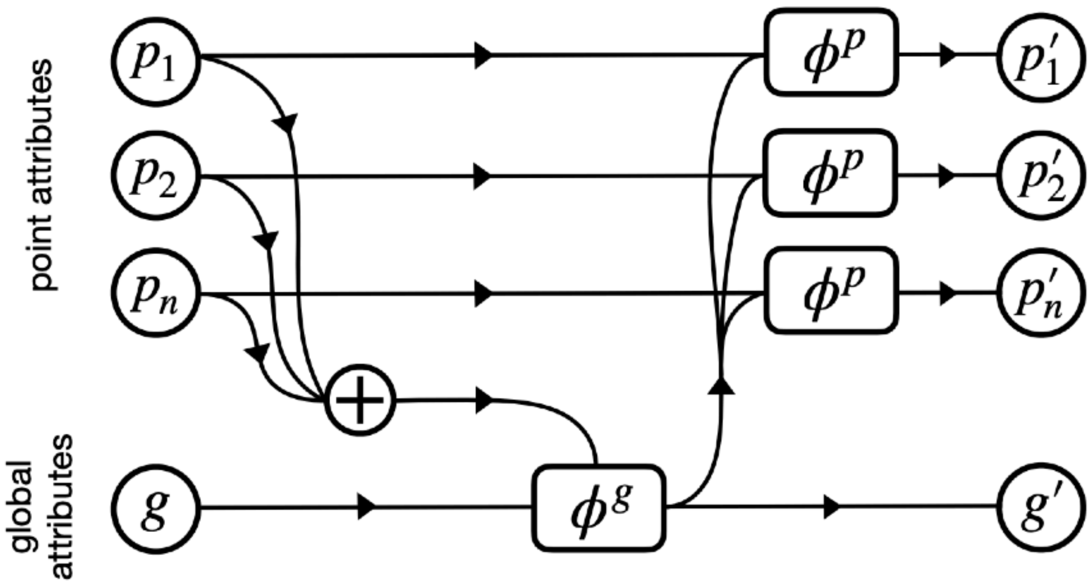
Latent Space Dimension

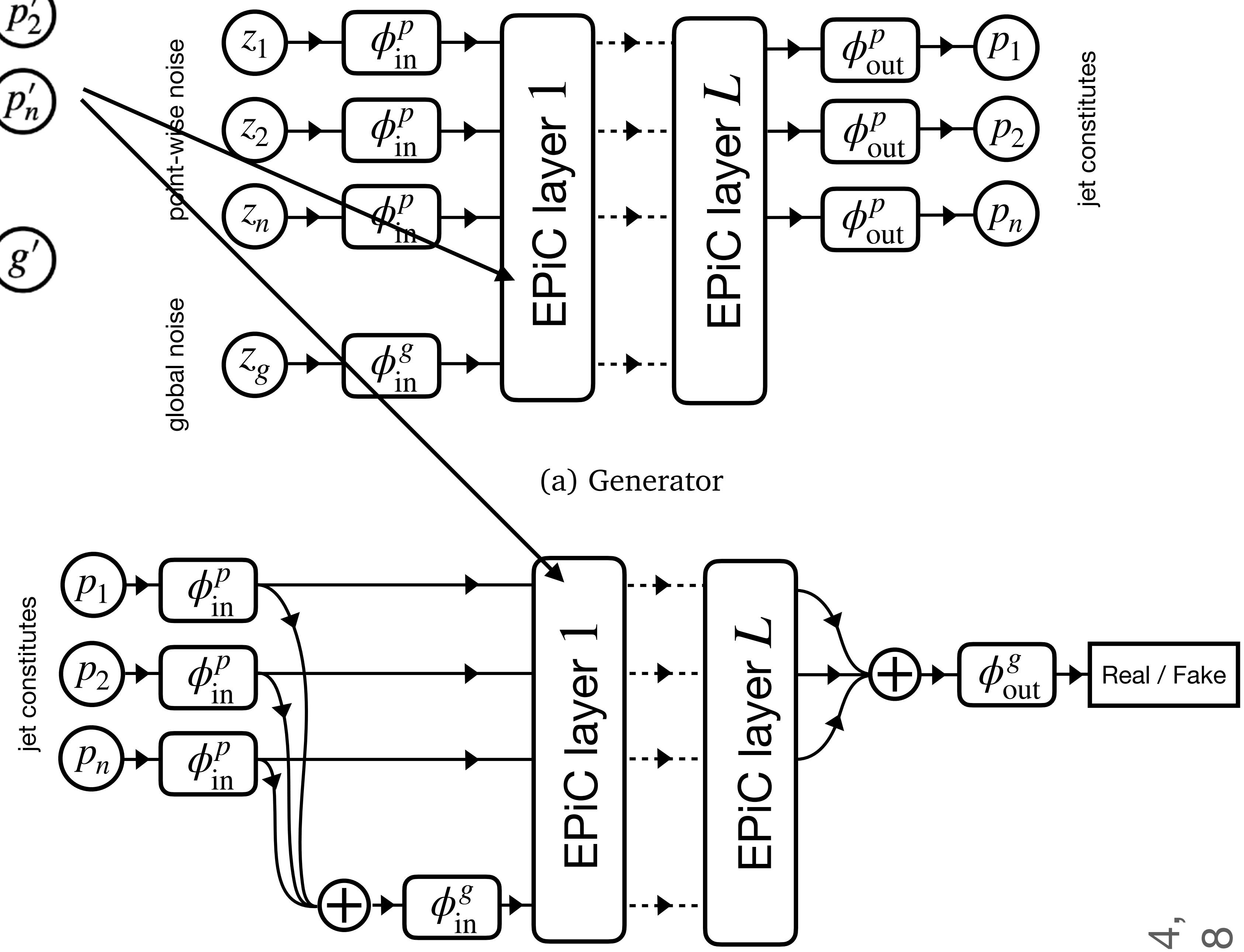
$$\Sigma : \mathbb{R}^{S\times L} \to \mathbb{R}^S$$

$$\rho : \mathbb{R}^S \to \mathbb{R}^1$$

1703.06114

How to GAN with it





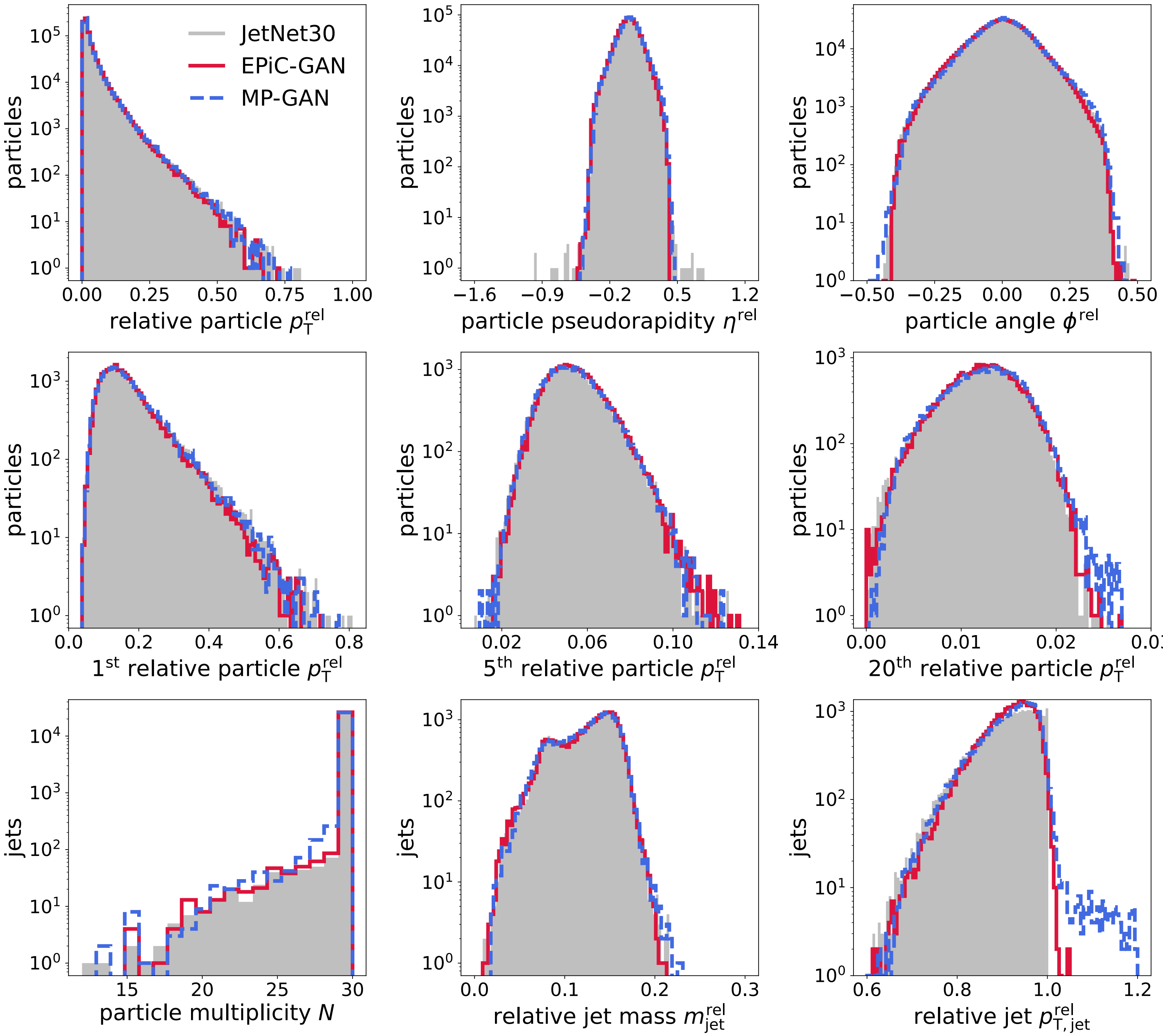
(a) Generator

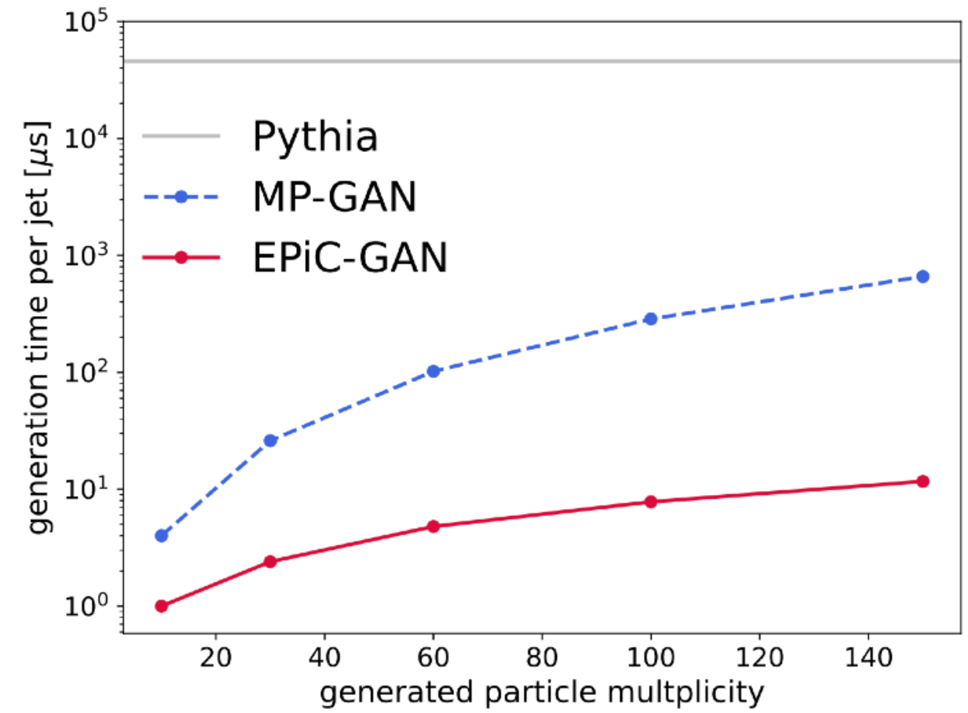
(b) Discriminator

Add permutation symmetry to GAN architecture

1703.06114, 2301.08128

Generative results III

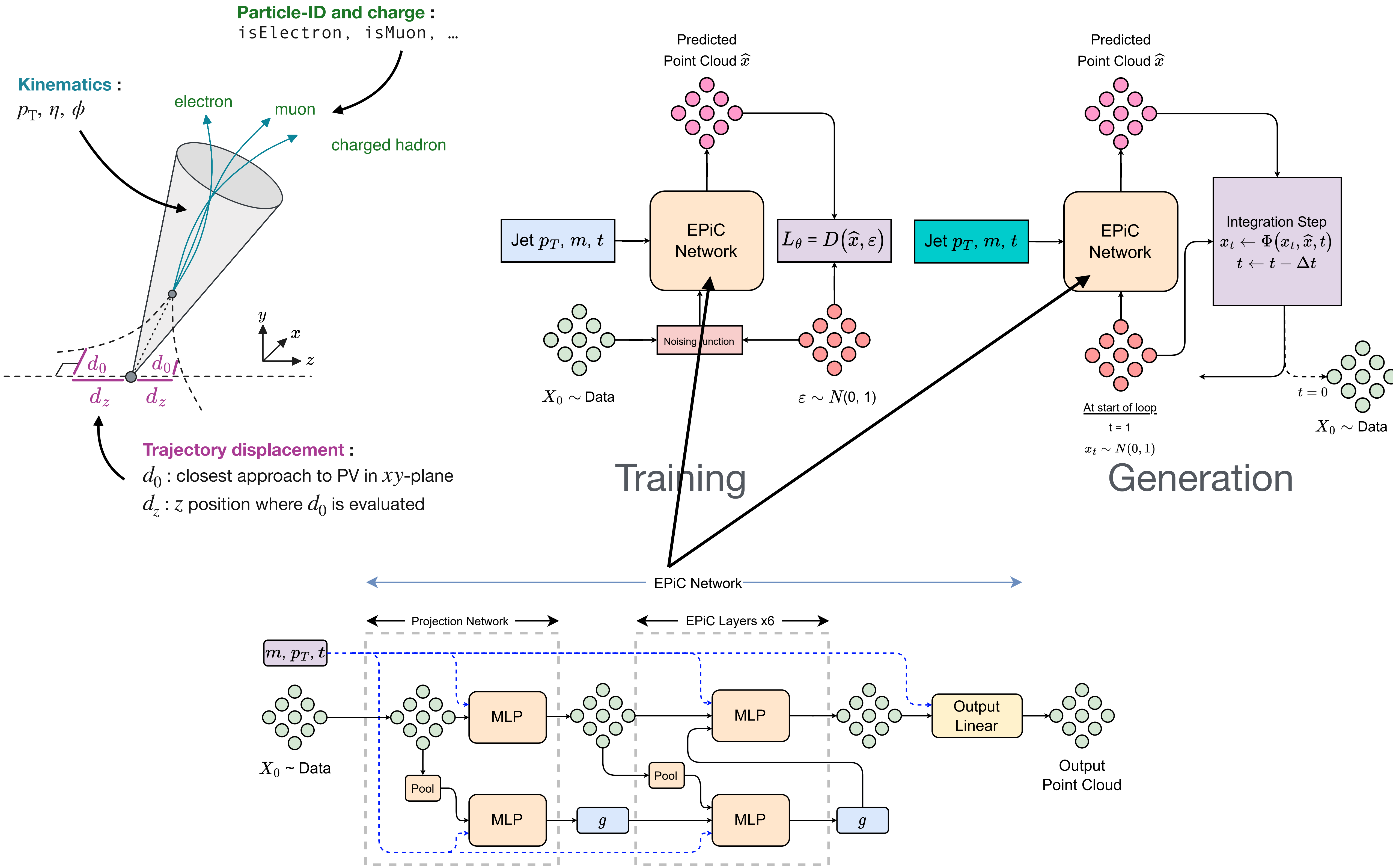




Point clouds/graph GANs successfully generate jet constituents

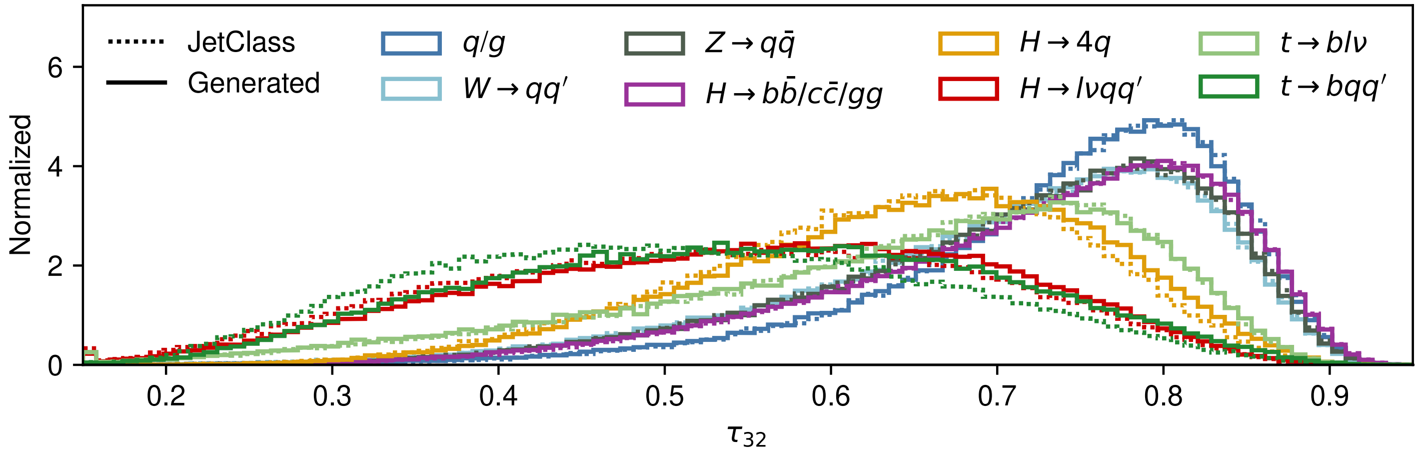
2301.08128

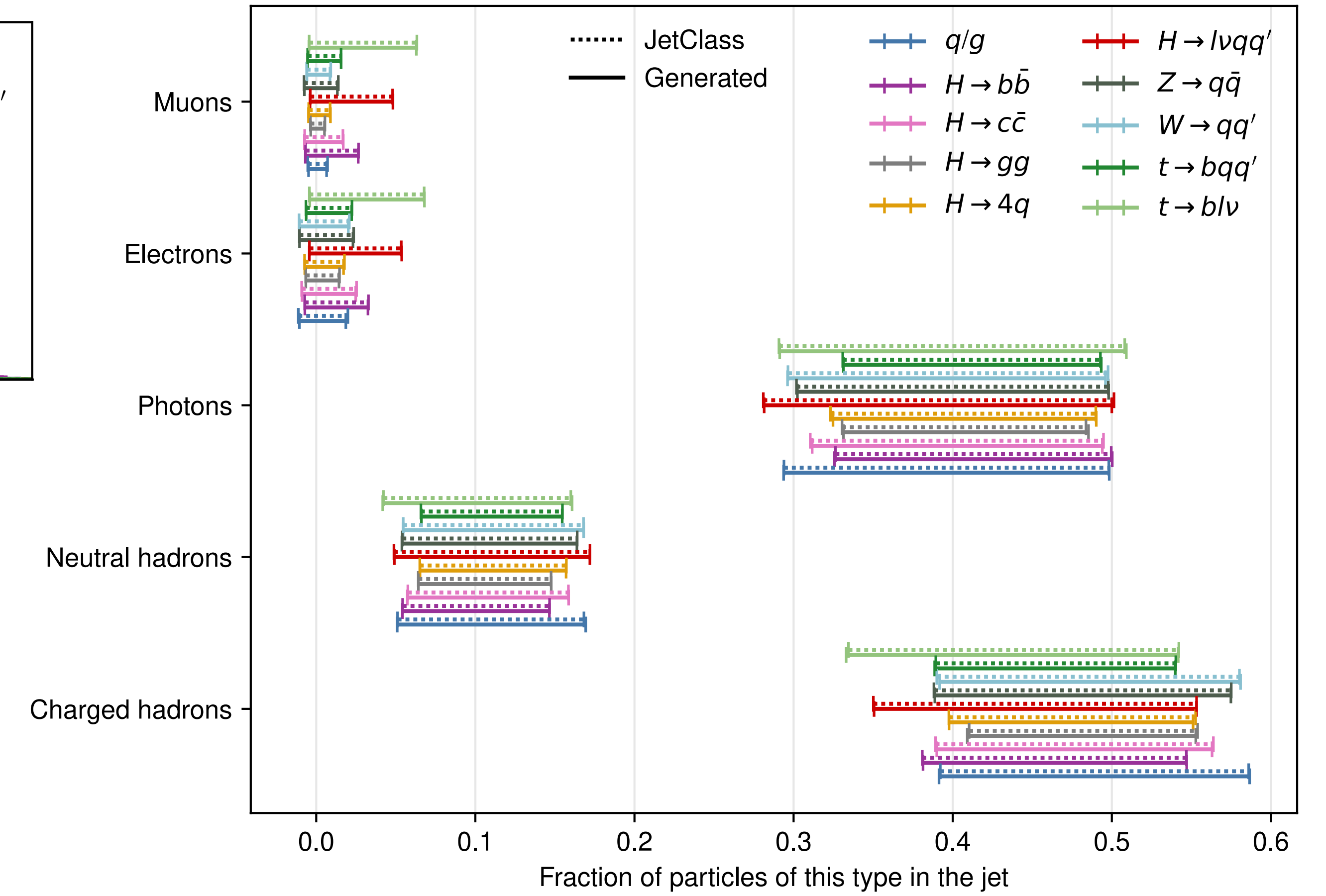
Beyond kinematics



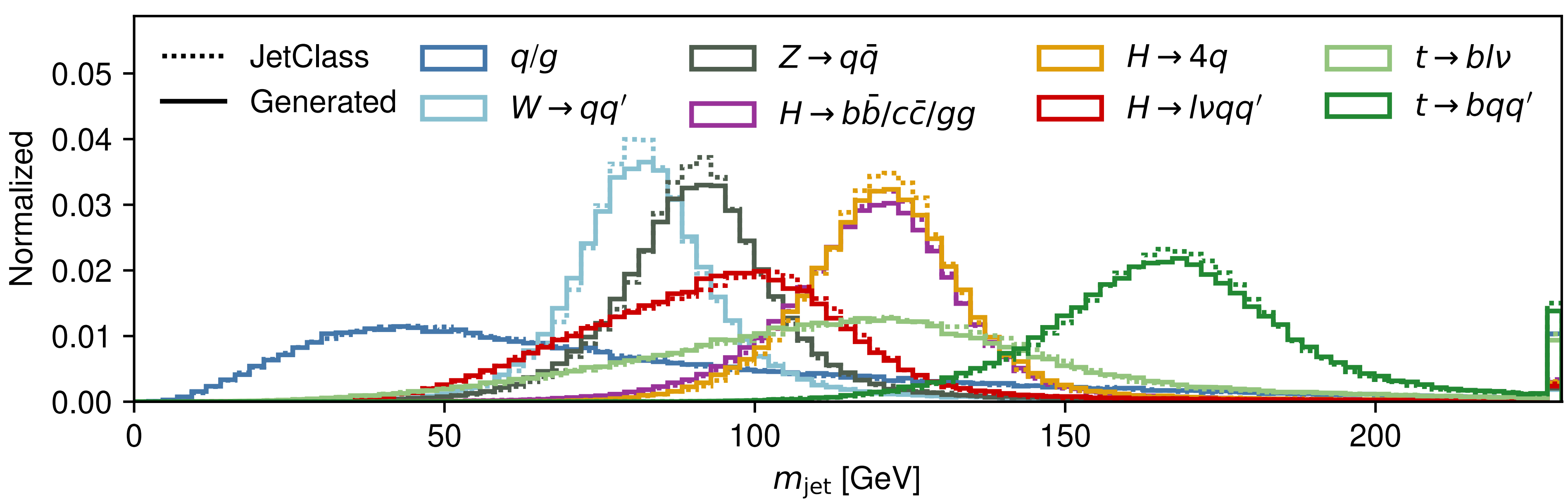
2310.00049, 2312.00123

Beyond kinematics





Good agreement across distributions



2310.00049, 2312.00123

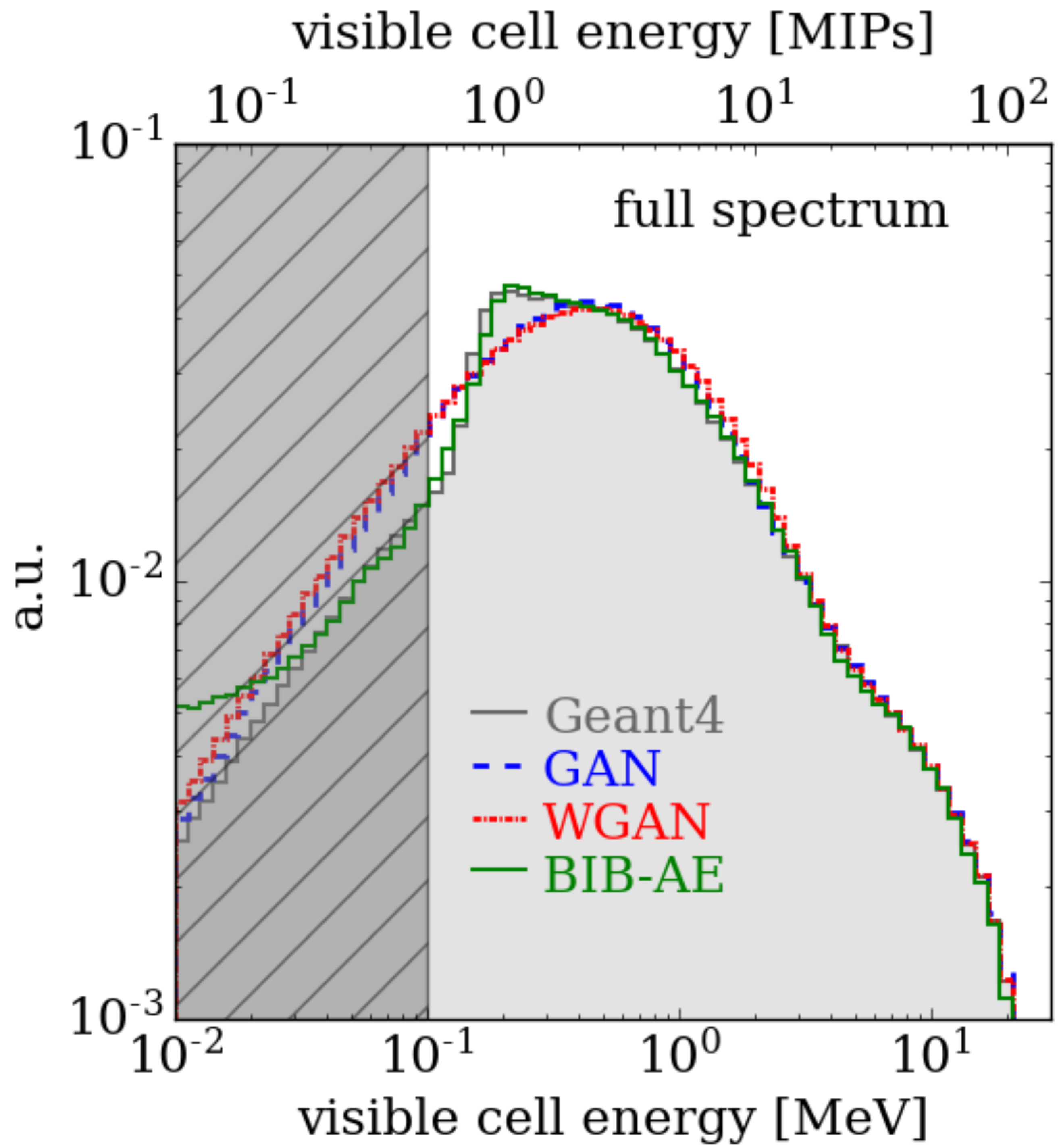
Quality Metrics

Quality of simulation

Target $\longleftrightarrow$ Generative Model

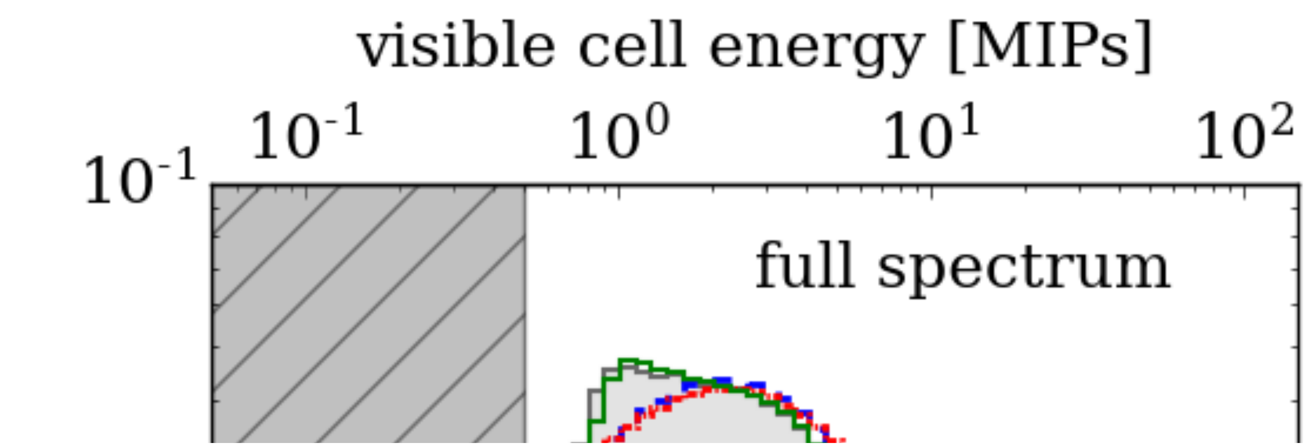
How well does the generative model describe the training data?

One-dimensional metrics



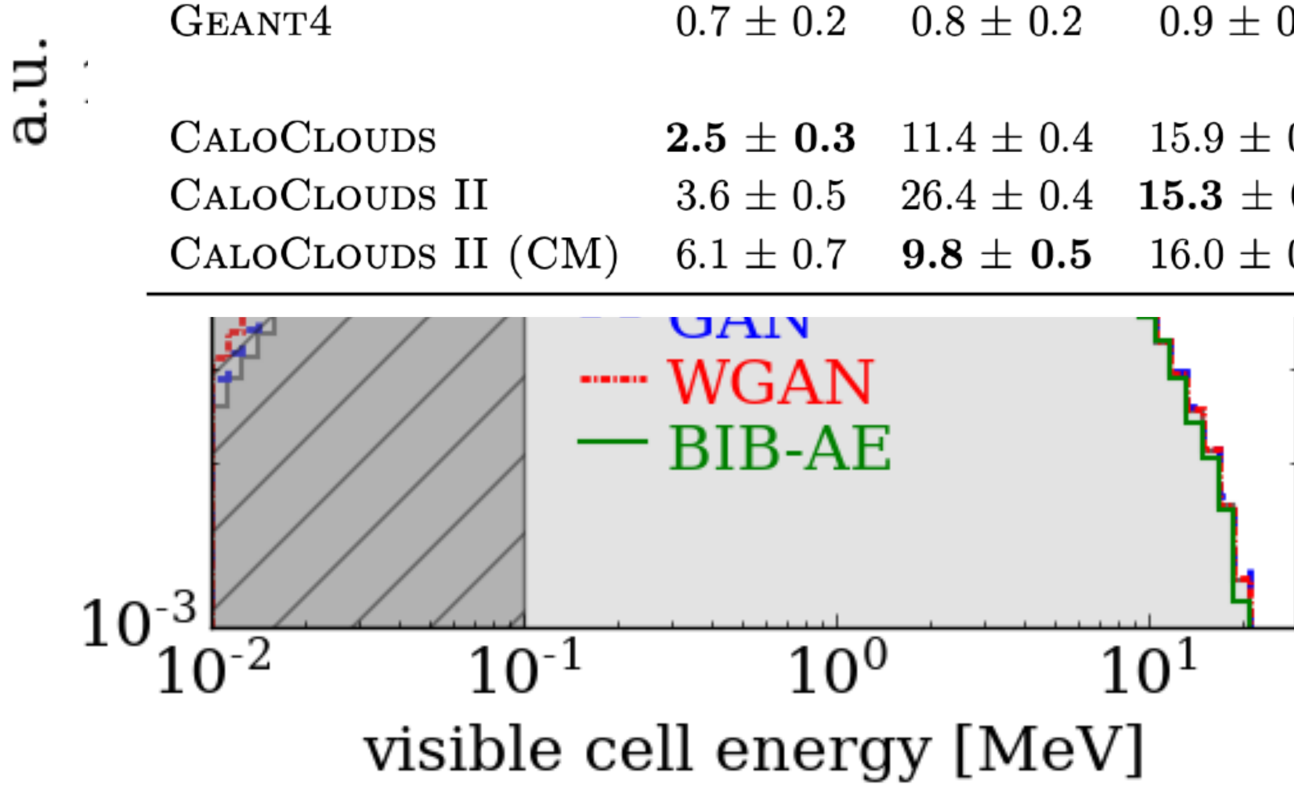
Visual inspection is insufficient

One-dimensional metrics

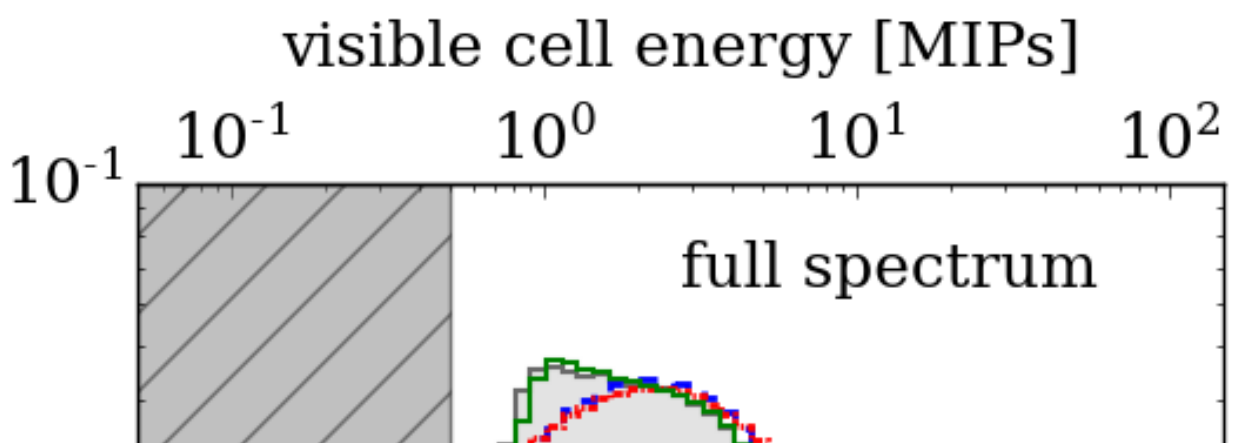


Wasserstein distance

Simulator	$W_1^{N_{\mathrm{hits}}}$ $(\times 10^{-3})$	$W_1^{E_{\mathrm{vis}}/E_{\mathrm{inc}}}$ $(\times 10^{-3})$	$W_1^{E_{\mathrm{cell}}}$ $(\times 10^{-3})$	$W_1^{E_{\mathrm{long}}}$ $(\times 10^{-3})$	$W_1^{E_{\mathrm{radial}}}$ $(\times 10^{-3})$	$W_1^{m_{1,X}}$ $(\times 10^{-3})$	$W_1^{m_{1,Y}}$ $(\times 10^{-3})$	$W_1^{m_{1,Z}}$ $(\times 10^{-3})$
GEANT4	0.7 ± 0.2	0.8 ± 0.2	0.9 ± 0.4	0.7 ± 0.8	0.7 ± 0.1	0.9 ± 0.1	1.1 ± 0.3	0.9 ± 0.3
CALOCLOUDS	**2.5 ± 0.3**	11.4 ± 0.4	15.9 ± 0.7	**2.0 ± 1.3**	38.8 ± 1.4	4.0 ± 0.4	8.7 ± 0.3	1.4 ± 0.5
CALOCLOUDS II	3.6 ± 0.5	26.4 ± 0.4	**15.3 ± 0.6**	3.7 ± 1.6	11.6 ± 1.5	**2.4 ± 0.4**	**7.6 ± 0.2**	3.9 ± 0.4
CALOCLOUDS II (CM)	6.1 ± 0.7	**9.8 ± 0.5**	16.0 ± 0.7	**2.0 ± 1.4**	**8.3 ± 1.9**	3.0 ± 0.4	9.5 ± 0.6	**1.2 ± 0.5**

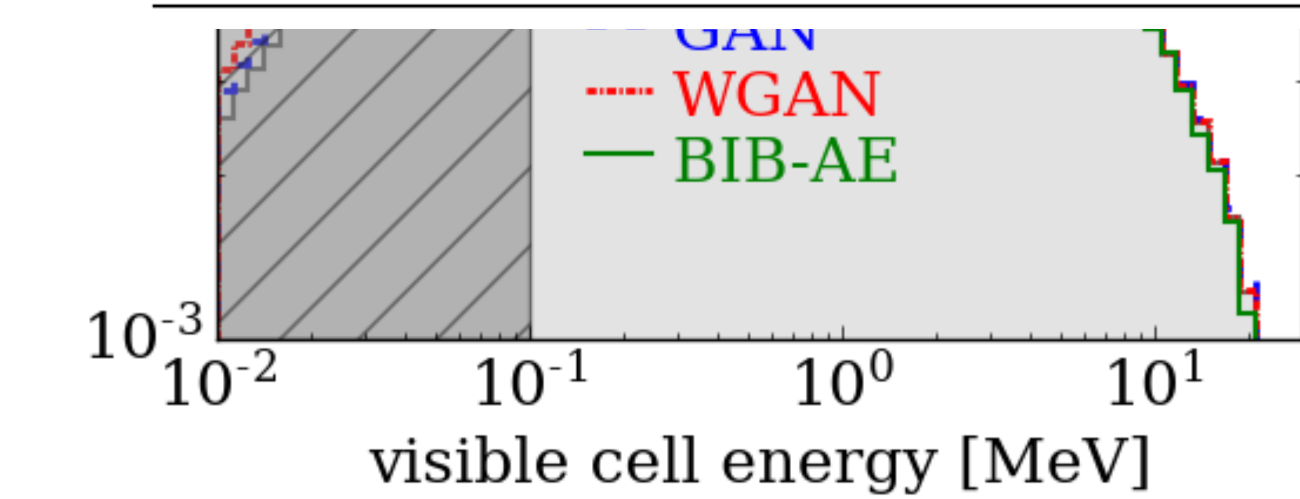


One-dimensional metrics



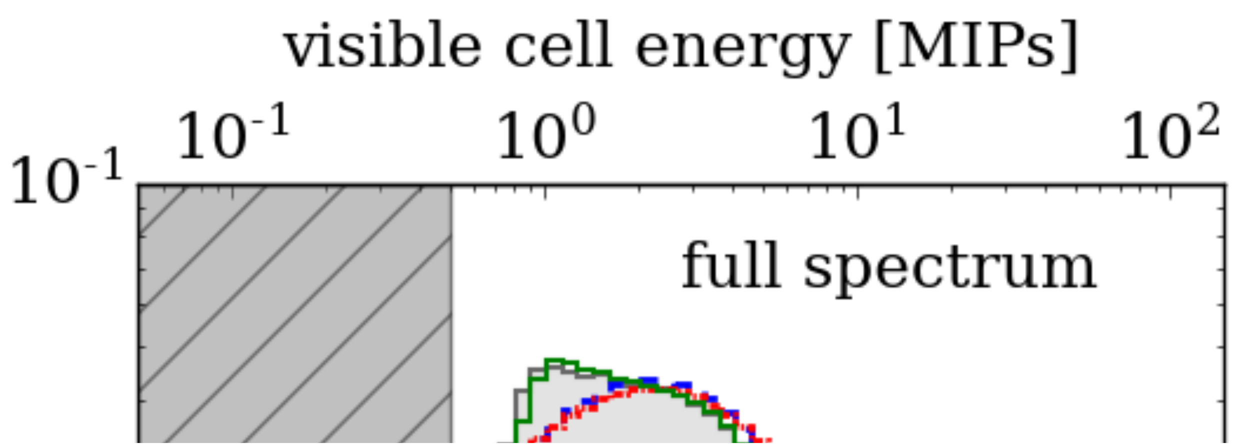
Wasserstein distance

Simulator	$W_1^{N_{\text{hits}}}$ $(\times 10^{-3})$	$W_1^{E_{\text{vis}}/E_{\text{inc}}}$ $(\times 10^{-3})$	$W_1^{E_{\text{cell}}}$ $(\times 10^{-3})$	$W_1^{E_{\text{long}}}$ $(\times 10^{-3})$	$W_1^{E_{\text{radial}}}$ $(\times 10^{-3})$	$W_1^{m_{1,X}}$ $(\times 10^{-3})$	$W_1^{m_{1,Y}}$ $(\times 10^{-3})$	$W_1^{m_{1,Z}}$ $(\times 10^{-3})$
GEANT4	0.7 ± 0.2	0.8 ± 0.2	0.9 ± 0.4	0.7 ± 0.8	0.7 ± 0.1	0.9 ± 0.1	1.1 ± 0.3	0.9 ± 0.3
CALOCLOUDS	**2.5 ± 0.3**	11.4 ± 0.4	15.9 ± 0.7	**2.0 ± 1.3**	38.8 ± 1.4	4.0 ± 0.4	8.7 ± 0.3	1.4 ± 0.5
CALOCLOUDS II	3.6 ± 0.5	26.4 ± 0.4	**15.3 ± 0.6**	3.7 ± 1.6	11.6 ± 1.5	**2.4 ± 0.4**	**7.6 ± 0.2**	3.9 ± 0.4
CALOCLOUDS II (CM)	6.1 ± 0.7	**9.8 ± 0.5**	16.0 ± 0.7	**2.0 ± 1.4**	**8.3 ± 1.9**	3.0 ± 0.4	9.5 ± 0.6	**1.2 ± 0.5**



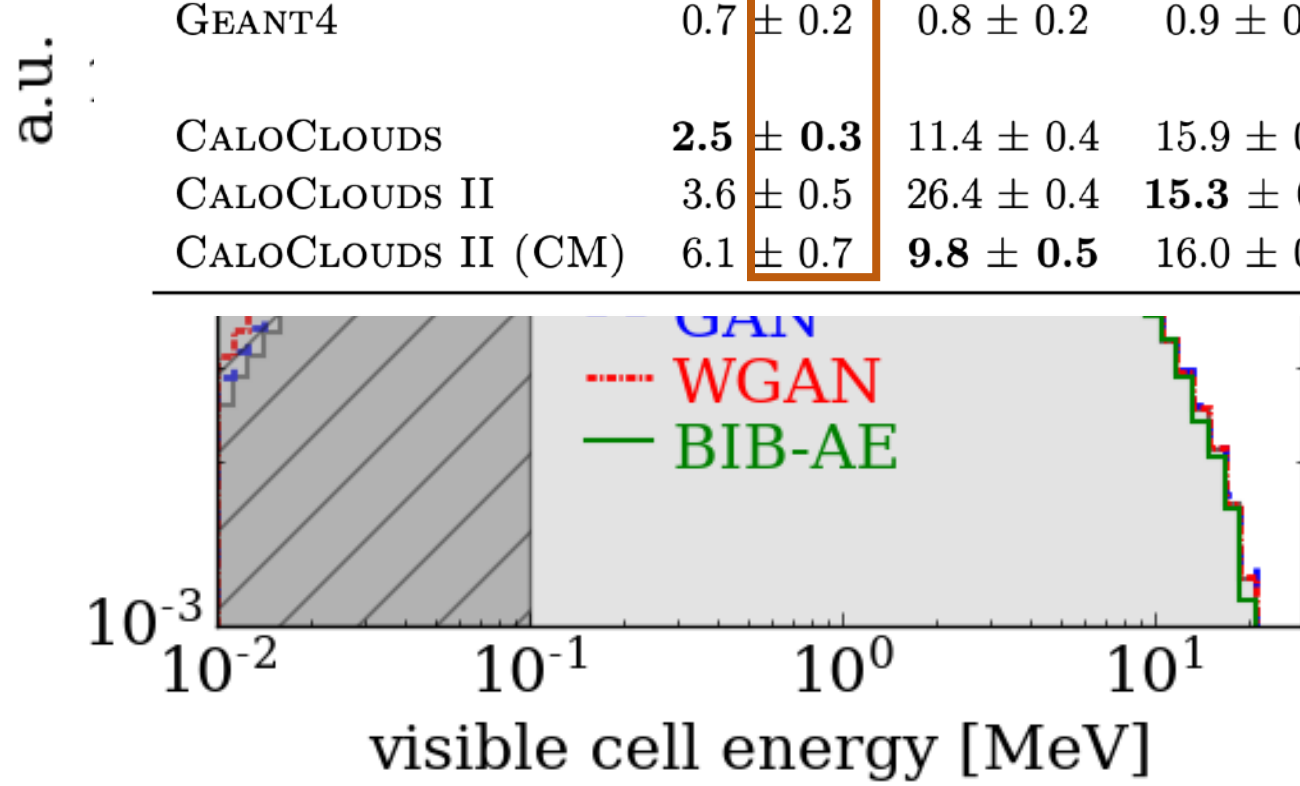
Floor is given by sample-size effects of GEANT4 vs itself

One-dimensional metrics



Wasserstein distance

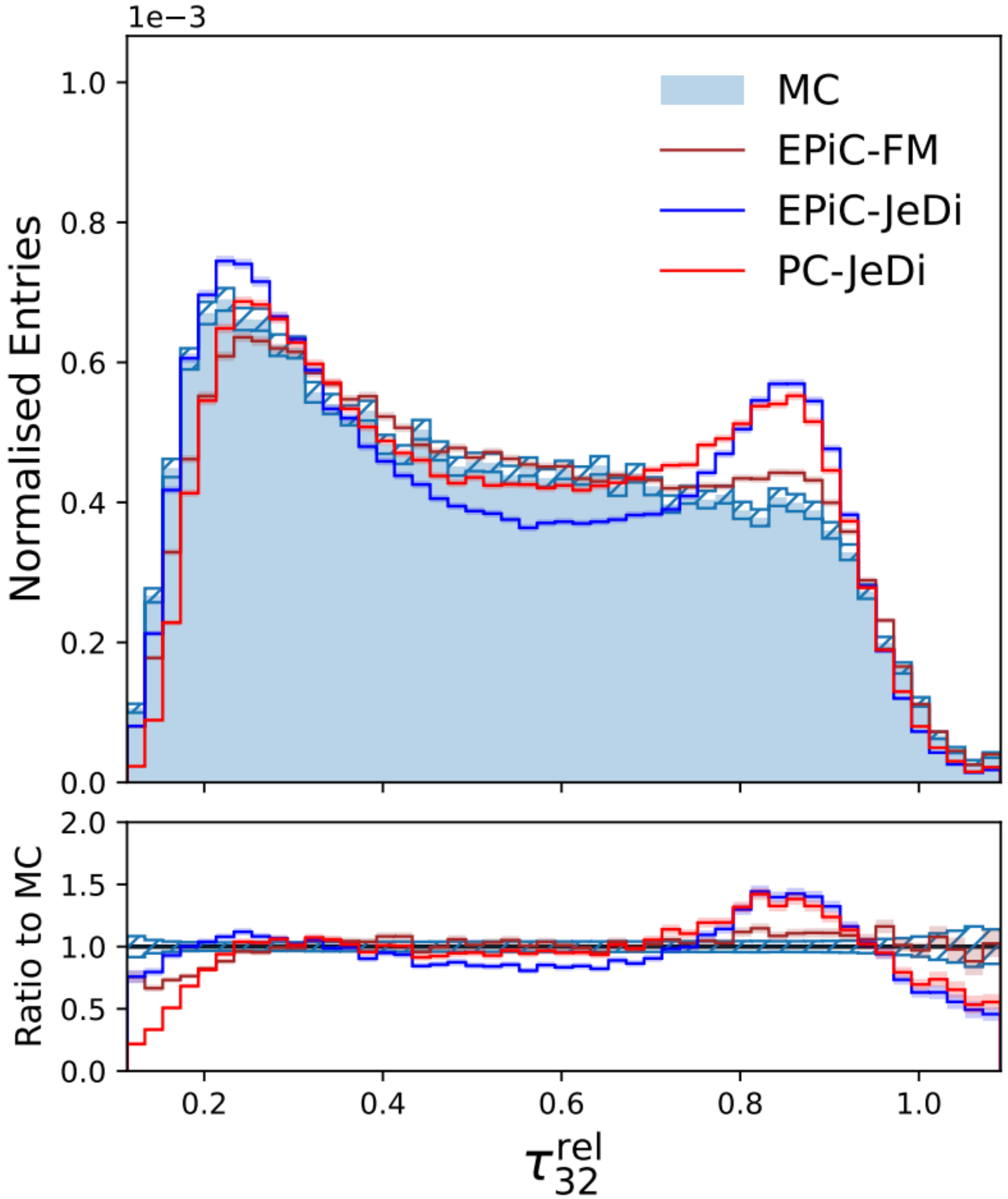
Simulator	$W_1^{N_{\text{hits}}}$ $(\times 10^{-3})$	$W_1^{E_{\text{vis}}/E_{\text{inc}}}$ $(\times 10^{-3})$	$W_1^{E_{\text{cell}}}$ $(\times 10^{-3})$	$W_1^{E_{\text{long}}}$ $(\times 10^{-3})$	$W_1^{E_{\text{radial}}}$ $(\times 10^{-3})$	$W_1^{m_{1,X}}$ $(\times 10^{-3})$	$W_1^{m_{1,Y}}$ $(\times 10^{-3})$	$W_1^{m_{1,Z}}$ $(\times 10^{-3})$
GEANT4	0.7 ± 0.2	0.8 ± 0.2	0.9 ± 0.4	0.7 ± 0.8	0.7 ± 0.1	0.9 ± 0.1	1.1 ± 0.3	0.9 ± 0.3
CALOCLOUDS	**2.5 ± 0.3**	11.4 ± 0.4	15.9 ± 0.7	**2.0 ± 1.3**	38.8 ± 1.4	4.0 ± 0.4	8.7 ± 0.3	1.4 ± 0.5
CALOCLOUDS II	3.6 ± 0.5	26.4 ± 0.4	**15.3 ± 0.6**	3.7 ± 1.6	11.6 ± 1.5	**2.4 ± 0.4**	**7.6 ± 0.2**	3.9 ± 0.4
CALOCLOUDS II (CM)	6.1 ± 0.7	**9.8 ± 0.5**	16.0 ± 0.7	**2.0 ± 1.4**	**8.3 ± 1.9**	3.0 ± 0.4	9.5 ± 0.6	**1.2 ± 0.5**



Floor is given by sample-size effects of GEANT4 vs itself

Uncertainty is standard deviation of 10 independent samples

One-dimensional metrics



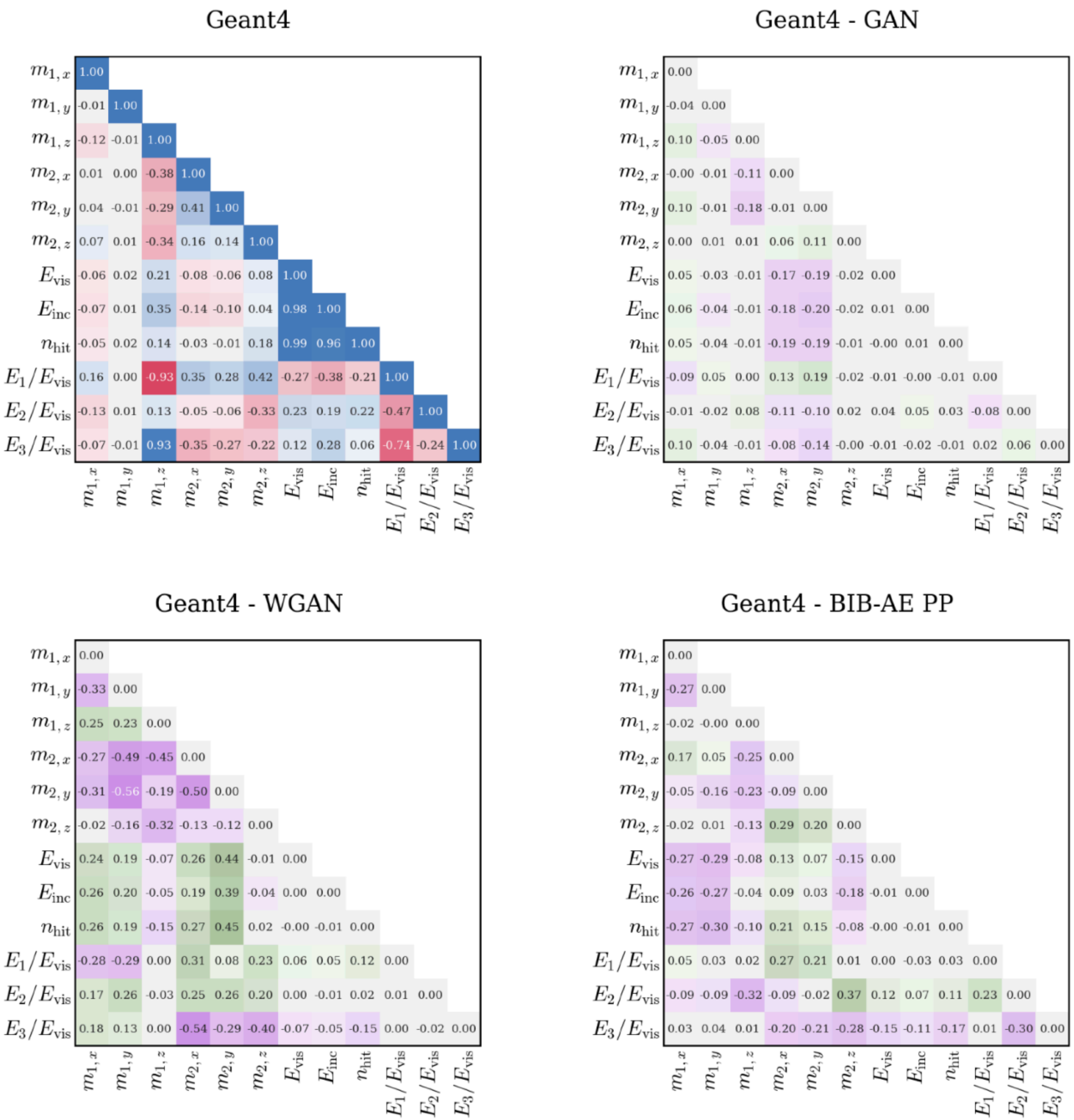
Wasserstein distance

More robust, well defined also for non-overlapping distributions

Kullback-Leibler divergence

More intuitive for shape-mismodelling

Two-dimensional metrics



Pair-wise correlations contain more information

Multi-dimensional metrics

# Showers per simulator	AUC GEANT4 vs L2LFLOWS	AUC GEANT4 vs BIB-AE
95k	0.8518 ± 0.0042	0.9947 ± 0.0025
190k	0.8768 ± 0.0029	–
380k	0.8962 ± 0.0024	–
760k	0.9402 ± 0.0011	–

Capture full phase space information with classifiers

Still depends on training data

Choice of classifier

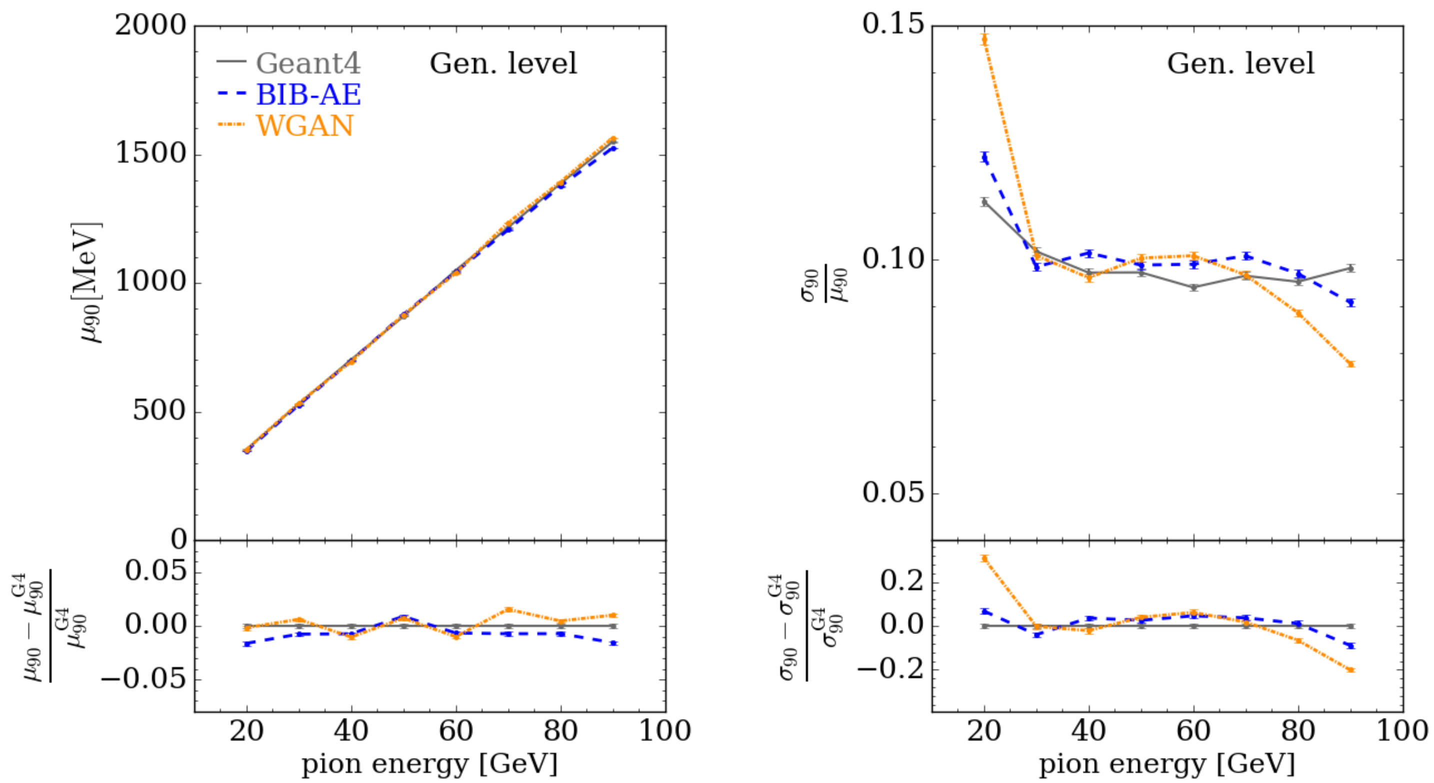
How good, is good enough really?

(Can also compare multiple models with multi-class classifier and then evaluate on data)

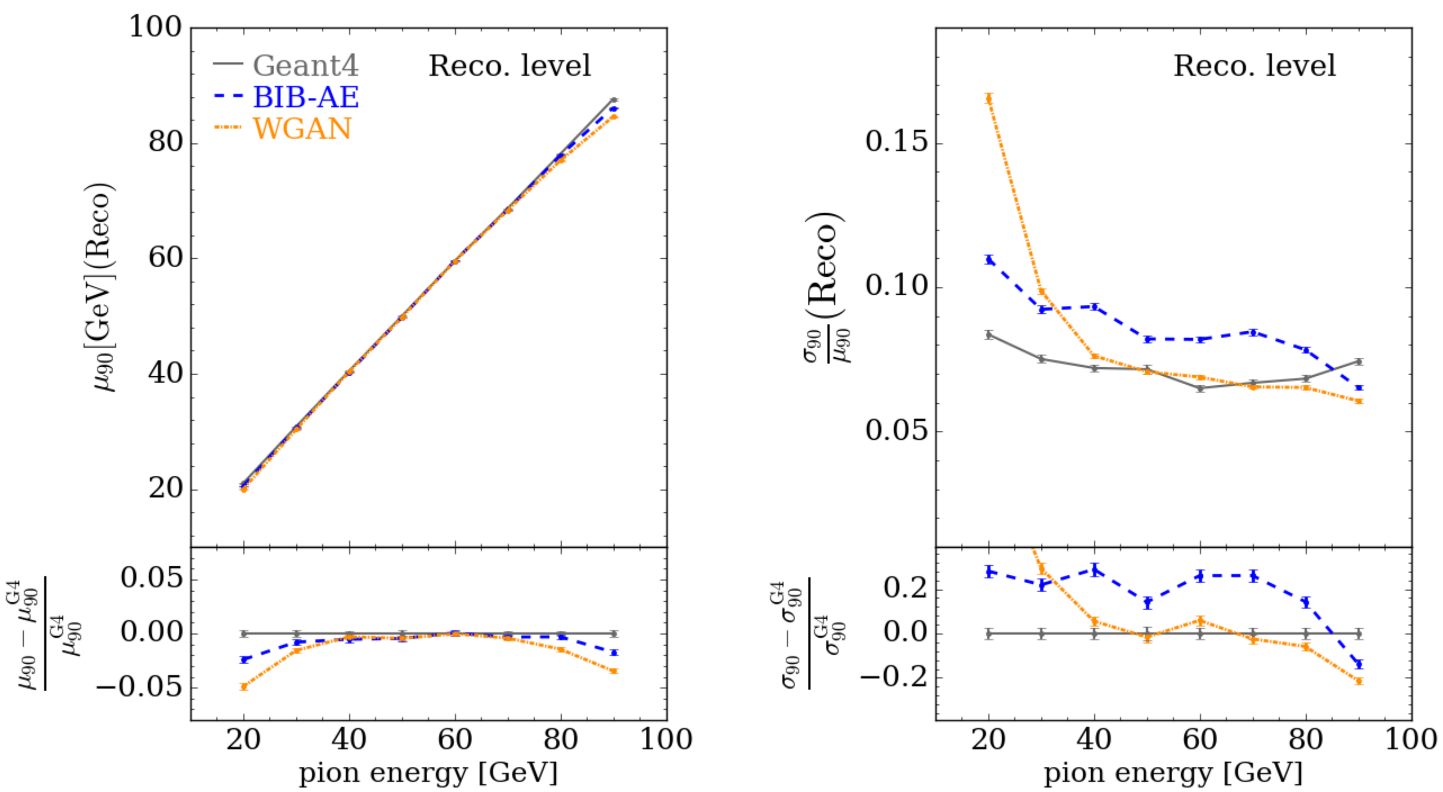
(Or use latent space of pre-trained classifier: Frechet Inception/Particle Distance)

2309.05704, 2211.10295

Adding reconstruction



Without reconstruction

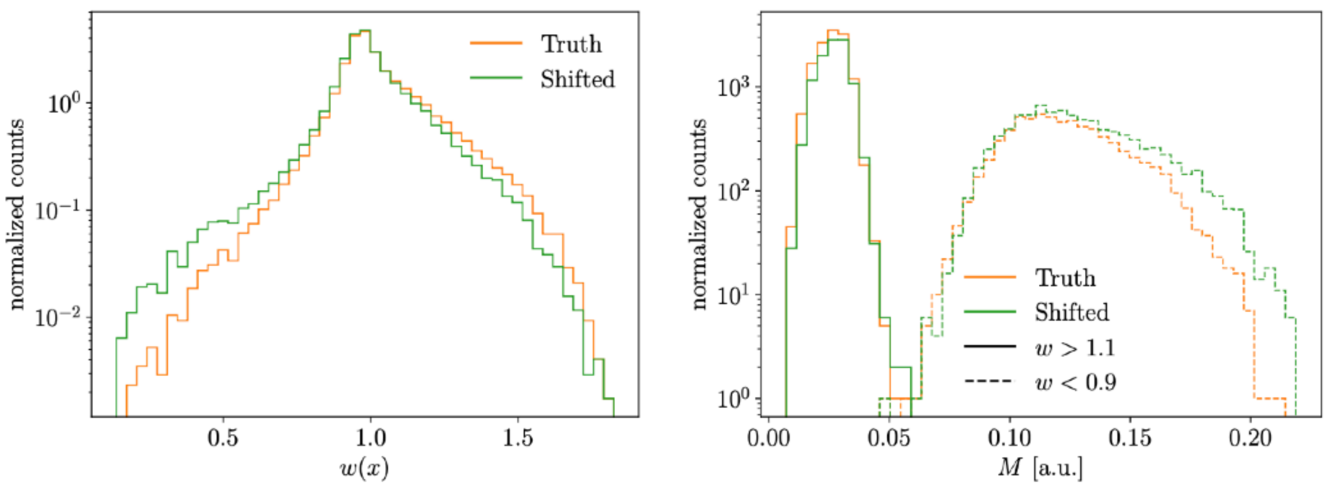


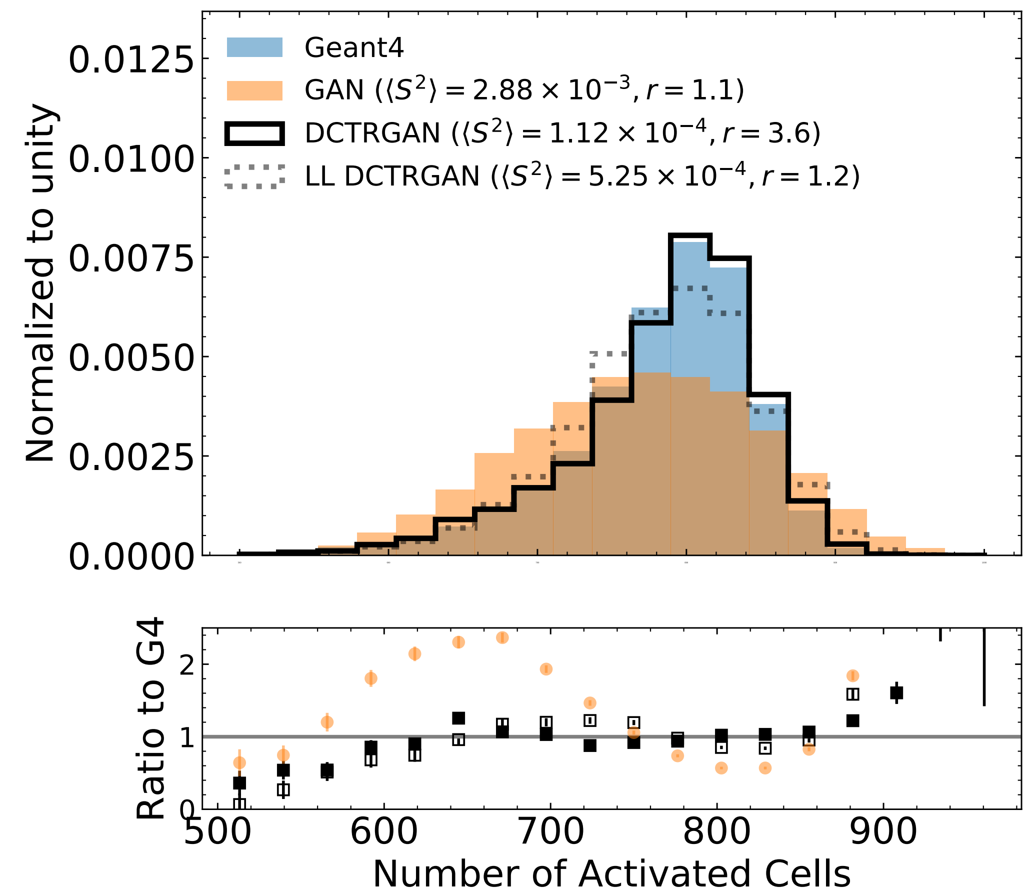
With reconstruction

→ Non-linear effects of reconstruction can change relative performance

Weights

$$w(x) = \frac{p_{\text{data}}(x)}{p_{\text{model}}(x)} = \frac{C(x)}{1 - C(x)} \qquad \text{with} \qquad C(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_{\text{model}}(x)}$$





Train classifiers to find mismodelling

or to reweight distributions

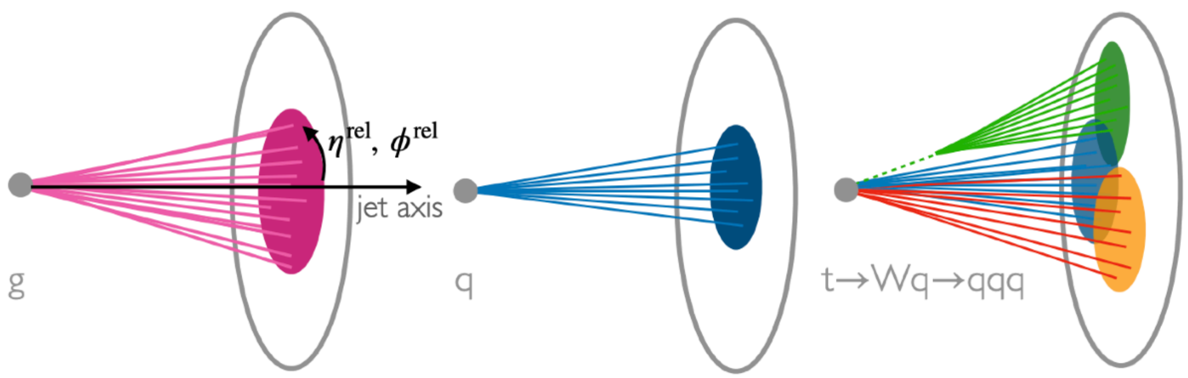
2009.03796, 2305.16774

Quality of simulation

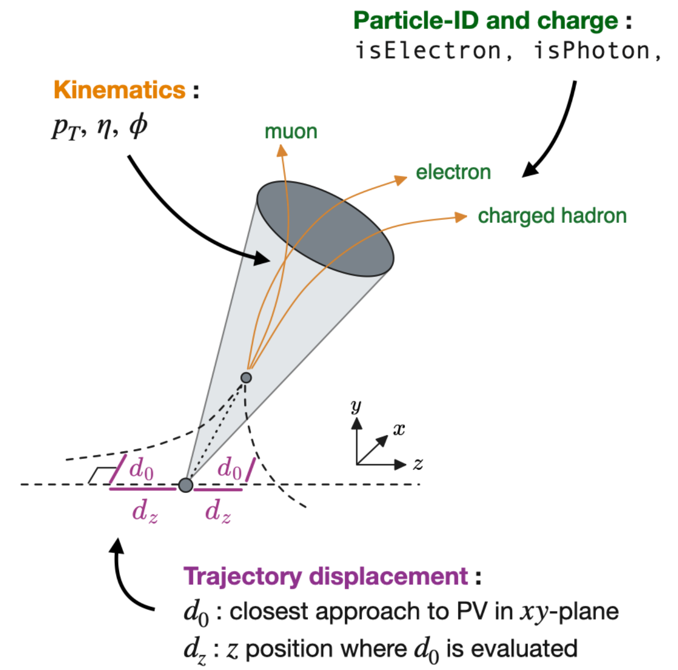
~~Target~~
Actual data ⟷ Generative Model

Closing II

Common Datasets



JetNet

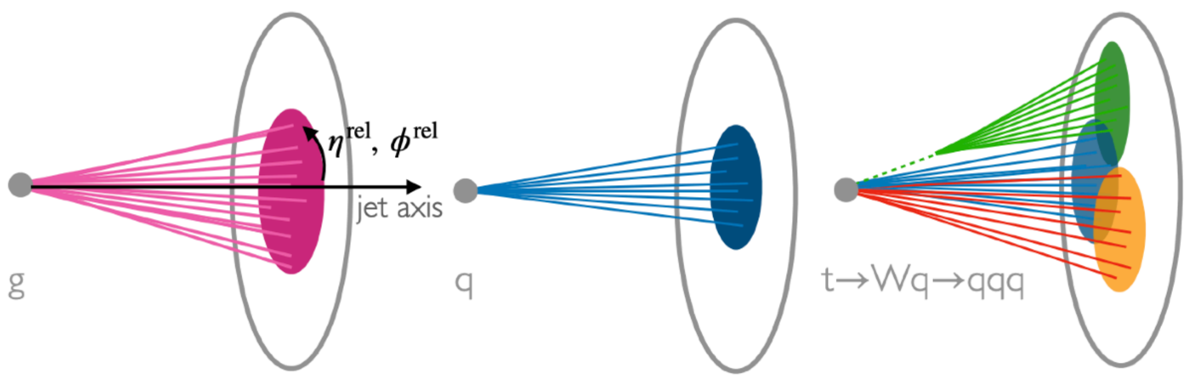


JetClass

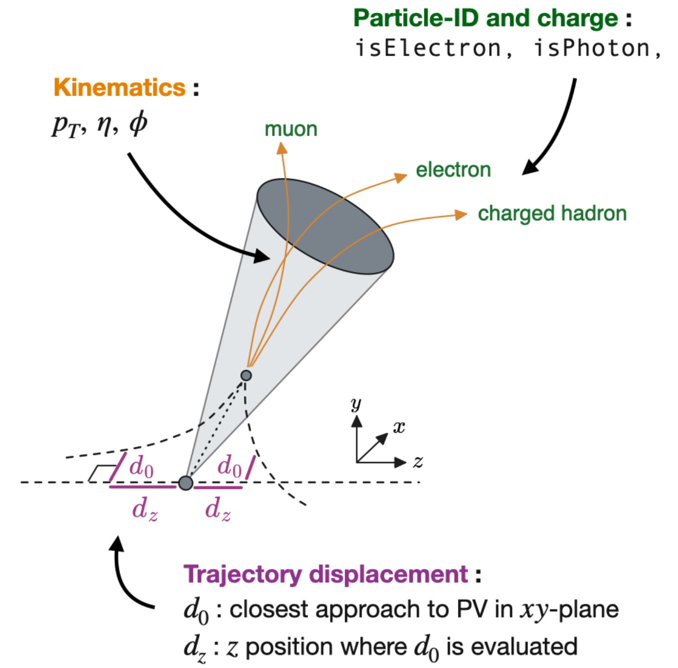
	JetNet [3]	**JetClass [1]**
Jet types	5 types	10 types (several decay channels for top and H jets)
Dataset size	180 thousand jets per class	12.5 million jets per class (70x more than JetNet)
Features	Kinematics	Kinematics, Particle-ID and charge, trajectory displacement

2106.11535, 2202.03772

Common Datasets



JetNet



JetClass

Fast Calorimeter Simulation Challenge 2022

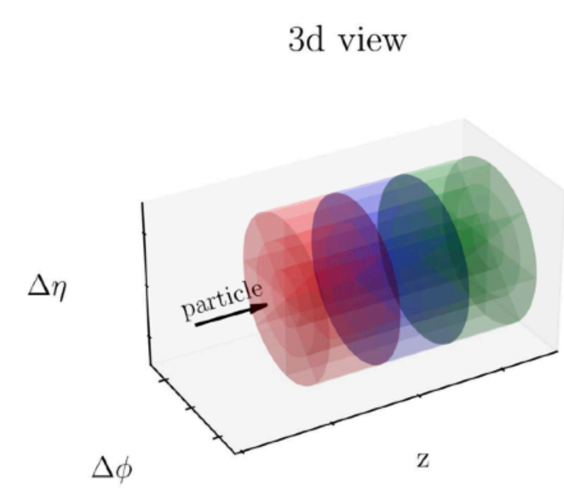
View on GitHub

Welcome to the home of the first-ever Fast Calorimeter Simulation Challenge!

The purpose of this challenge is to spur the development and benchmarking of fast and high-fidelity calorimeter shower generation using deep learning methods. Currently, generating calorimeter showers of interacting particles (electrons, photons, pions, ...) using GEANT4 is a major computational bottleneck at the LHC, and it is forecast to overwhelm the computing budget of the LHC experiments in the near future. Therefore there is an urgent need to develop GEANT4 emulators that are both fast (computationally lightweight) and accurate. The LHC collaborations have been developing fast simulation methods for some time, and the hope of this challenge is to directly compare new deep learning approaches on common benchmarks. It is expected that participants will make use of cutting-edge techniques in generative modeling with deep learning, e.g. GANs, VAEs and normalizing flows.

This challenge is modeled after two previous, highly successful data challenges in HEP – the top tagging community challenge and the LHC Olympics 2020 anomaly detection challenge.

3 datasets of increasing complexity:
DS1: Up to 533 voxels (ATLAS)
DS2: 6480 voxels
DS3: 40500 voxels



2106.11535, 2202.03772, https://calochallenge.github.io/homepage/

Generative Cheat Sheet