High Performance Computing and Aspects of Computing in Lattice Gauge Theories

//
Giannis Koutsou
Computation-Based Science and Technology Research Centre (CaSToRC)
The Cyprus Institute
//

The Cyprus Institute
RESEARCH·TECHNOLOGY·INNOVATION

EuroPLEx School
Lecture 5 – 20th November 2020

Machine learning applications for LGT

Some applications of machine learning to the study of Lattice Gauge Theories and related models that have emerged recently

- To the detection of *bulk phenomena* such as *phase transitions* and *critical points*, e.g.:
    - Scientific Reports 7, 1 (2017), 8823
    - Nucl. Phys. B 944 (2019) 114639 arXiv:1812.06726
    - arXiv:1903.03506

- To the *analysis of correlation functions*, including in *reconstructing parton distribution functions*, e.g.:
    - Phys. Rev. D100 (2019) 014504, arXiv:1807.05971
    - Phys. Rev. D102 (2020) 9, 094508, arXiv:2007.13800
    - arXiv:2010.03996

- To the *generation* of field configurations, e.g.:
    - Phys. Rev. Lett. 125 (2020) arXiv:2003.06413
    - arXiv:2007.07115
    - arXiv:2008.05456

Machine learning applications for LGT

Some applications of machine learning to the study of Lattice Gauge Theories and related models that have emerged recently

- To the detection of *bulk phenomena* such as *phase transitions* and *critical points*, e.g.:
  - Scientific Reports 7, 1 (2017), 8823
  - Nucl. Phys. B 944 (2019) 114639 arXiv:1812.06726
  - arXiv:1903.03506

- To the *analysis of correlation functions*, including in *reconstructing parton distribution functions*, e.g.:
  - Phys. Rev. D100 (2019) 014504, arXiv:1807.05971
  - Phys. Rev. D102 (2020) 9, 094508, arXiv:2007.13800
  - arXiv:2010.03996

- To the *generation* of field configurations, e.g.:
  - Phys. Rev. Lett. 125 (2020) arXiv:2003.06413
  - arXiv:2007.07115
  - arXiv:2008.05456

Lattice Field Theories

LGT typically require computing integrals such as the following,

$$\langle \mathcal{O} \rangle = \frac{1}{Z} \int d\phi_1 d\phi_2 ... d\phi_D \mathcal{O}(\phi) e^{-S(\phi)} = \frac{1}{Z} \int \prod_{i=1}^{D} d\phi_i \mathcal{O}(\phi) e^{-S(\phi)}$$

- ϕ: the *fields*, with D degrees of freedom
- $\mathcal{O}(\phi)$: a *physical observable* of which we want the expectation value
- $S(\phi)$: the *action* of the theory, a scalar function of ϕ
- Z: the *partition function*, $Z = \int \prod_{i=1}^{D} d\phi_i e^{-S(\phi)}$

Lattice Field Theories

LGT typically require computing integrals such as the following,

$$\langle \mathcal{O} \rangle = \frac{1}{Z} \int d\phi_1 d\phi_2 ... d\phi_D \mathcal{O}(\phi) e^{-S(\phi)} = \frac{1}{Z} \int \prod_{i=1}^{D} d\phi_i \mathcal{O}(\phi) e^{-S(\phi)}$$

- ϕ: the *fields*, with D degrees of freedom
- $\mathcal{O}(\phi)$: a *physical observable* of which we want the expectation value
- $S(\phi)$: the *action* of the theory, a scalar function of ϕ
- Z: the *partition function*, $Z = \int \prod_{i=1}^{D} d\phi_i e^{-S(\phi)}$

Equivalently:

$$\langle \mathcal{O} \rangle = \int \prod_{i=1}^{D} d\phi_i p(\phi) \mathcal{O}(\phi)$$

$p(\phi) = \dfrac{e^{-S(\phi)}}{Z}$ can be interpreted as a probability

Markov chain Monte Carlo

Markov chain Monte Carlo: Generate a chain starting from an arbitrary field:

$$\phi^0 \to \phi^1 \to \ldots \to \phi^k \to \ldots \to \phi^M$$

Call $T(\phi^k, \phi')$ the transition probability $\phi^k \to \phi'$

$\{\phi\}$ will converge to $p(\phi)$ [e.g. to $p(\phi)=e^{-S(\phi)}/Z$] if:

- *Ergodicity* is satisfied, i.e. $T^n(\phi, \phi') > 0$ for any ϕ, ϕ' for a finite n
- *Balance* is satisfied, i.e. $\int \prod_{i=1}^{D} d\phi_i p(\phi) T(\phi, \phi') = p(\phi')$

Markov chain Monte Carlo

Metropolis sampling

1. Draw an *update proposal* ϕ' from a distribution $\tilde{p}(\phi')$
2. Accept ϕ' as the next configuration in the Markov chain (ϕ^{k+1}) with probability:

$$\min\left(1, \frac{\tilde{p}(\phi^k)p(\phi')}{p(\phi^k)\tilde{p}(\phi')}\right)$$

3. Otherwise: $\phi^{k+1}=\phi^k$

Markov chain Monte Carlo

Metropolis sampling

1. Draw an *update proposal* ϕ' from a distribution $\tilde{p}(\phi')$
2. Accept ϕ' as the next configuration in the Markov chain (ϕ^{k+1}) with probability:

$$\min\left(1, \frac{\tilde{p}(\phi^k)p(\phi')}{p(\phi^k)\tilde{p}(\phi')}\right)$$

3. Otherwise: $\phi^{k+1} = \phi^k$

- Satisfies ergodicity and balance
- Allows drawing from an arbitrary distribution $\tilde{p}(\phi')$, e.g. normal or uniform
- Requires calculating: $p(\phi')/p(\phi^k) = e^{-[S(\phi')-S(\phi^k)]}$, i.e. Z cancels

Markov chain Monte Carlo

Metropolis sampling

1. Draw an *update proposal* ϕ' from a distribution $\tilde{p}(\phi')$
2. Accept ϕ' as the next configuration in the Markov chain (ϕ^{k+1}) with probability:

$$\min\left(1, \frac{\tilde{p}(\phi^k)p(\phi')}{p(\phi^k)\tilde{p}(\phi')}\right)$$

3. Otherwise: $\phi^{k+1}=\phi^k$

- Satisfies ergodicity and balance
- Allows drawing from an arbitrary distribution $\tilde{p}(\phi')$, e.g. normal or uniform
- Requires calculating: $p(\phi')/p(\phi^k) = e^{-[S(\phi')-S(\phi^k)]}$, i.e. Z cancels

---

- Since the configurations ϕ are distributed according to the desired $p(\phi)$:

$$\langle\mathcal{O}\rangle = \frac{1}{M}\sum_{i=1}^{M}\mathcal{O}(\phi^i)$$

and statistical errors scale like $\frac{1}{\sqrt{M}}$

Markov chain Monte Carlo

Metropolis sampling in Markov chain Monte Carlo

- *Acceptance rate*:
  - Ratio of accepted trials over total number of configurations
  - Usually can be tuneable

Markov chain Monte Carlo

Metropolis sampling in Markov chain Monte Carlo

- *Acceptance rate*:
  - Ratio of accepted trials over total number of configurations
  - Usually can be tuneable
- *Autocorrelation* $\rho(\tau)$:
  - Probability of having τ rejections in a row, with $\rho(0) = 1$
  - *Autocorrelation time* is loosely the value of τ for which $\rho(\tau) = 0$
  - More formally: $\tau_{\text{int}} = \dfrac{1}{2} + \sum_{\tau=1}^{\infty} \rho(\tau)$

Markov chain Monte Carlo

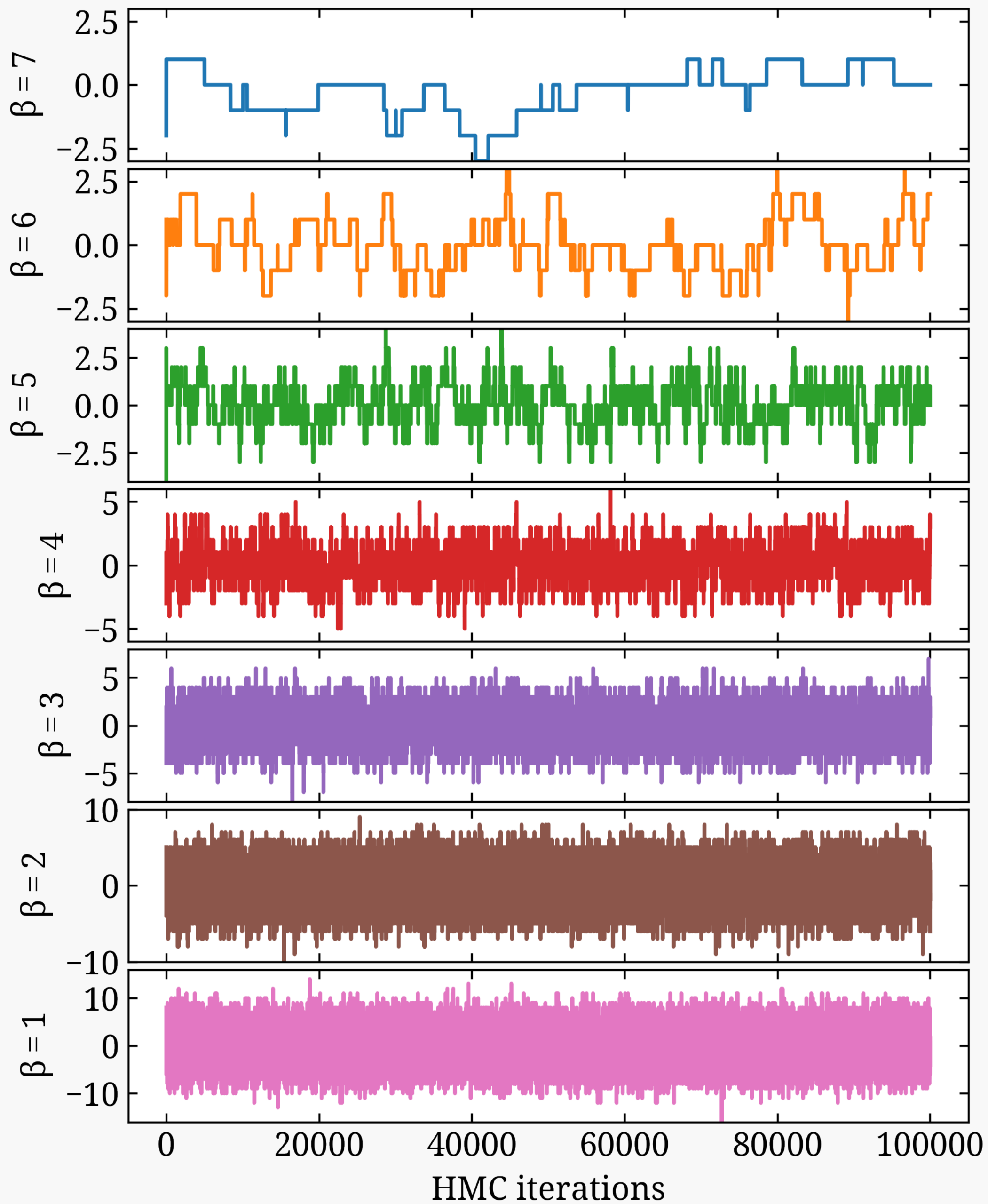
Metropolis sampling in Markov chain Monte Carlo

- *Acceptance rate:*
  - Ratio of accepted trials over total number of configurations
  - Usually can be tuneable
- *Autocorrelation* $\rho(\tau)$:
  - Probability of having τ rejections in a row, with $\rho(0) = 1$
  - *Autocorrelation time* is loosely the value of τ for which $\rho(\tau) = 0$
  - More formally: $\tau_{\text{int}} = \dfrac{1}{2} + \displaystyle\sum_{\tau=1}^{\infty} \rho(\tau)$
- *Critical slowing down*:
  - The divergence of τ_{int} as some parameters of the theory approach their critical value, e.g. as we approach a phase transition

Example: U(1) pure-gauge with HMC

2-dimensional U(1) gauge-theory

- Critical slowing down as $\beta \to \infty$



"Freezing" of topological charge:

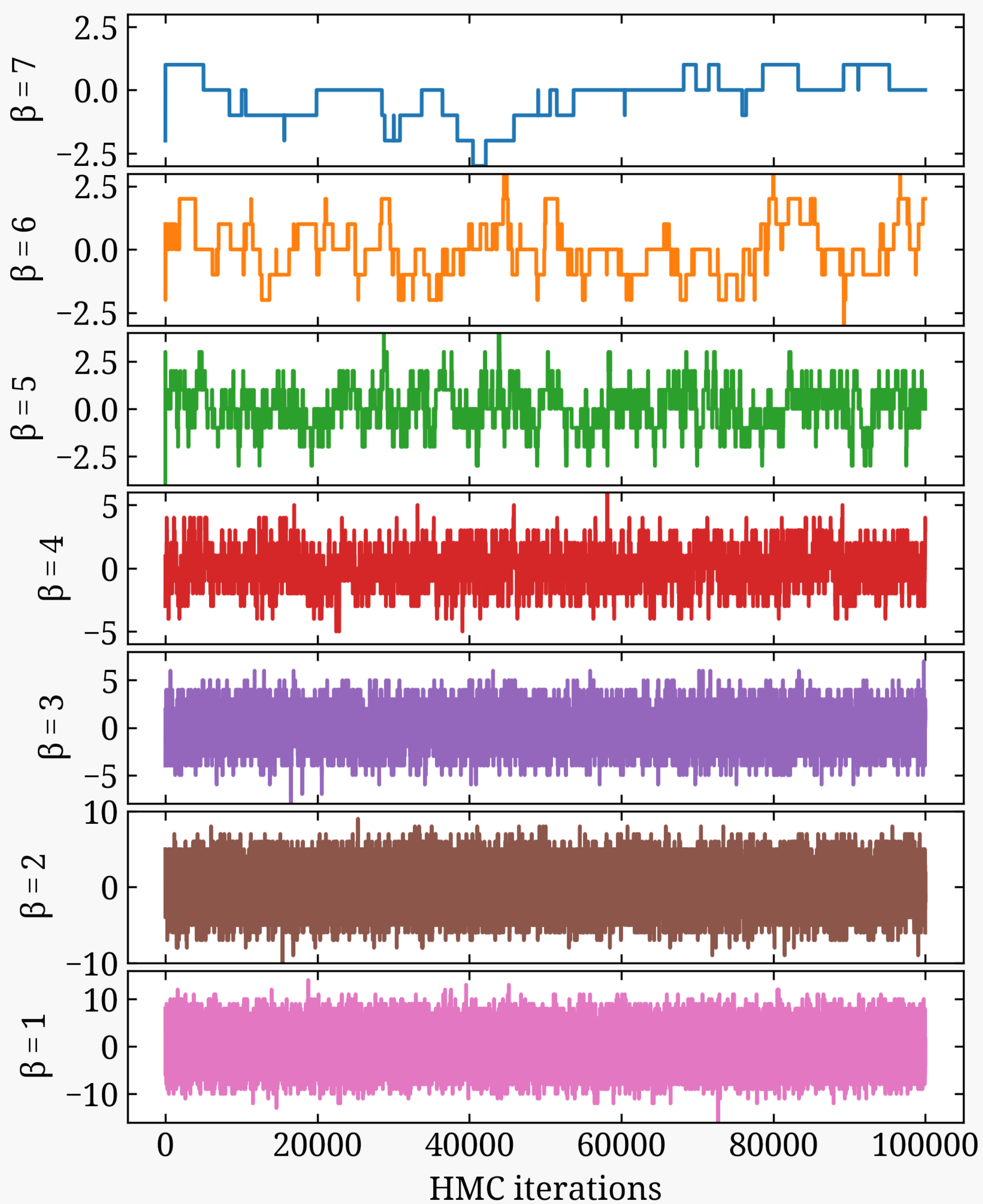
$$Q = \frac{1}{2\pi} \sum_{y} \arg P(y)$$

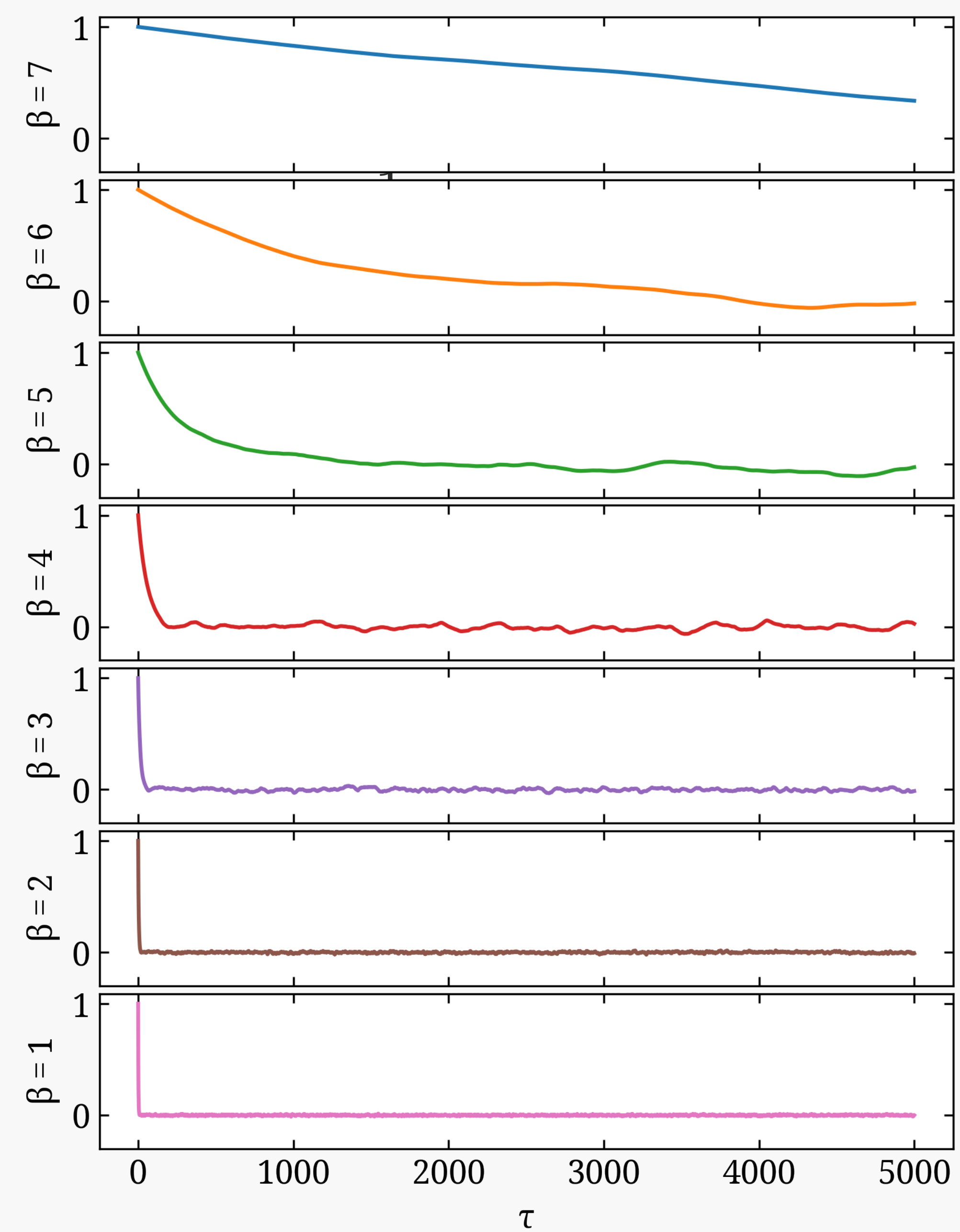
Example: U(1) pure-gauge with HMC

2-dimensional U(1) gauge-theory

- Critical slowing down as $\beta \to \infty$

$$\rho_{\mathcal{O}}(\tau) = \frac{\frac{1}{M-\tau}\sum_{i=1}^{M-\tau}(\mathcal{O}_i - \langle\mathcal{O}\rangle)(\mathcal{O}_{i+\tau} - \langle\mathcal{O}\rangle)}{\frac{1}{M}\sum_{i=1}^{M}(\mathcal{O}_i - \langle\mathcal{O}\rangle)^2}$$



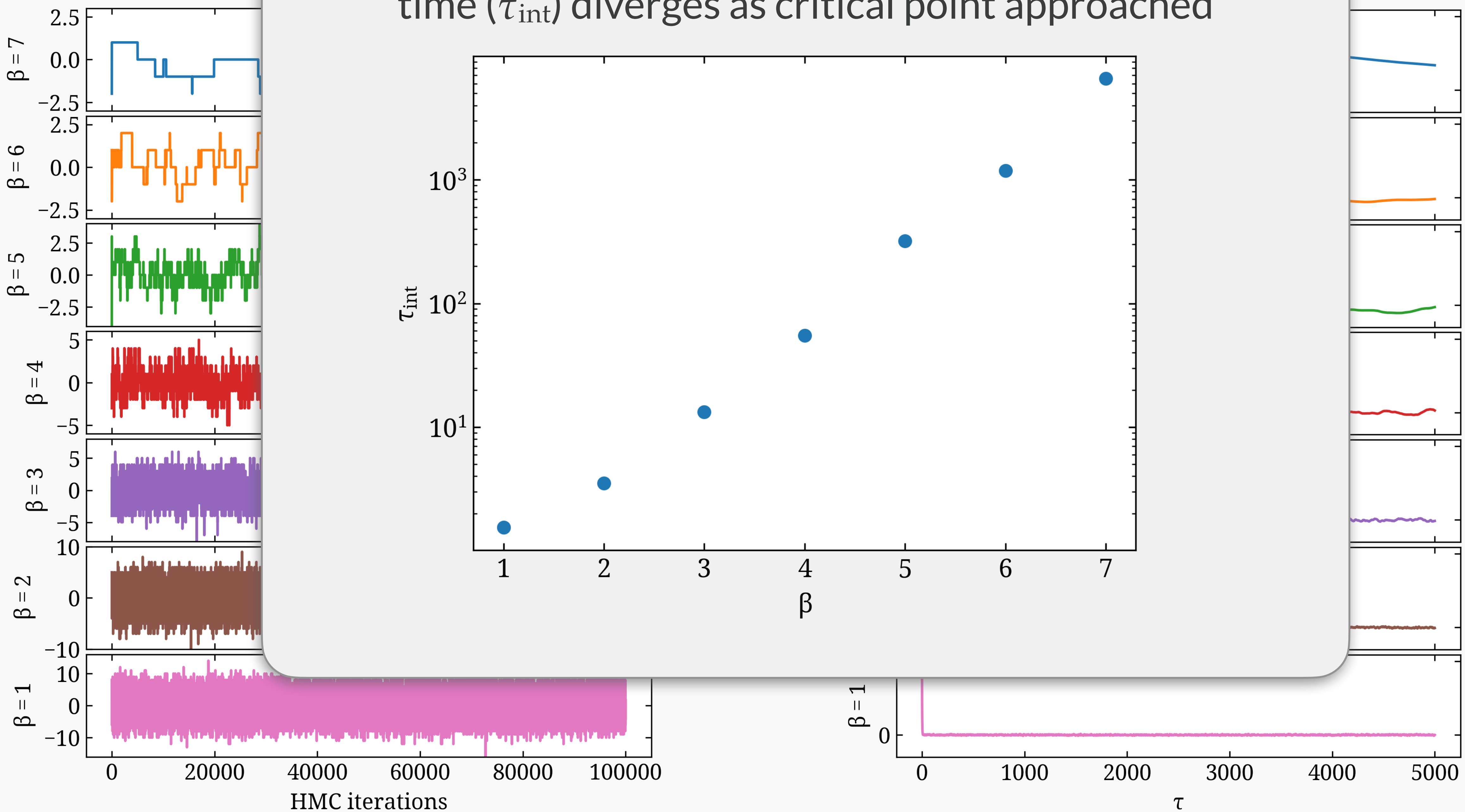


Example: U(1) pure-gauge with HMC

2-dimensional

- Critical slo

Critical slowing down: integrated autocorrelation time (τ_{int}) diverges as critical point approached



Markov chain Monte Carlo

Metropolis sampling

1. Draw an *update proposal* ϕ' from a distribution $\tilde{p}(\phi')$
2. Accept ϕ' as the next configuration in the Markov chain (ϕ^{k+1}) with probability:

$$\min\left(1, \frac{\tilde{p}(\phi^k)p(\phi')}{p(\phi^k)\tilde{p}(\phi')}\right)$$

3. Otherwise: $\phi^{k+1}=\phi^k$

Flow-based generative models for Markov chain Monte Carlo

- Use neural networks to provide proposals ϕ'
- Train neural network to yield $\tilde{p}(\phi')$ as close to $p(\phi')$ as possible

Flow-based generative model

Prerequisite: reminder on transforming distributions

Flow-based generative model

Prerequisite: reminder on transforming distributions

If X is a random variable with probability density function $f(x)$, then for $Y{=}h(X)$, the probability distribution of Y is:

$$p(y) = f\left(h^{-1}(y)\right)\left|\frac{d}{dy}h^{-1}(y)\right|$$

Flow-based generative model

Prerequisite: reminder on transforming distributions

If X is a random variable with probability density function $f(x)$, then for $Y=h(X)$, the probability distribution of Y is:

$$p(y) = f\left(h^{-1}(y)\right) \left| \frac{d}{dy} h^{-1}(y) \right|$$

Trivial example, $f(x) = 1$ (uniform) and $y = h(X) =$ -$\ln(X)$

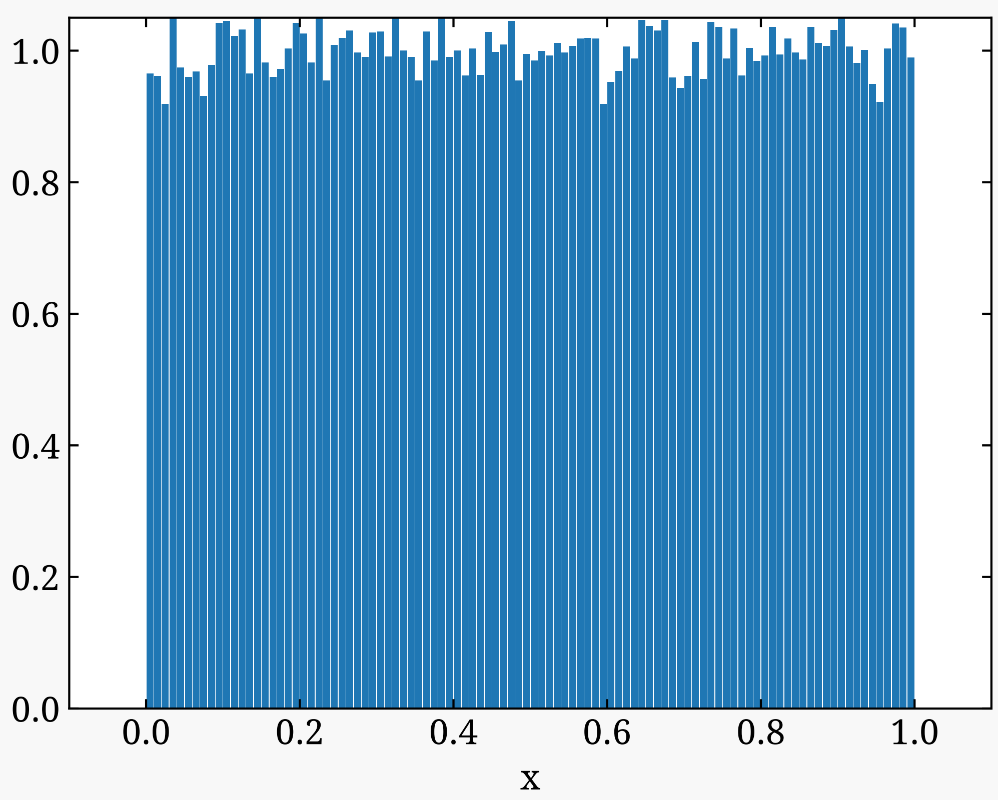
Flow-based generative model

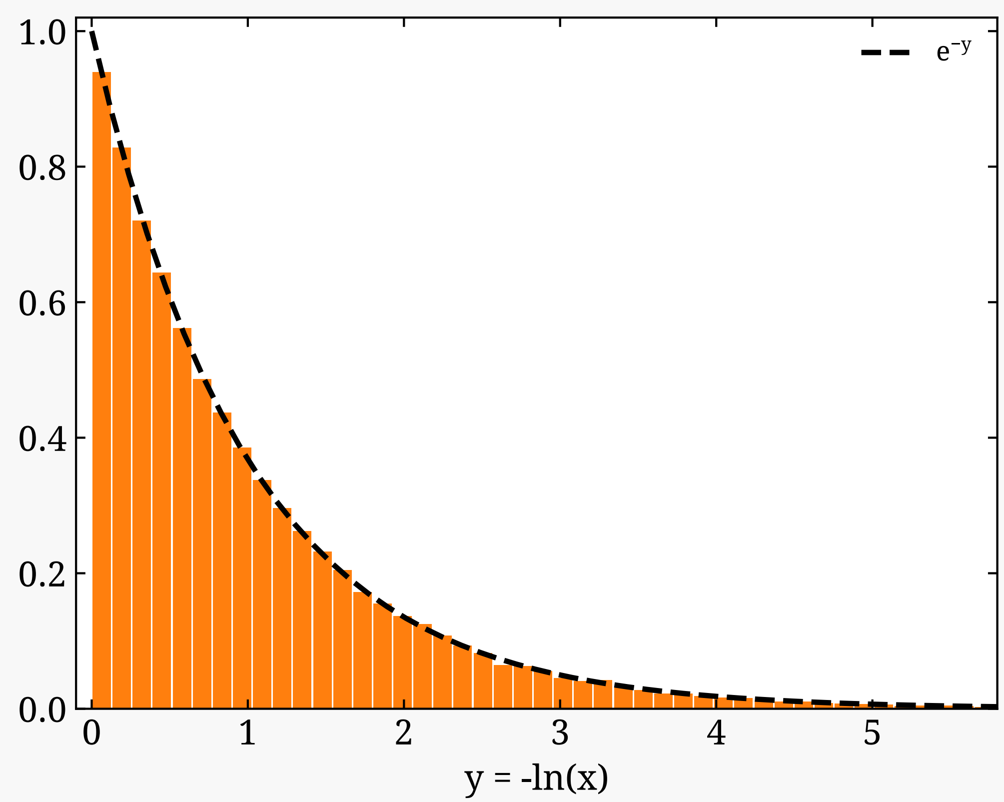
Prerequisite: reminder on transforming distributions

If X is a random variable with probability density function $f(x)$, then for $Y=h(X)$, the probability distribution of Y is:

$$p(y) = f\left(h^{-1}(y)\right) \left| \frac{d}{dy} h^{-1}(y) \right|$$

Trivial example, $f(x) = 1$ (uniform) and $y = h(X) = -\ln(X)$





Flow-based generative model

Prerequisite: reminder on transforming distributions

Generalizes to vectors of random variables:

$$p(\vec{y}) = f(h_a^{-1}(\vec{y})) \left| \det \frac{dh_i^{-1}(\vec{y})}{dy_j} \right|$$

Flow-based generative model

Prerequisite: reminder on transforming distributions

Generalizes to vectors of random variables:

$$p(\vec{y}) = f(h_a^{-1}(\vec{y})) \left| \det \frac{dh_i^{-1}(\vec{y})}{dy_j} \right|$$

E.g. the well-known Box-Muller transformation for transforming two uniformly distributed random variables u to two normally distributed random variables z

$$\vec{u} = \begin{pmatrix} u_0 \\ u_1 \end{pmatrix} \qquad \vec{z} = \begin{pmatrix} z_0 \\ z_1 \end{pmatrix} = h_a(\vec{u}) = \begin{pmatrix} \sqrt{-2 \ln u_0} \cos(2\pi u_1) \\ \sqrt{-2 \ln u_0} \sin(2\pi u_1) \end{pmatrix}$$

$$p(\vec{z}) = \left| \det \frac{dh_i^{-1}(\vec{z})}{dz_j} \right| = \prod_{i=0}^{1} \frac{1}{\sqrt{2\pi}} e^{-\frac{z_i^2}{2}}$$

Flow-based generative model

The idea: given a set of random variables z drawn from a distribution $r(z)$ which we know how to sample from, find a transformation $\phi = f^{-1}(z)$ such that:

$$\tilde{p}(\phi) = r(f(\phi)) \left| \det \frac{d}{d\phi} f(\phi) \right| \simeq \frac{e^{-S(\phi)}}{Z}$$

Flow-based generative model

The idea: given a set of random variables z drawn from a distribution $r(z)$ which we know how to sample from, find a transformation $\phi = f^{-1}(z)$ such that:

$$\tilde{p}(\phi) = r(f(\phi)) \left| \det \frac{d}{d\phi} f(\phi) \right| \simeq \frac{e^{-S(\phi)}}{Z}$$

- Requirements:
  - $f(\phi)$ must be invertible
  - Derivative must exist and Jacobean must be cheap to compute

Flow-based generative model

The idea: given a set of random variables z drawn from a distribution $r(z)$ which we know how to sample from, find a transformation $\phi = f^{-1}(z)$ such that:

$$\tilde{p}(\phi) = r(f(\phi)) \left| \det \frac{d}{d\phi} f(\phi) \right| \simeq \frac{e^{-S(\phi)}}{Z}$$

- Requirements:
  - $f(\phi)$ must be invertible
  - Derivative must exist and Jacobean must be cheap to compute

---

- *Real non-volume preserving flow*:

$$g(\phi) = \begin{cases} z_a = \phi_a \\ z_b = \phi_b \odot e^{s(\phi_a)} + t(\phi_a) \end{cases}$$

- z_a, z_b and ϕ_a, ϕ_b are half-vectors of z and ϕ (e.g. even and odd elements)
- s and t are neural networks that have inputs and outputs half the number of elements in ϕ and z

Flow-based generative model

- *Real non-volume preserving flow*:

$$g(\phi) = \begin{cases} z_a = \phi_a \\ z_b = \phi_b \odot e^{s(\phi_a)} + t(\phi_a) \end{cases}$$

  - z_a, z_b and ϕ_a, ϕ_b are half-vectors of z and ϕ (e.g. even and odd elements)
  - s and t are neural networks that have inputs and outputs half the number of elements in ϕ and z

$$g^{-1}(z) = \begin{cases} \phi_a = z_a \\ \phi_b = (z_b - t(z_a)) \odot e^{-s(z_a)} \end{cases}$$

  - Note that these functions are invertible *without requiring inverting the neural networks* s and t

Flow-based generative model

- *Real non-volume preserving flow*:

$$g(\phi) = \begin{cases} z_a = \phi_a \\ z_b = \phi_b \odot e^{s(\phi_a)} + t(\phi_a) \end{cases}$$

  - z_a, z_b and ϕ_a, ϕ_b are half-vectors of z and ϕ (e.g. even and odd elements)
  - s and t are neural networks that have inputs and outputs half the number of elements in ϕ and z

$$g^{-1}(z) = \begin{cases} \phi_a = z_a \\ \phi_b = (z_b - t(z_a)) \odot e^{-s(z_a)} \end{cases}$$

  - Note that these functions are invertible *without requiring inverting the neural networks* s and t

$$\left|\det \frac{dg(\phi)}{d\phi}\right| = \left|\det \begin{pmatrix} \frac{dg(\phi)_a}{d\phi_a} & \frac{dg(\phi)_b}{d\phi_a} \\ \frac{dg(\phi)_a}{d\phi_b} & \frac{dg(\phi)_b}{d\phi_b} \end{pmatrix}\right| = \left|\det \begin{pmatrix} 1 & \frac{dg(\phi)_b}{d\phi_a} \\ 0 & e^{s(\phi_a)} \end{pmatrix}\right| = \prod_{i \in a} e^{s(\phi_a)_i}$$

  - Determinant is easy to compute

Flow-based generative model

- *Real non-volume preserving flow*:

$$g(\phi) = \begin{cases} z_a = \phi_a \\ z_b = \phi_b \odot e^{s(\phi_a)} + t(\phi_a) \end{cases}$$

  - z_a, z_b and ϕ_a, ϕ_b are half-vectors of z and ϕ (e.g. even and odd elements)
  - s and t are neural networks that have inputs and outputs half the number of elements in ϕ and z

$$g^{-1}(z) = \begin{cases} \phi_a = z_a \\ \phi_b = (z_b - t(z_a)) \odot e^{-s(z_a)} \end{cases}$$

  - Note that these functions are invertible *without requiring inverting the neural networks* s and t

$$\left|\det \frac{dg(\phi)}{d\phi}\right| = \left|\det \begin{pmatrix} \frac{dg(\phi)_a}{d\phi_a} & \frac{dg(\phi)_b}{d\phi_a} \\ \frac{dg(\phi)_a}{d\phi_b} & \frac{dg(\phi)_b}{d\phi_b} \end{pmatrix}\right| = \left|\det \begin{pmatrix} 1 & \frac{dg(\phi)_b}{d\phi_a} \\ 0 & e^{s(\phi_a)} \end{pmatrix}\right| = \prod_{i\in a} e^{s(\phi_a)_i}$$

  - Determinant is easy to compute
  - It's also easy to chain these functions:

$$f(\phi) = g_1(g_2(\ldots g_n(\phi)\ldots)) \Rightarrow f^{-1}(z) = g_n^{-1}(\ldots g_2^{-1}(g_1^{-1}(z))\ldots)$$

Flow-based generative model

- Real NVP flow for Markov chain Monte Carlo:

$$g(\phi) = \begin{cases} z_a = \phi_a \\ z_b = \phi_b \odot e^{s(\phi_a)} + t(\phi_a) \end{cases}$$

Flow-based generative model

- Real NVP flow for Markov chain Monte Carlo:

$$g(\phi) = \begin{cases} z_a = \phi_a \\ z_b = \phi_b \odot e^{s(\phi_a)} + t(\phi_a) \end{cases}$$

  - Define a function to transform random variables z from a known distribution $r(z)$:

$$f^{-1}(z) = g_n^{-1}(\ldots g_2^{-1}(g_1^{-1}(z))\ldots)$$

Flow-based generative model

- Real NVP flow for Markov chain Monte Carlo:

$$g(\phi) = \begin{cases} z_a = \phi_a \\ z_b = \phi_b \odot e^{s(\phi_a)} + t(\phi_a) \end{cases}$$

  – Define a function to transform random variables z from a known distribution $r(z)$:

$$f^{-1}(z) = g_n^{-1}(\ldots g_2^{-1}(g_1^{-1}(z))\ldots)$$

  – With $\phi = f^{1}(z)$, train the neural networks s_i and t_i in each g_i so that:

$$\tilde{p}(\phi) = r(f(\phi)) \left| \det \frac{d}{d\phi} f(\phi) \right| \simeq \frac{e^{-S(\phi)}}{Z}$$

Flow-based generative model

- Real NVP flow for Markov chain Monte Carlo:

$$g(\phi) = \begin{cases} z_a = \phi_a \\ z_b = \phi_b \odot e^{s(\phi_a)} + t(\phi_a) \end{cases}$$

– Define a function to transform random variables z from a known distribution $r(z)$:

$$f^{-1}(z) = g_n^{-1}(...g_2^{-1}(g_1^{-1}(z))...)$$

– With $\phi = f^{-1}(z)$, train the neural networks s_i and t_i in each g_i so that:

$$\tilde{p}(\phi) = r(f(\phi)) \left| \det \frac{d}{d\phi} f(\phi) \right| \simeq \frac{e^{-S(\phi)}}{Z}$$

– After training $\phi = f^{-1}(z)$ can be used to generate an arbitrary number of new fields. These should be *approximately* distributed according to the action.

– To ensure the right distribution, start from one and accept the next one according to Metropolis accept/reject algorithm.

Flow-based generative model

- How does one train (i.e. which function is to be minimised)?

– The *loss function* needs to reflect how close the output of the model is to the desired distribution

– Shifted Kullback-Leibler (KL) divergence:

$$L(\tilde{p}) = \int \prod_j d\phi_j \tilde{p}(\phi)(\log \tilde{p}(\phi) - \log p(\phi) - \log Z)$$

$$\hat{L}(\tilde{p}) = \frac{1}{M} \sum_{j=1}^{M} (\log \tilde{p}(\phi) + S(\phi))$$

– Minimum of loss function is bounded by $-\log Z$

Flow-based generative model

- Trivial exercise: transform uniform to normal distribution
  - Two affine layers (g_1, g_2)
  - In each affine layer, s and t have two hidden layers with 64 nodes each

Flow-based generative model

- Trivial exercise: transform uniform to normal distribution
  - Two affine layers (g_1, g_2)
  - In each affine layer, s and t have two hidden layers with 64 nodes each

$$\phi = g_2^{-1}(g_1^{-1}(u))$$

Flow-based generative model

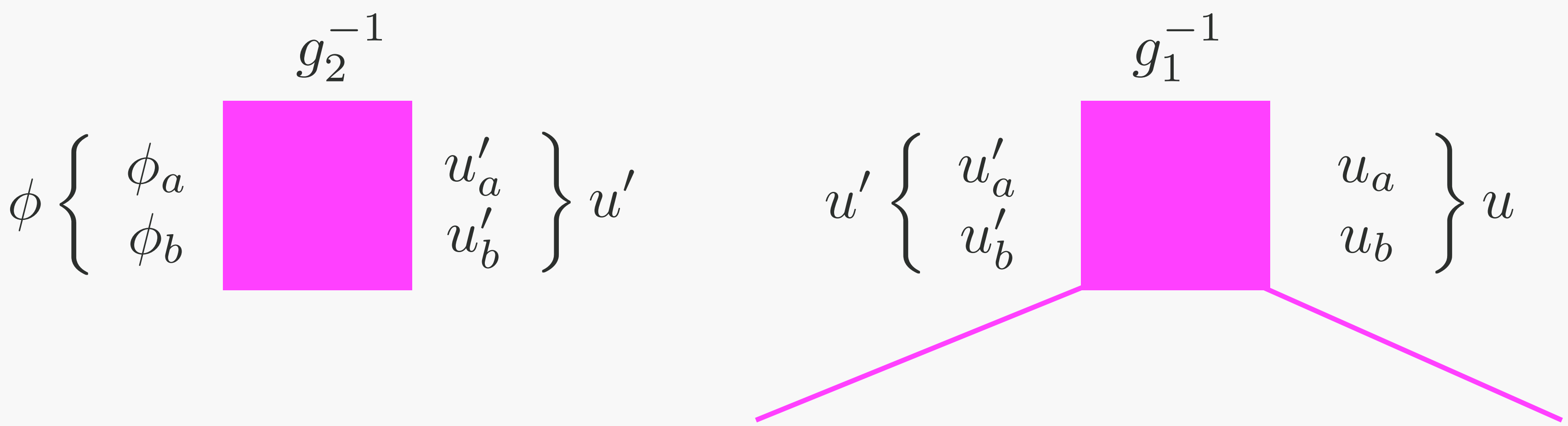
- Trivial exercise: transform uniform to normal distribution
  - Two affine layers (g_1, g_2)
  - In each affine layer, s and t have two hidden layers with 64 nodes each

$$\phi = g_2^{-1}(g_1^{-1}(u))$$

Flow-based generative model

- Trivial exercise: transform uniform to normal distribution
  - Two affine layers (g_1, g_2)
  - In each affine layer, s and t have two hidden layers with 64 nodes each

$$\phi = g_2^{-1}(g_1^{-1}(u))$$

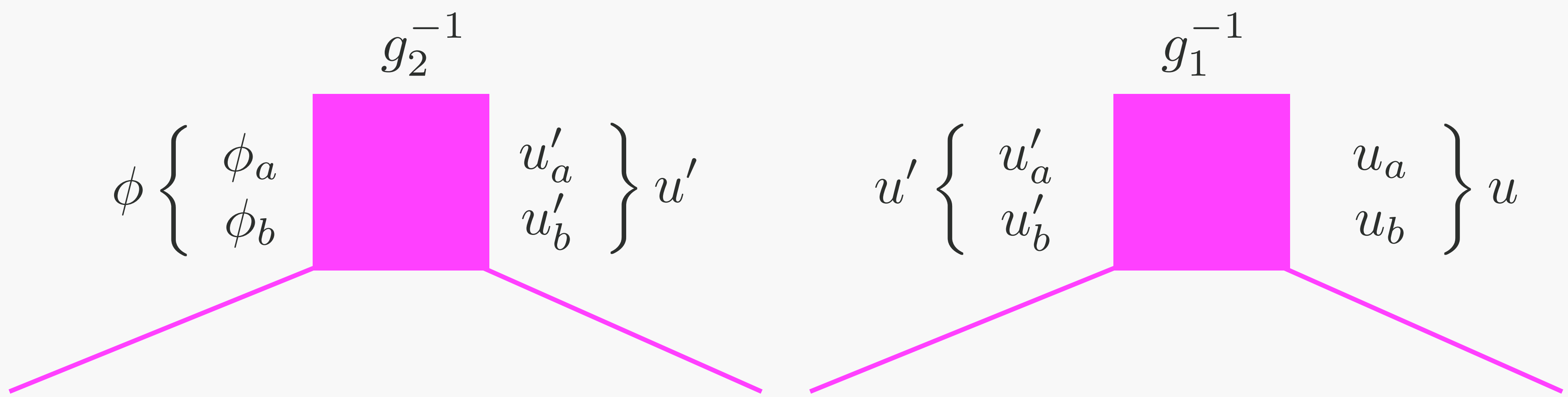


$$u'_a = u_a$$
$$u'_b = (u_b - t_1(u_a)) \odot e^{-s_1(u_a)}$$

Flow-based generative model

- Trivial exercise: transform uniform to normal distribution
- Two affine layers (g_1, g_2)
- In each affine layer, s and t have two hidden layers with 64 nodes each

$$\phi = g_2^{-1}(g_1^{-1}(u))$$



$$\phi_a = (u'_a - t_2(u'_b)) \odot e^{-s_2(u'_b)}$$
$$\phi_b = u'_b$$

$$u'_a = u_a$$
$$u'_b = (u_b - t_1(u_a)) \odot e^{-s_1(u_a)}$$

Flow-based generative model

- Trivial exercise: transform uniform to normal distribution
  - Two affine layers (g_1, g_2)
  - In each affine layer, s and t have two hidden layers with 64 nodes each

$$\phi = g_2^{-1}(g_1^{-1}(u))$$

$$g_2^{-1} \qquad\qquad\qquad\qquad g_1^{-1}$$

$$\phi \left\{ \begin{matrix} \phi_a \\ \phi_b \end{matrix} \right. \quad \left. \begin{matrix} u'_a \\ u'_b \end{matrix} \right\} u' \qquad\qquad u' \left\{ \begin{matrix} u'_a \\ u'_b \end{matrix} \right. \quad \left. \begin{matrix} u_a \\ u_b \end{matrix} \right\} u$$

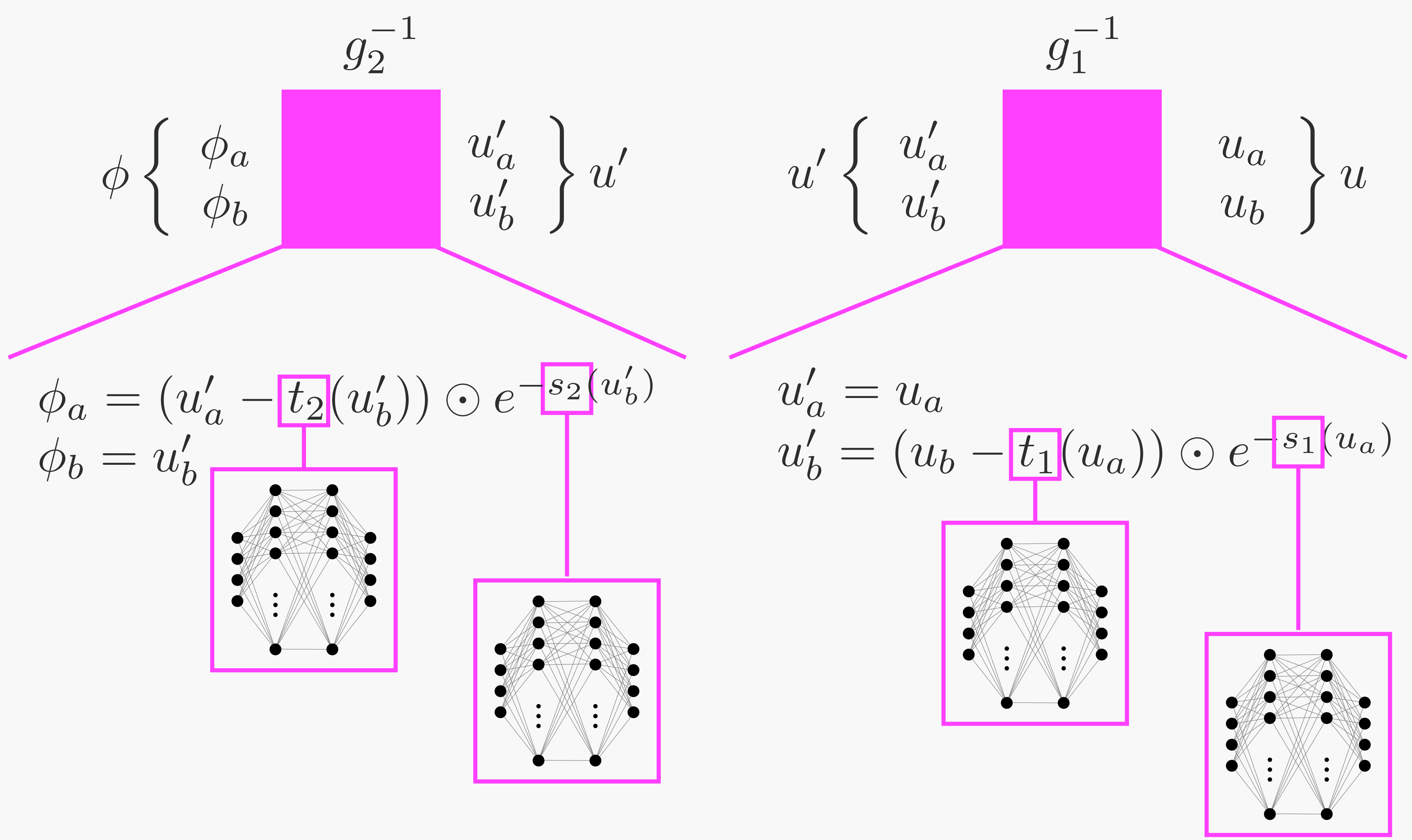
$$\begin{aligned} \phi_a &= (u'_a - \boxed{t_2}(u'_b)) \odot e^{-\boxed{s_2}(u'_b)} \\ \phi_b &= u'_b \end{aligned}$$

$$\begin{aligned} u'_a &= u_a \\ u'_b &= (u_b - \boxed{t_1}(u_a)) \odot e^{-\boxed{s_1}(u_a)} \end{aligned}$$

Flow-based generative model

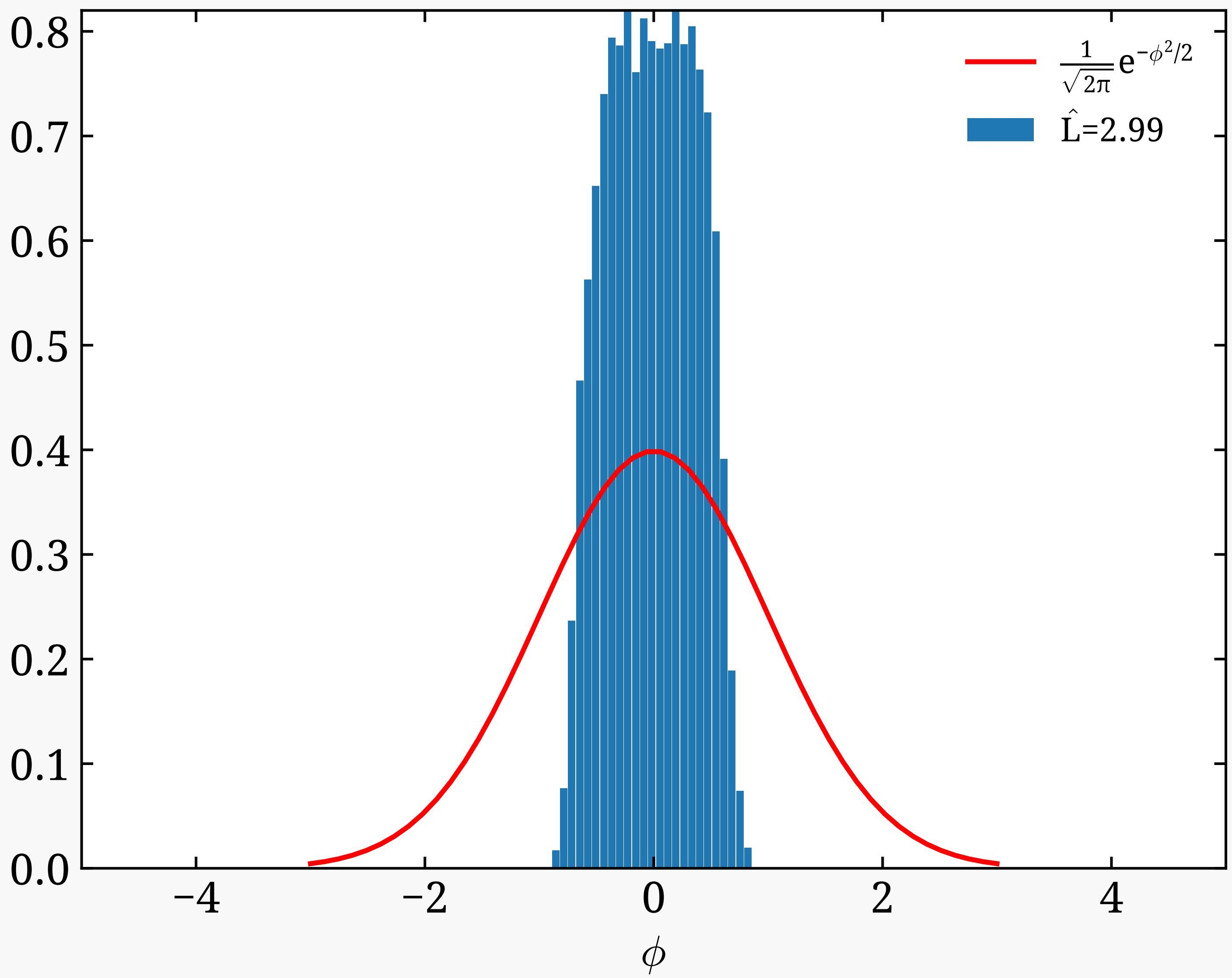
- Trivial exercise: transform uniform to normal distribution
  - Two affine layers (g_1, g_2)
  - In each affine layer, s and t have two hidden layers with 64 nodes each

$$\phi = g_2^{-1}(g_1^{-1}(u))$$



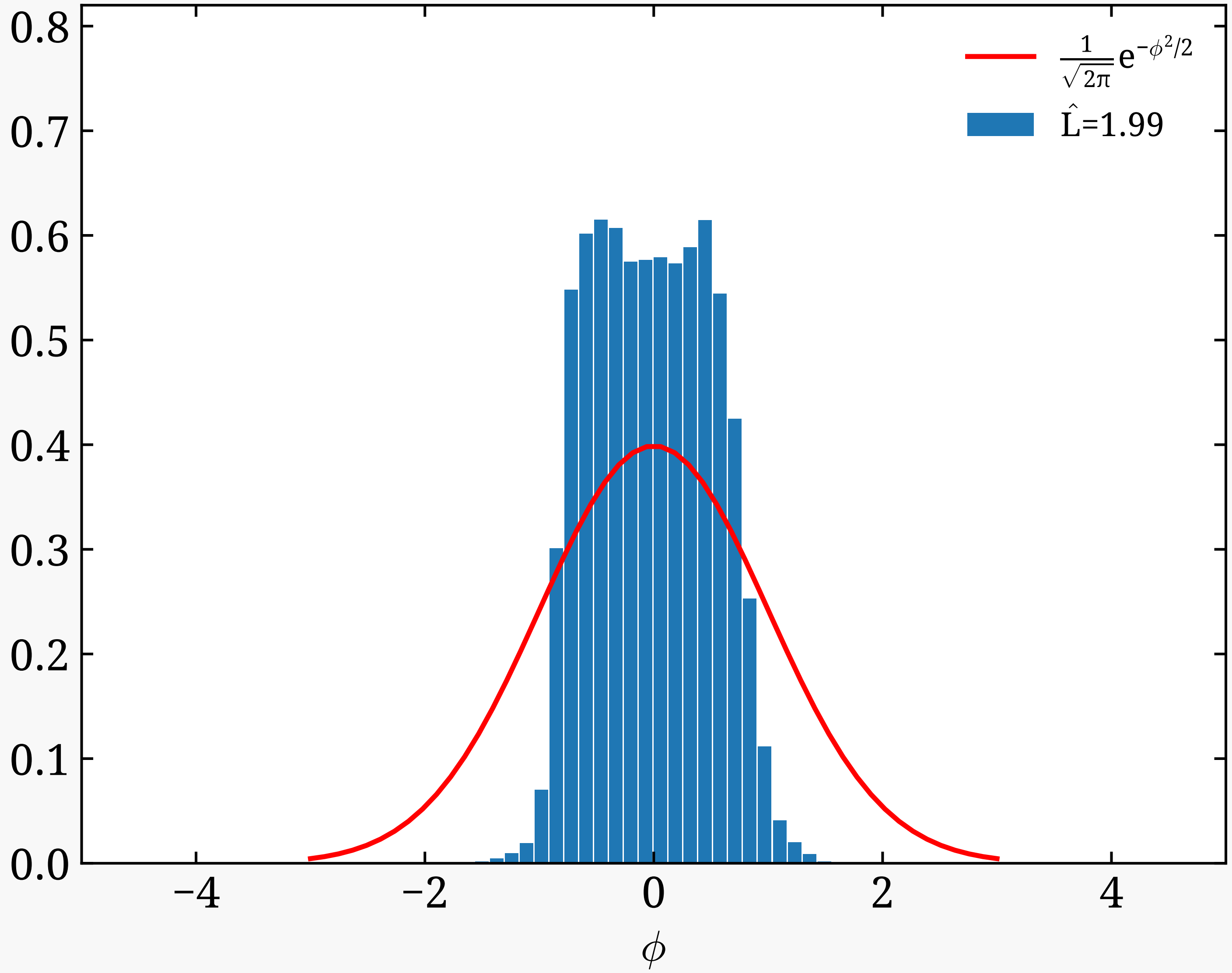
Flow-based generative model

- Transform uniform to normal distribution
  - Two affine layers (g_1, g_2)
  - In each affine layer, s and t have two hidden layers with 64 nodes each



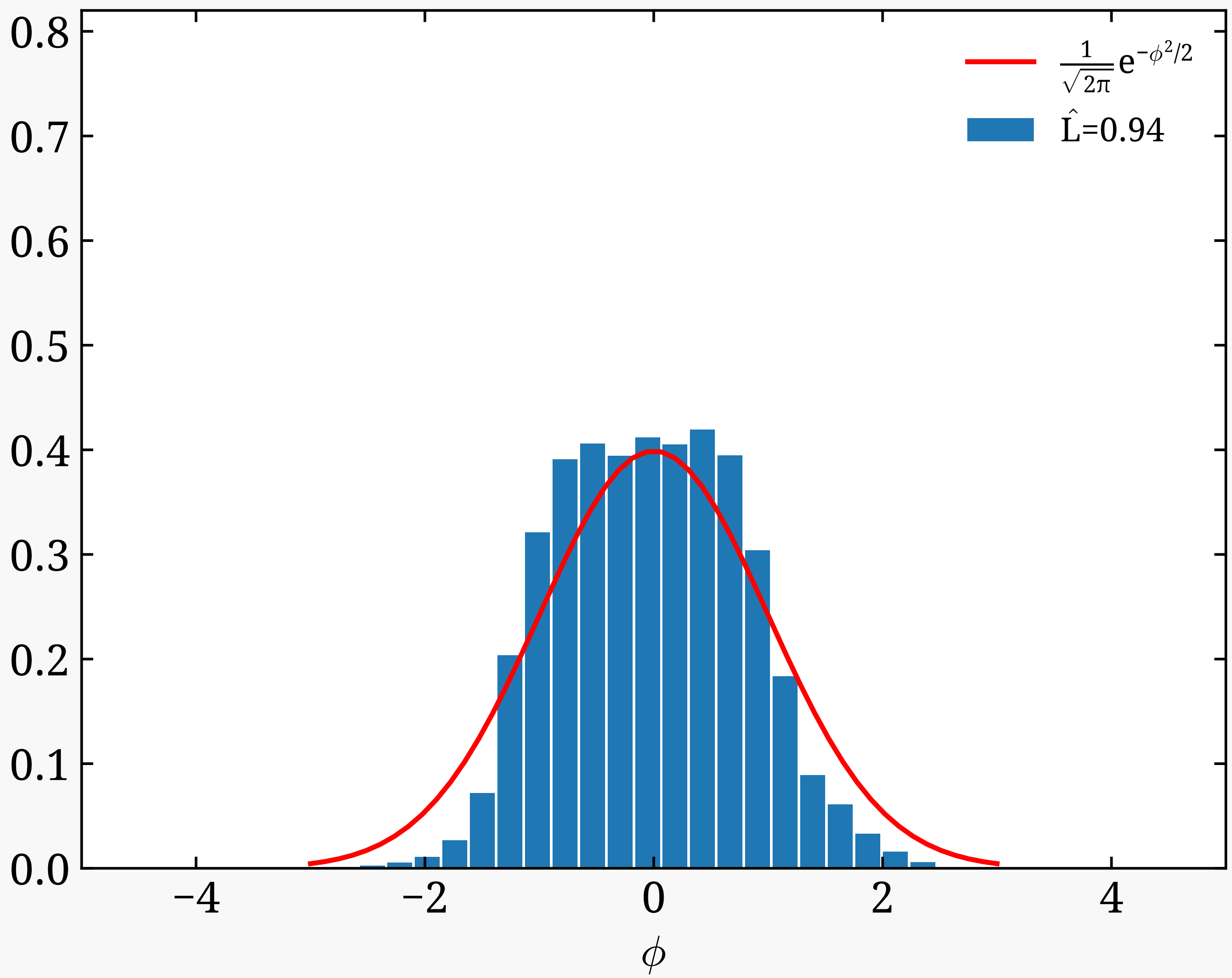
Flow-based generative model

- Transform uniform to normal distribution
  - Two affine layers (g_1, g_2)
  - In each affine layer, s and t have two hidden layers with 64 nodes each



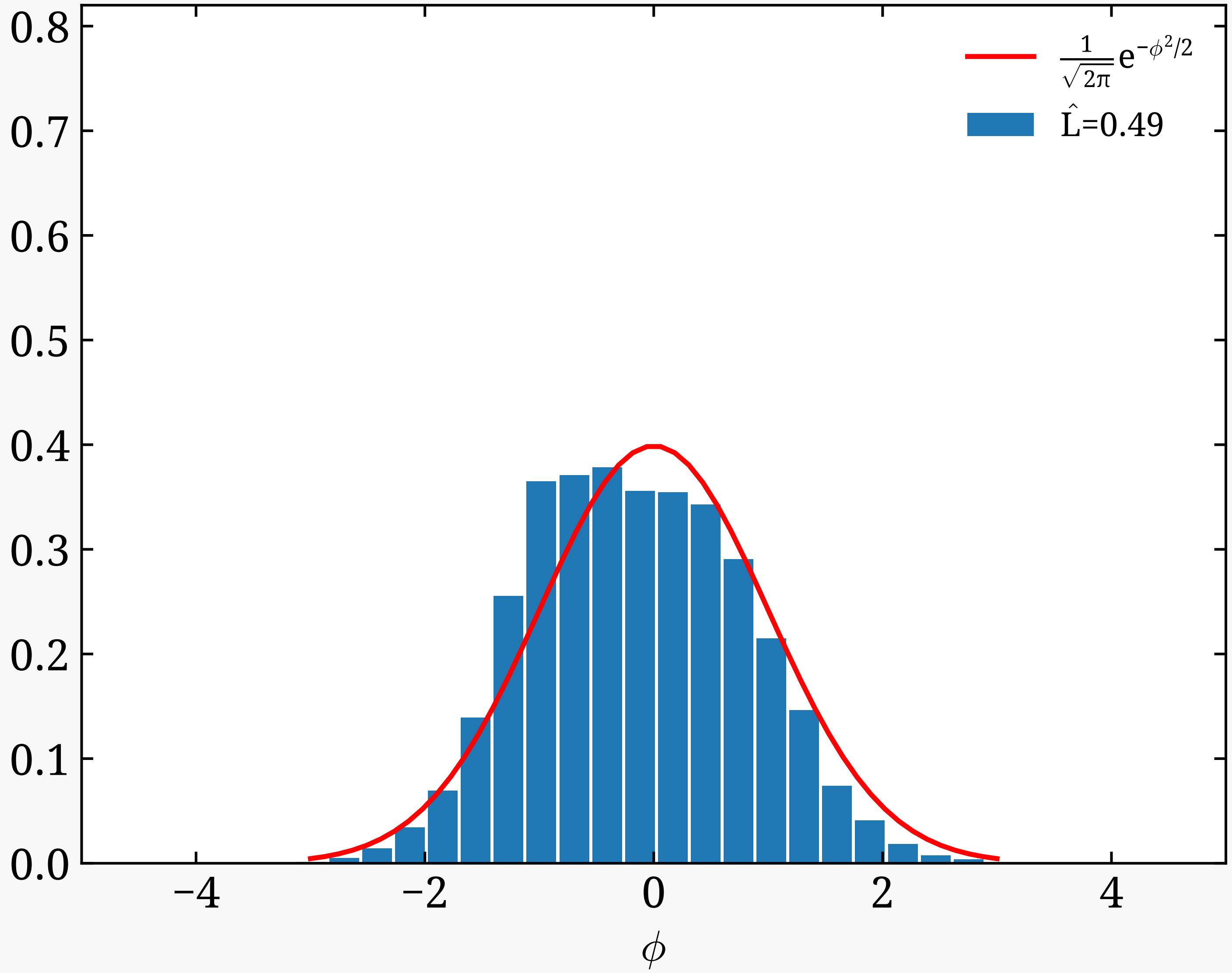
Flow-based generative model

- Transform uniform to normal distribution
  - Two affine layers (g_1, g_2)
  - In each affine layer, s and t have two hidden layers with 64 nodes each



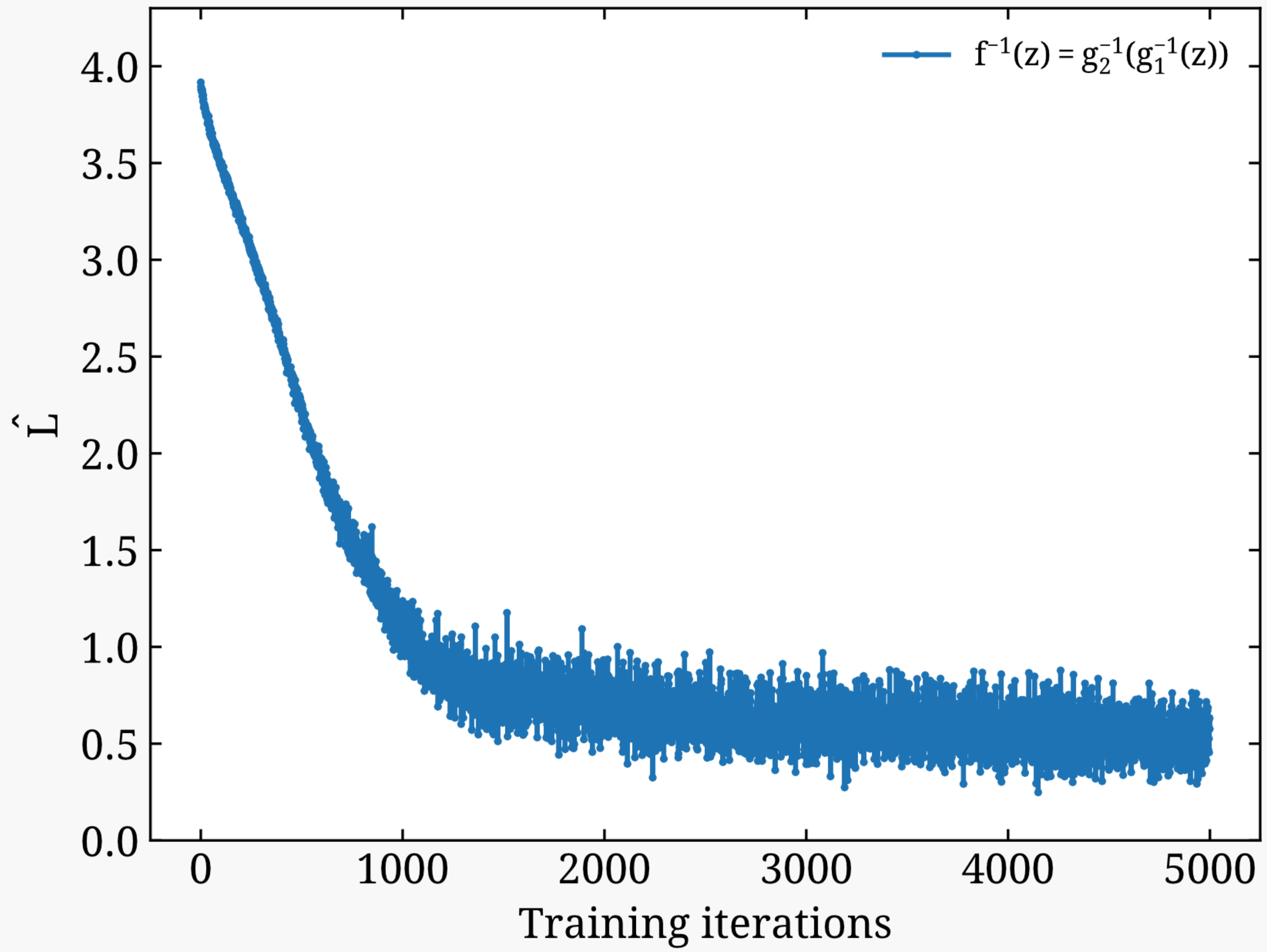
Flow-based generative model

- Transform uniform to normal distribution
  - Two affine layers (g_1, g_2)
  - In each affine layer, s and t have two hidden layers with 64 nodes each



Flow-based generative model

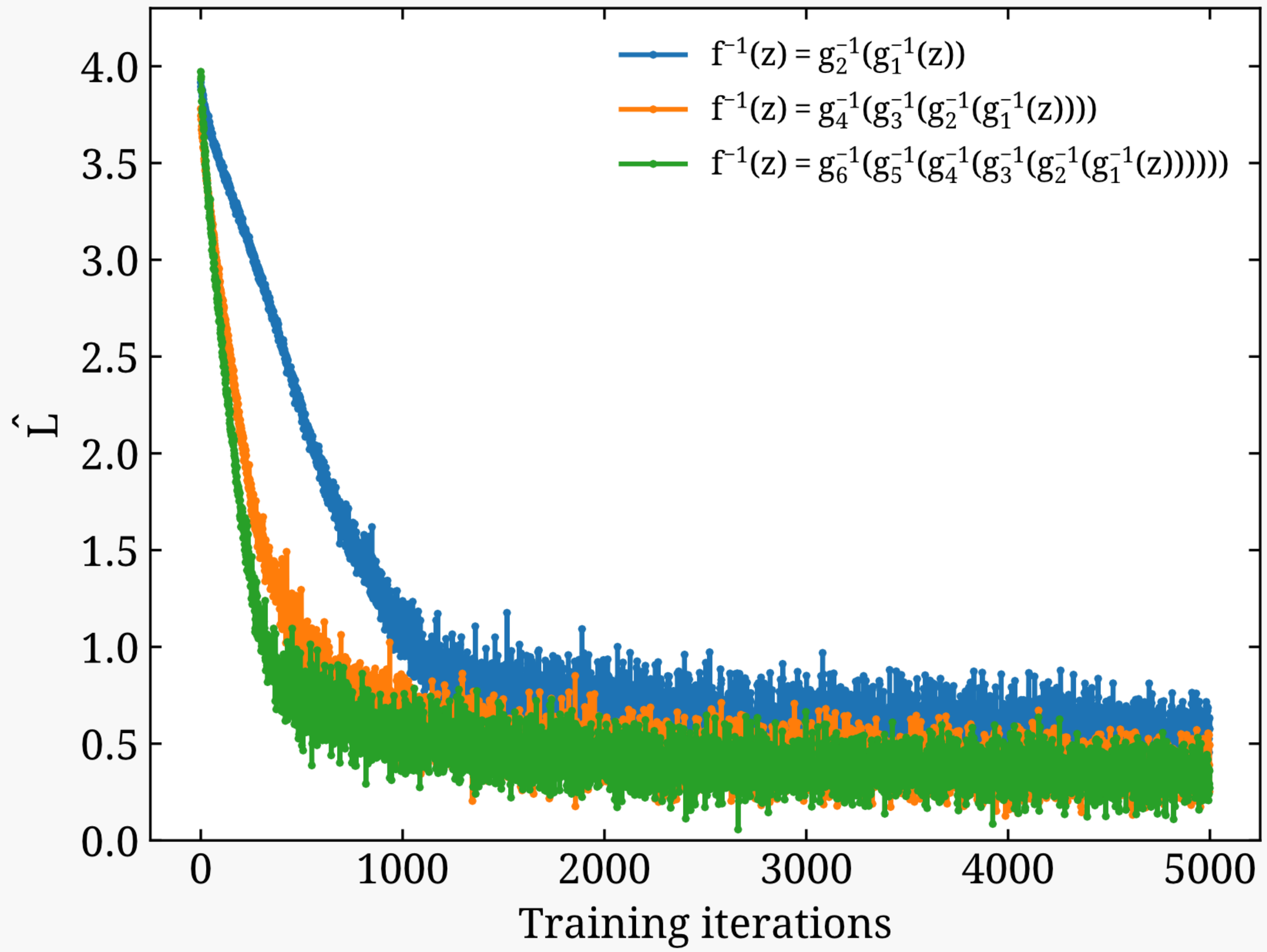
- Transform uniform to normal distribution
  - Two affine layers (g_1, g_2)
  - In each affine layer, s and t have two hidden layers with 64 nodes each



  - Loss function stagnates after some number of iterations

Flow-based generative model

- Transform uniform to normal distribution
  - Two affine layers (g_1, g_2)
  - In each affine layer, s and t have two hidden layers with 64 nodes each



- Increasing number of affine layers allows for a smaller loss function

ϕ^4 model

Application to ϕ^4 model as in arXiv:1904.12072

2-dimensional scalar field theory with the action:

$$S(\phi) = \sum_{i=1}^{L}\sum_{j=1}^{L} \phi_{i,j}(4\phi_{i,j} - \phi_{i-1,j} - \phi_{i+1,j} - \phi_{i,j-1} - \phi_{i,j+1}) + m^2\phi_{i,j}^2 + \lambda\phi_{i,j}^4$$

Five choices of the parameters considered:

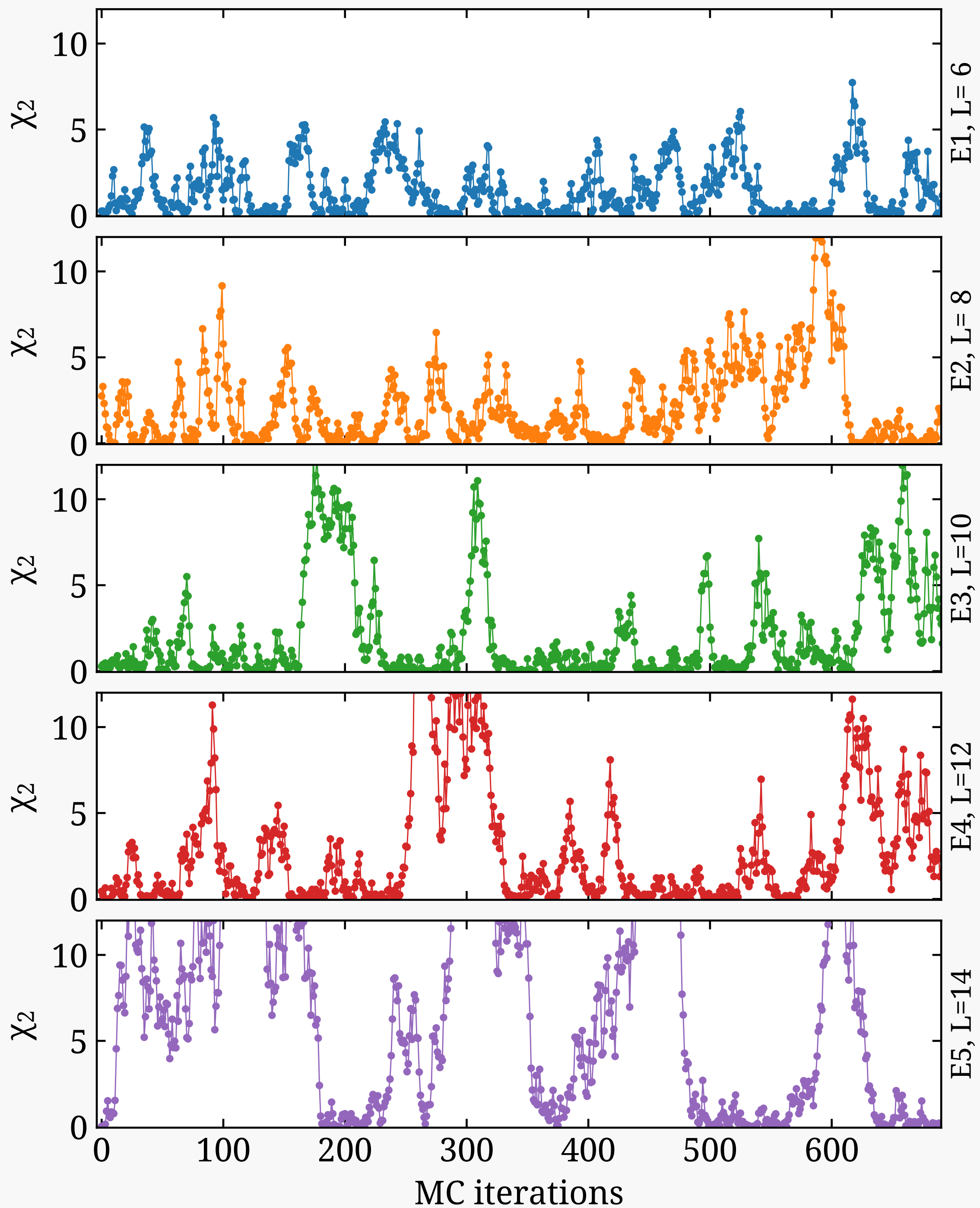
	E1	E2	E3	E4	E5
L	6	8	10	12	14
m^2	-4	-4	-4	-4	-4
λ	6.975	6.008	5.550	5.276	5.113

The parameters are chosen such that $m_p L$ is constant

⇒ as $L \to \infty$, $m_p \to 0$, a critical point.

Example: ϕ^4 model

Example of critical slowing down



Observable is the *two-point susceptibility*

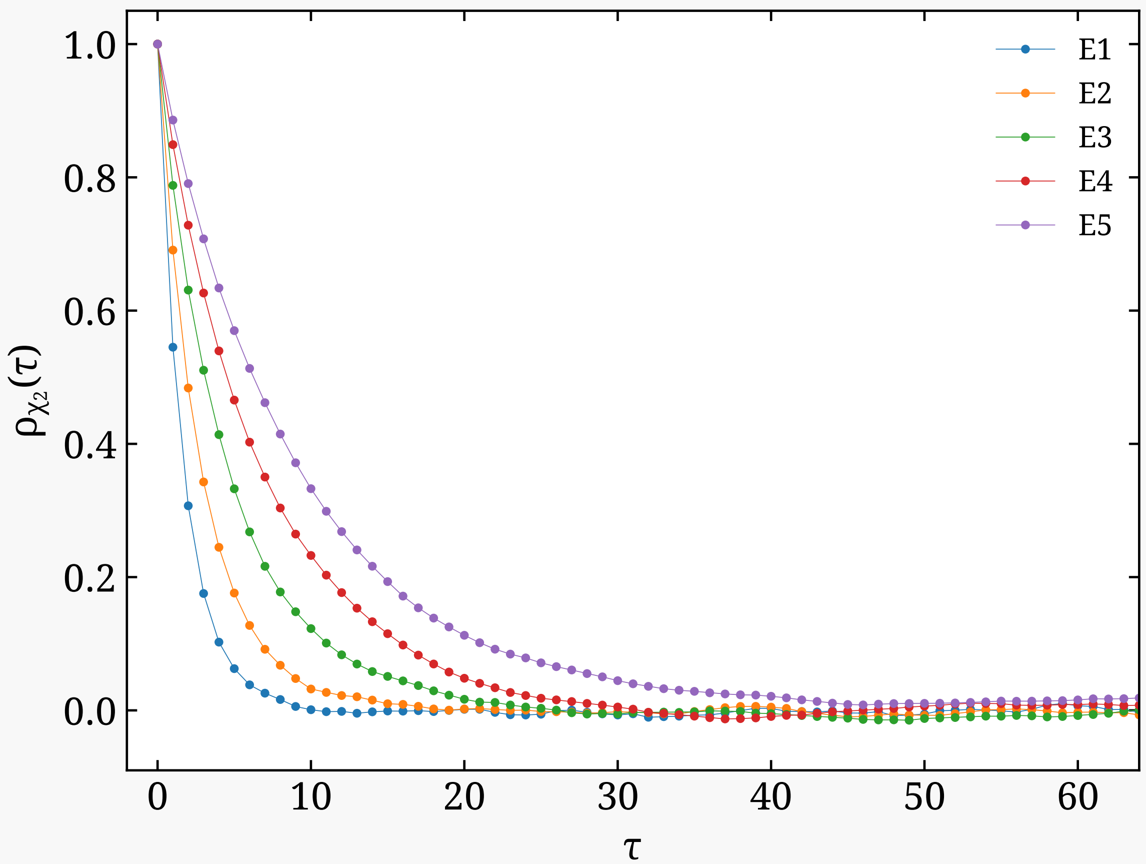
$$\chi_2 = \sum_x G_c(x)$$

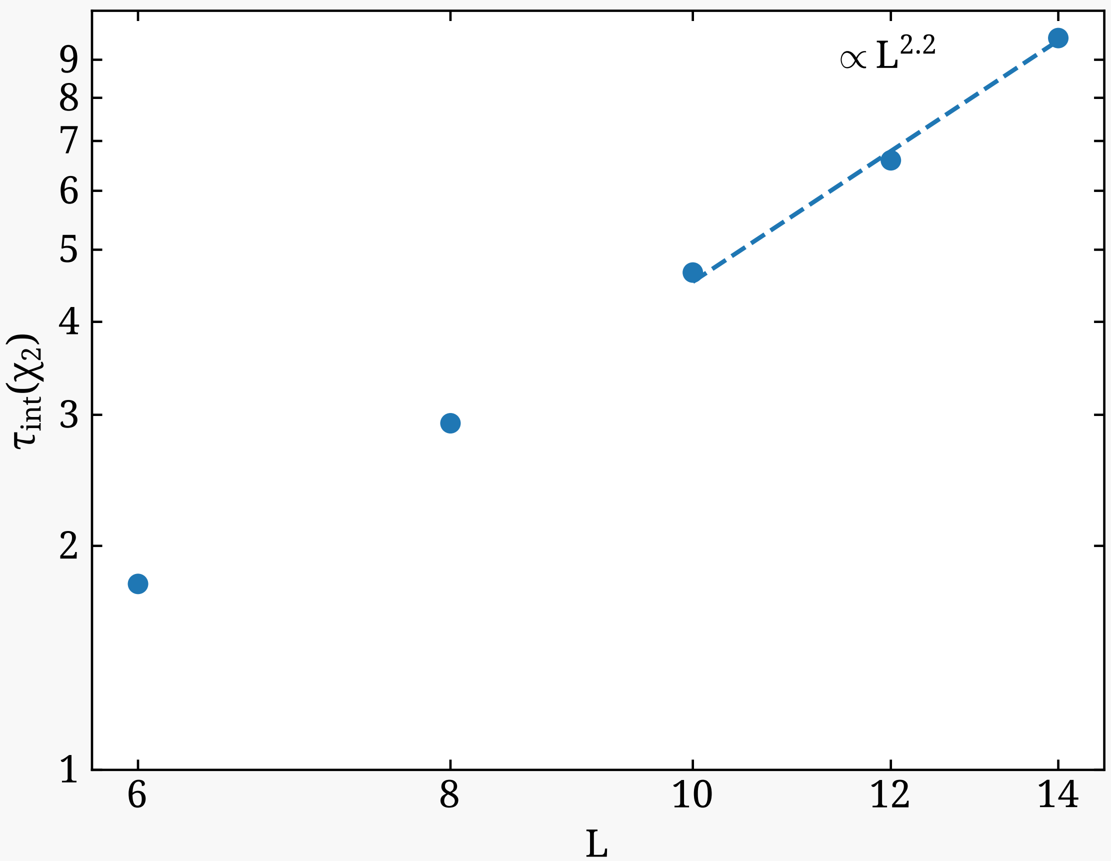
Where G_c is the two-point correlation function:

$$\chi_2 = \frac{1}{L^2} \sum_y \langle \phi(y)\phi(y+x) \rangle - \langle \phi(y) \rangle \langle \phi(y+x) \rangle$$

Example: ϕ^4 model

Example of critical slowing down



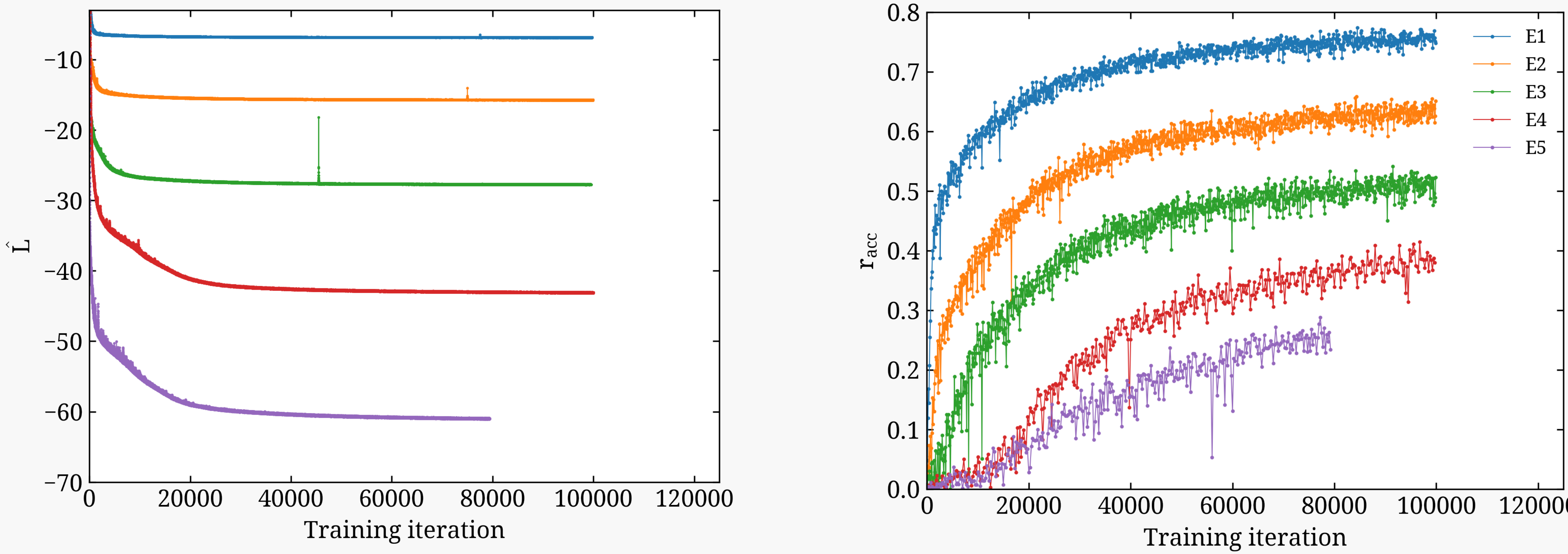


Observable autocorrelation time:

$$\rho_{\mathcal{O}}(\tau) = \frac{\frac{1}{M-\tau}\sum_{i=1}^{M-\tau}(\mathcal{O}_i - \langle\mathcal{O}\rangle)(\mathcal{O}_{i+\tau} - \langle\mathcal{O}\rangle)}{\frac{1}{M}\sum_{i=1}^{M}(\mathcal{O}_i - \langle\mathcal{O}\rangle)^2}$$

Flow-based generative model

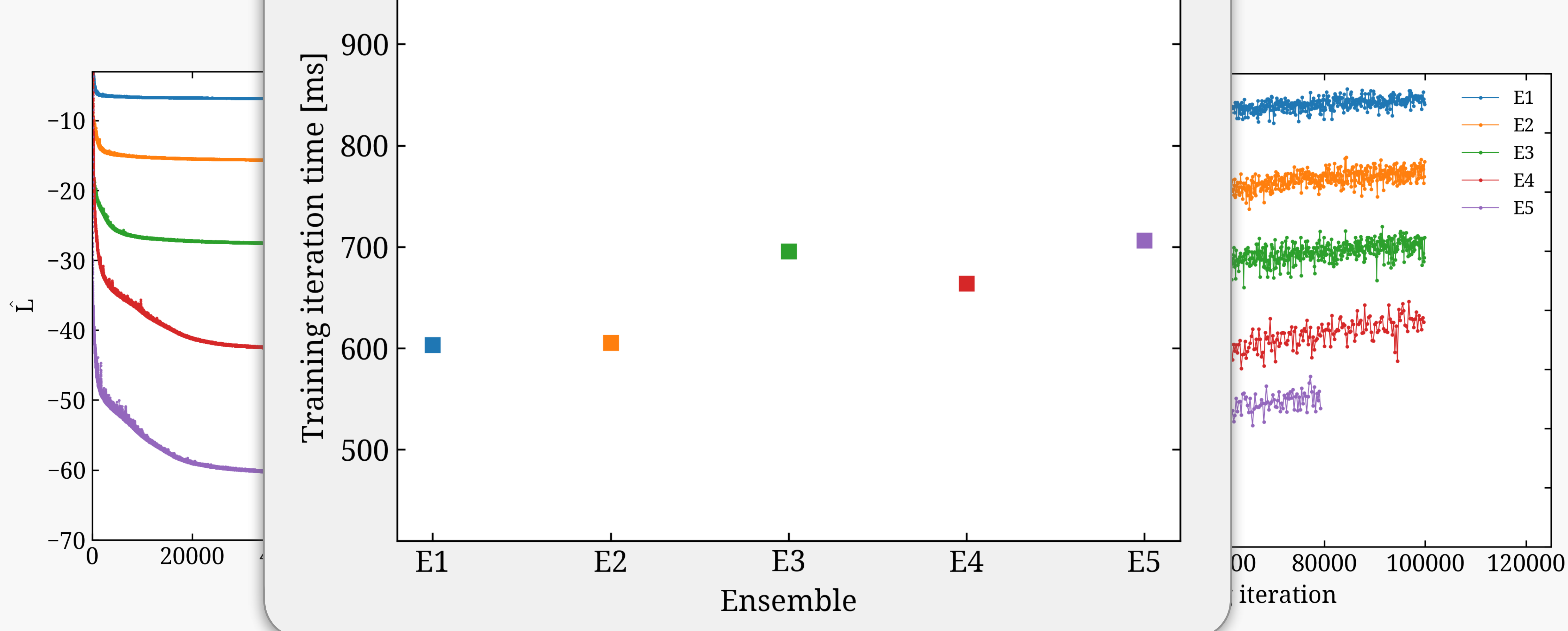
- Application to ϕ^4 model:



- Each ensemble on 1 Marconi100 node (4 V100 GPUs)
- Increasing complexity of neural networks and coupling layers from E1 to E5

Flow-based generative model

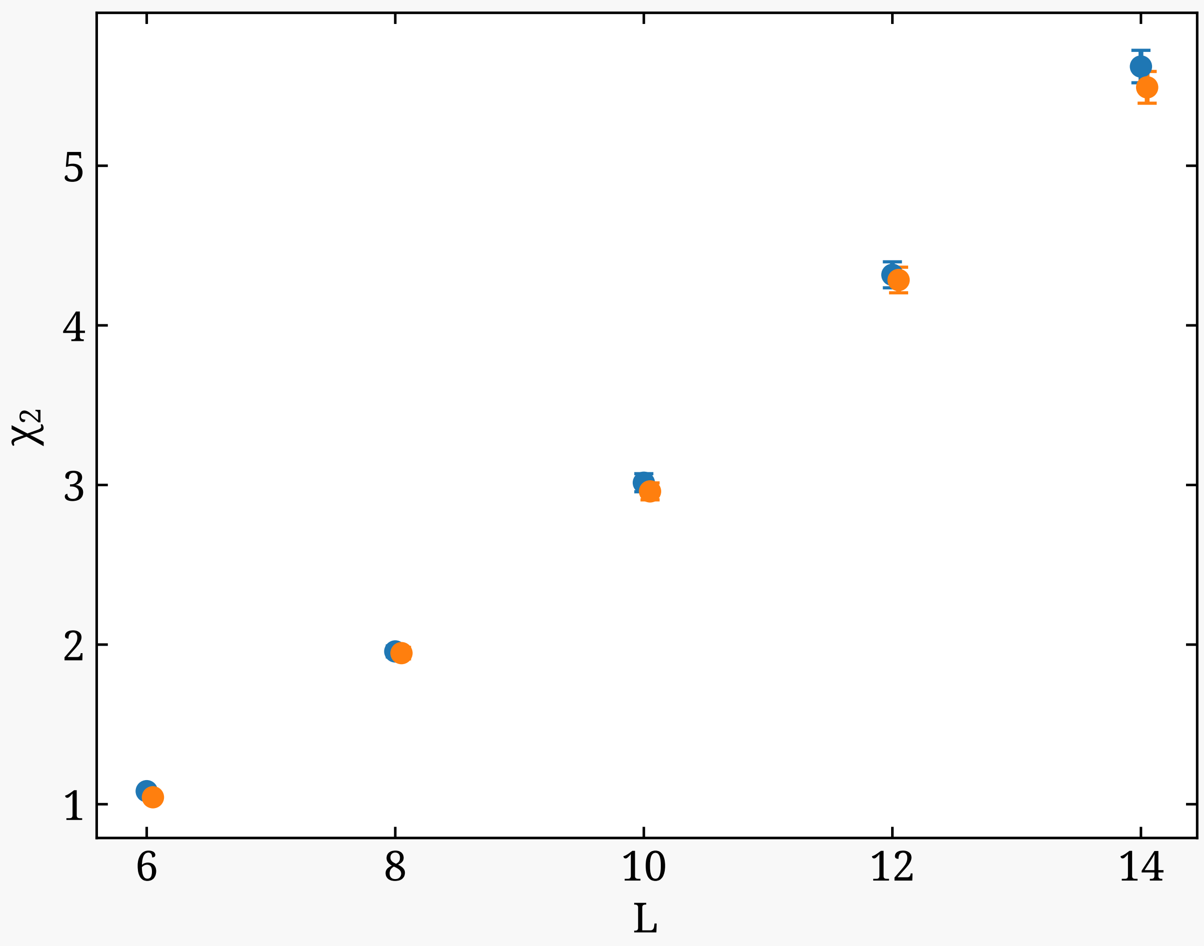
- Applicati

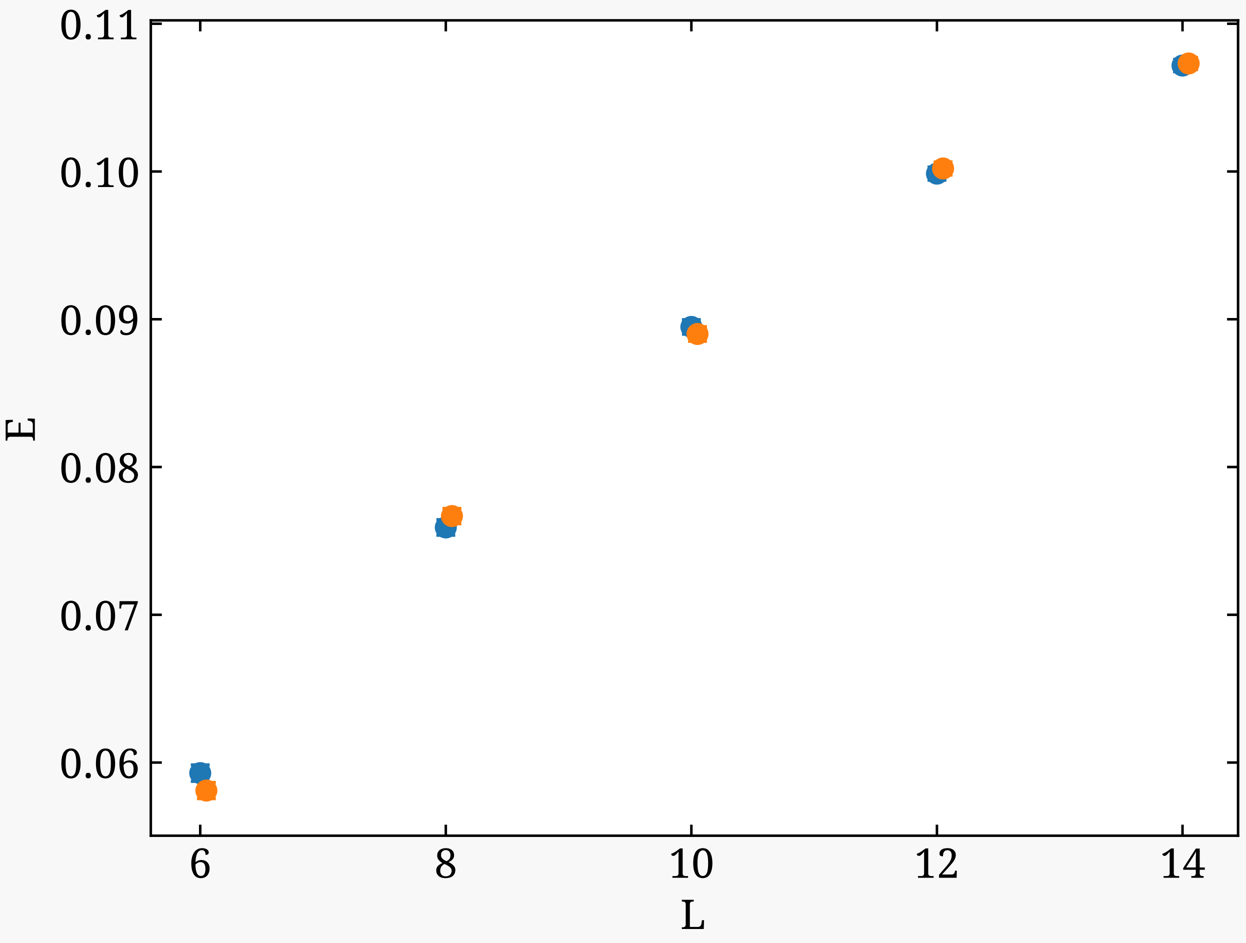


  - Each ensemble on 1 Marconi100 node (4 V100 GPUs)
  - Increasing complexity of neural networks and coupling layers from E1 to E5
  - Training cost: ~0.5-1 second per iteration

Flow-based generative model

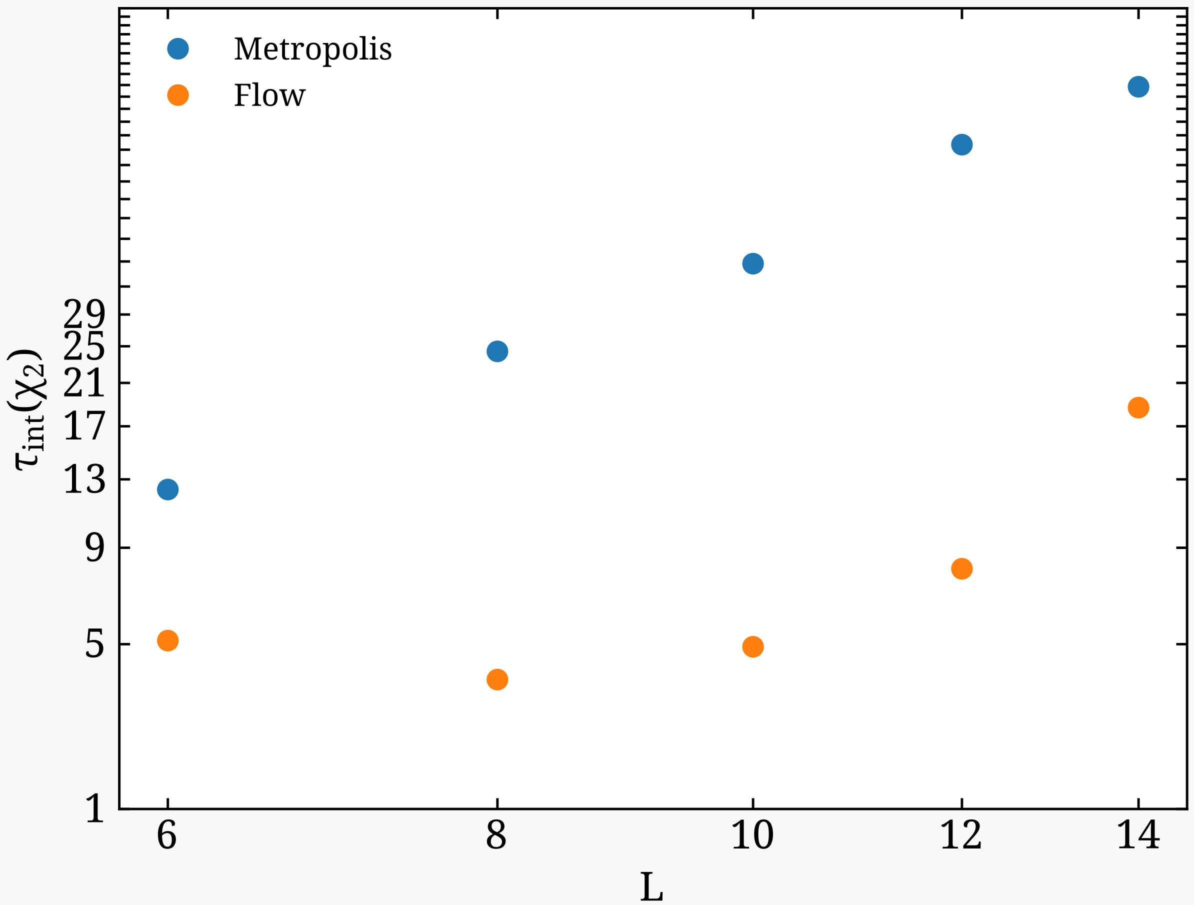
- Observable values towards criticality

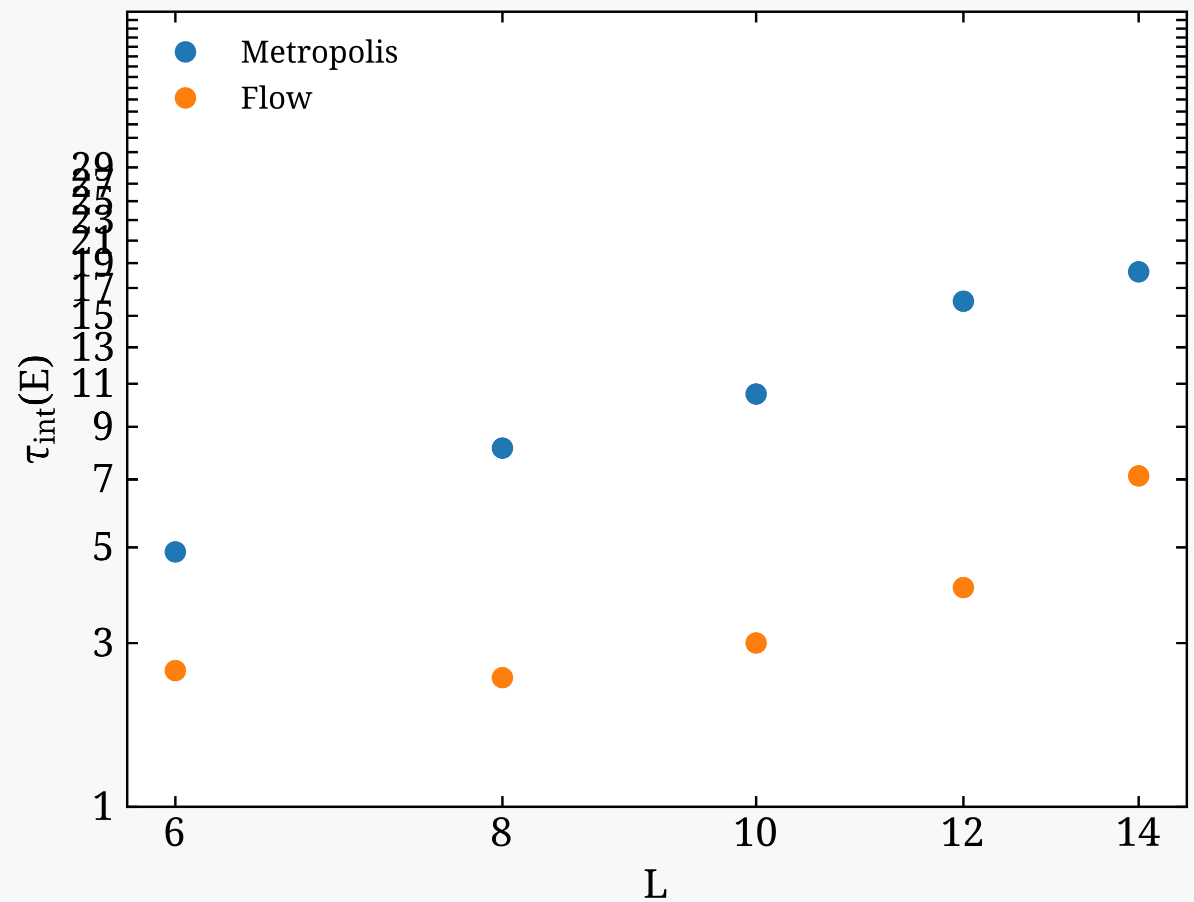




Flow-based generative model

- Autocorrelation times





Acknowledgements

Computer time used to produce some of the data shown

Marconi100, CINECA
PRACE Tier-0 project

SuperMUC-NG, LRZ
Gauss Large Scale project

Cyclone, CyI
Local project

Funding

RESEARCH
& INNOVATION
FOUNDATION

Project "NextQCD", Cyprus
Research and Innovation Foundation