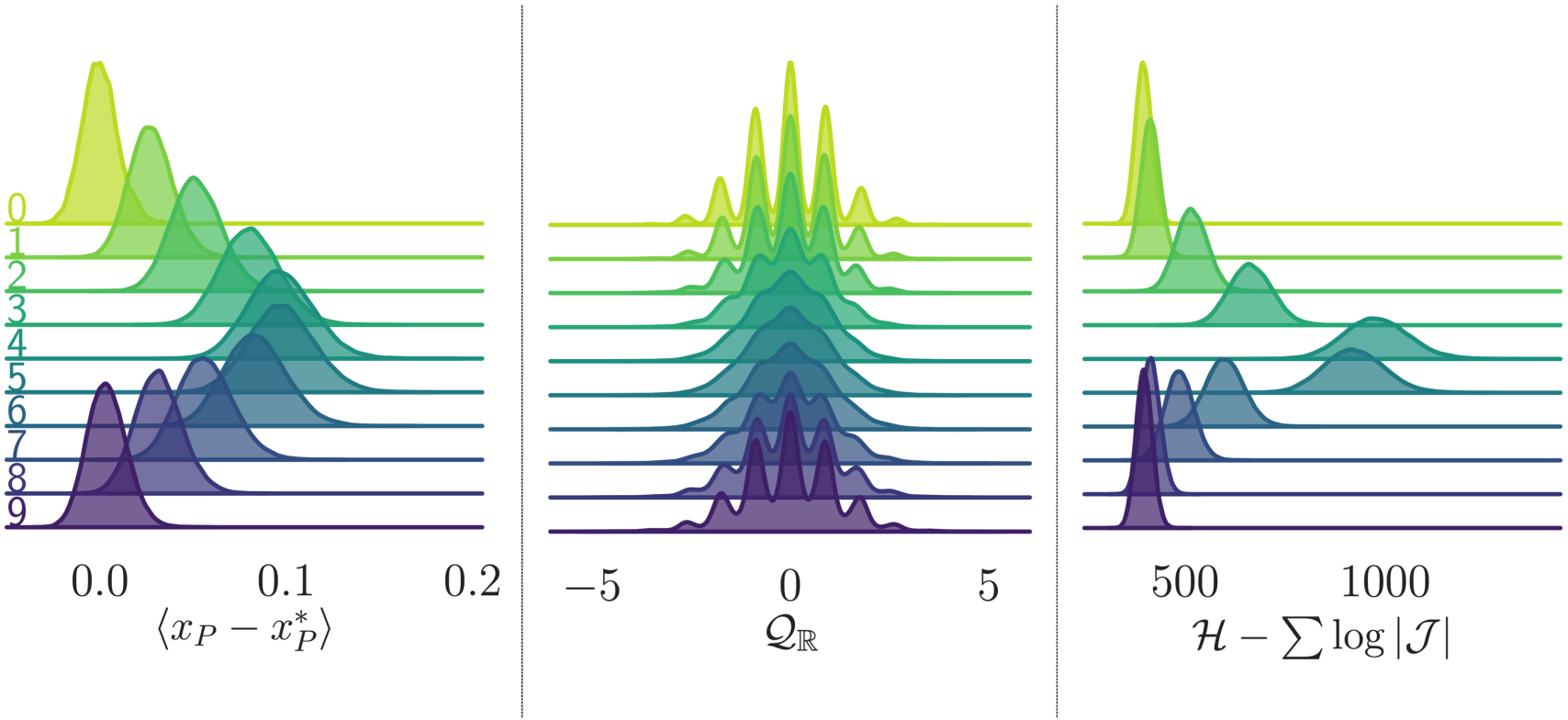
Deep Learning HMC

Building Topological Samplers for Lattice QCD



Argonne NATIONAL LABORATORY

Sam Foreman
May, 2021

saforem2/l2hmc-qcd
arXiv:2105.03418

Acknowledgements

Collaborators:

- Xiao-Yong Jin
- James C. Osborn

This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357.

Huge thank you to:

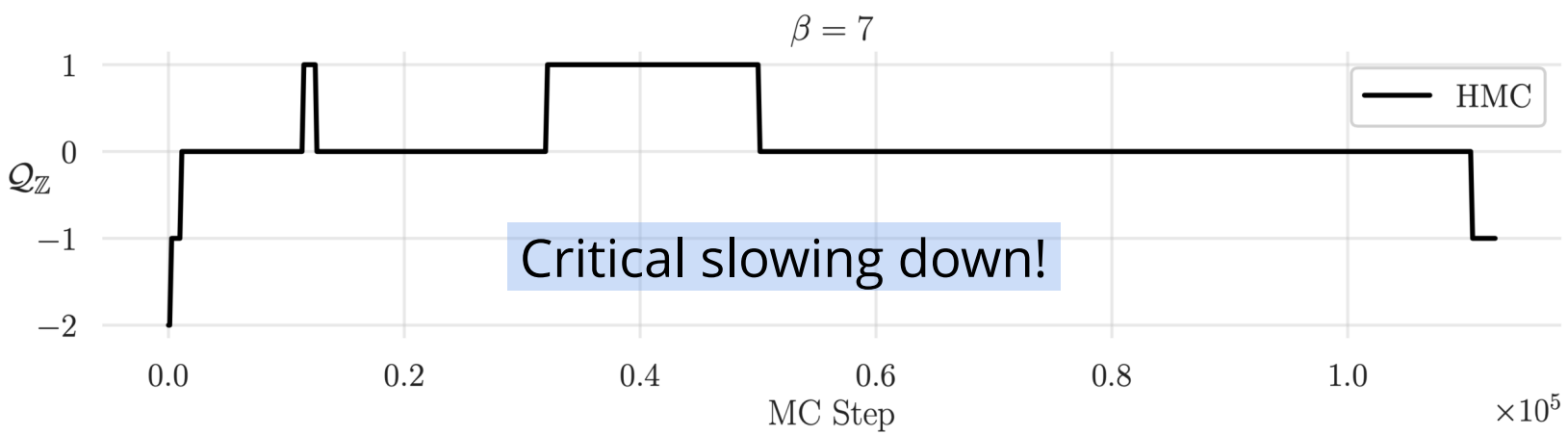
- Norman Christ
- Akio Tomiya
- Luchang Jin
- Chulwoo Jung
- Peter Boyle
- Taku Izubuchi
- Critical Slowing Down group (ECP)
- ALCF Staff + Datascience group

U.S. DEPARTMENT OF ENERGY

Argonne NATIONAL LABORATORY

MCMC in Lattice QCD

- Generating ***independent*** gauge configurations is a MAJOR bottleneck for LatticeQCD.
- As the lattice spacing, $a \to 0$, the MCMC updates tend to get stuck in sectors of fixed gauge topology.
  - This causes the number of steps needed to adequately sample different topological sectors to increase exponentially.



Markov Chain Monte Carlo (MCMC)

- **Goal:** Draw *independent* samples from a *target distribution,* $p(x)$
- Starting from some initial state $x_0 \sim \mathcal{N}(0, 1)$, we generate *proposal configurations* x'

$$x' = x_0 + \delta, \quad \delta \sim \mathcal{N}(0, 1)$$

- Use **Metropolis-Hastings** acceptance criteria

$$x_{i+1} = \begin{cases} x', & \text{with probability } A(x'|x) \\ x, & \text{with probability } 1 - A(x'|x) \end{cases}$$

$$A(x'|x) = \min\left\{1, \frac{p(x')}{p(x)} \left| \frac{\partial x'}{\partial x^T} \right|\right\}$$

Issues with MCMC

Goal: Generate an ensemble of *independent* configurations

- Generate proposal x':

  $x' = x \ + \delta$, where $\delta \sim \mathcal{N}(0, 1)$ random walk

1. **Construct chain:**

   $$x_0 \to x_1 \to x_2 \to \cdots \to x_{m-1} \to x_m \to x_{m+1} \to \cdots \to x_{n-2} \to x_{n-1} \to x_n$$

2. **Thermalize** ("burn-in"):

   $$x_0 \to x_1 \to x_2 \to \cdots \to x_{m-1} \to x_m \to x_{m+1} \to \cdots \to x_{n-2} \to x_{n-1} \to x_n$$

3. **Drop correlated samples** ("thinning"):

   $$x_0 \to x_1 \to x_2 \to \cdots \to x_{m-1} \to x_m \to x_{m+1} \to \cdots \to x_{n-2} \to x_{n-1} \to x_n$$

dropped saved

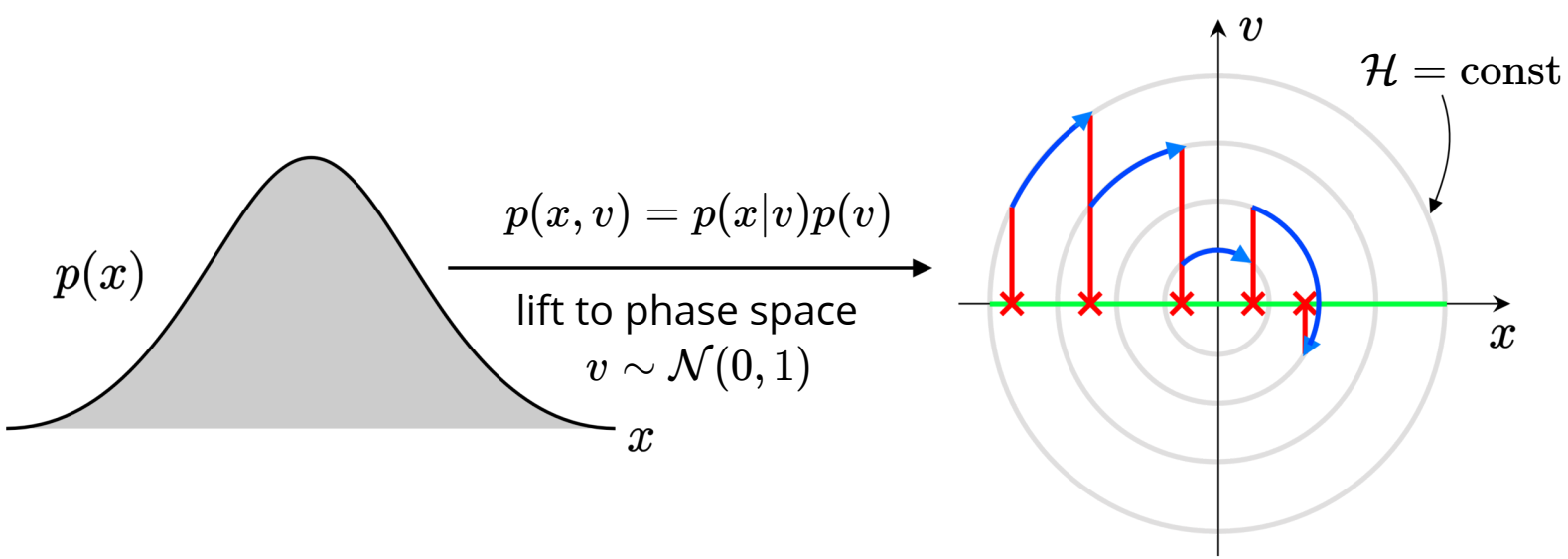
Inefficient!

Hamiltonian Monte Carlo (HMC)

- Target distribution: $p(x) \propto e^{-S(x)}$
- Introduce fictitious momentum: $v \sim \mathcal{N}(0, 1)$
- Joint target distribution:

$$p(x, v) = p(x) \cdot p(v) = e^{-S(x)} \cdot e^{-\frac{1}{2} v^T v} = e^{-\mathcal{H}(x,v)}$$

- Hamilton's Equations $\dot{x} = \frac{\partial \mathcal{H}}{\partial v}$, $\dot{v} = -\frac{\partial \mathcal{H}}{\partial x}$



HMC: Leapfrog Integrator

Leapfrog Integrator

1. Half-step v-update:

$$v^{1/2} = v - \tfrac{\varepsilon}{2}\partial_x S(x)$$

2. Full-step x-update:

$$x' = x + \varepsilon v^{1/2}$$

3. Half-step v-update:

$$v' = v^{1/2} - \tfrac{\varepsilon}{2}\partial_x S(x')$$

- **Hamiltonian:** $\Longrightarrow$

$$\mathcal{H}(x, v) = S(x) + \tfrac{1}{2} v^T v$$

- **Hamilton's Eqs:**

$$\dot{x} = \frac{\partial \mathcal{H}}{\partial v}, \dot{v} = -\frac{\partial \mathcal{H}}{\partial x}$$

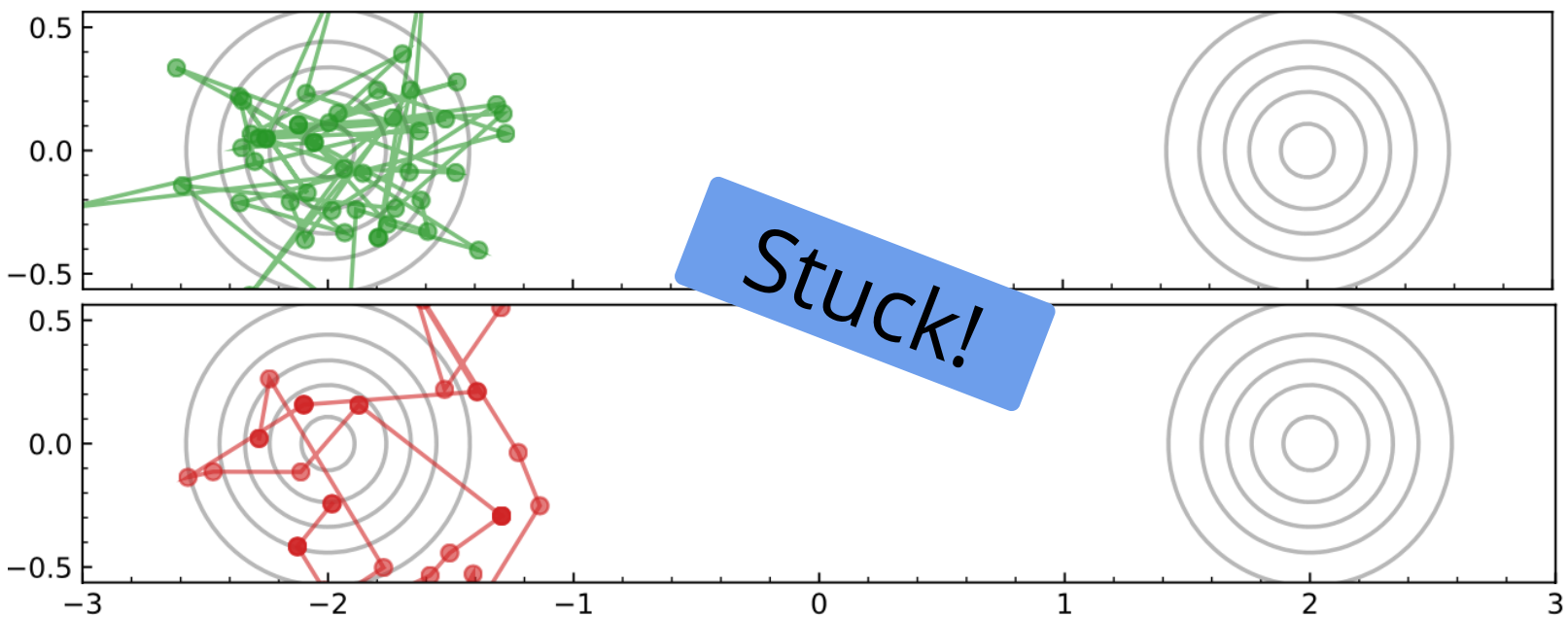
- N_{LF} **leapfrog steps:**

$$(x_0, v_0) \to \cdots \to (x_{N_{\mathrm{LF}}}, v_{N_{\mathrm{LF}}})$$

(trajectory)

HMC: Issues

- Energy levels selected randomly $\longrightarrow$ slow mixing!
- Cannot easily traverse low-density zones.
- What do we want in a ***good*** sampler?
  - Fast mixing
  - Fast burn-in
  - Mix across energy levels
  - Mix between modes



Leapfrog Layer

- Introduce a persistent *direction* $d \sim \mathcal{U}(+, -)$ *(forward/backward)*
- Let $\xi = (x, v, \pm)$ denote a complete state, then the *target distribution* is given by

$$p(\xi) = p(x) \cdot p(v) \cdot p(d)$$

- Introduce a discrete index $k \in \{1, 2, \ldots, N_{\mathrm{LF}}\}$ to denote the current *leapfrog step*
- Each leapfrog step transforms $\xi_k = (x_k, v_k, \pm) \to (x_k'', v_k'', \pm) = \xi_k''$ by passing it through the k^{th} *leapfrog layer*

Leapfrog Layer

- Each leapfrog step transforms $\xi_k = (x_k, v_k, \pm) \to (x_k'', v_k'', \pm) = \xi_k''$ by passing it through the k^{th} *leapfrog layer.*

- **v-update ($d = +$):**

$$v_k' = \Gamma_k^+(v_k; \zeta_{v_k}) \qquad \zeta_{v_k} \equiv \big(x_k, \partial_x S(x_k)\big) \qquad (v\text{-independent})$$

$$\equiv \underbrace{v_k \odot \exp\left(\tfrac{\varepsilon_v^k}{2} s_v^k(\zeta_{v_k})\right)}_{\text{Momentum } (v_k) \text{ scaling}} - \tfrac{\varepsilon_v^k}{2} \underbrace{\left[\partial_x S(x_k) \odot \exp\left(\varepsilon_v^k q_v^k(\zeta_{v_k})\right)\right]}_{\text{Gradient } \partial_x S(x_k) \text{ scaling}} + \underbrace{t_v^k(\zeta_{v_k})}_{\text{Translation}}$$

- **x-update ($d = +$):** masks: $\bar{m}_t + m_t = 1$

$$x_k' = \Lambda_k^+(x_k; \zeta_{x_k}) \qquad \zeta_{x_k} \equiv \big(\bar{m}_t \odot x_k, \partial_x S(x_k)\big) \quad (m_t \odot x)\text{-independent}$$

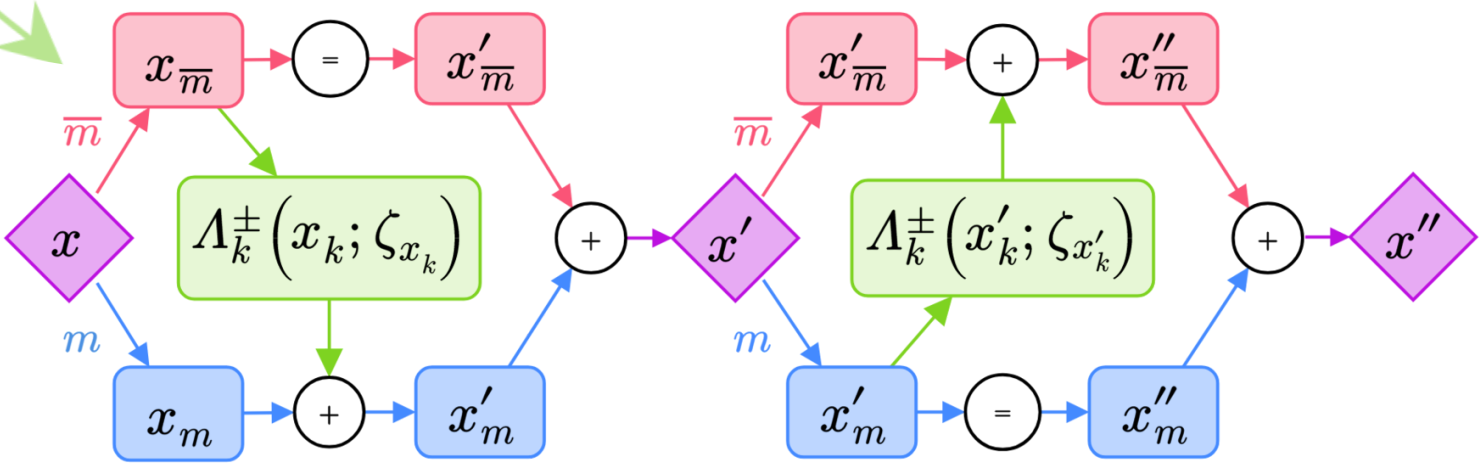
$$= x_k \odot \exp\left(\varepsilon_x^k s_x^k(\zeta_{x_k})\right) + \varepsilon_x^k \left[v_k' \odot \exp\left(\varepsilon_x^k q_x^k(\zeta_{x_k})\right) + t_x^k(\zeta_{x_k})\right]$$

where (s_v^k, q_v^k, t_v^k), and (s_x^k, q_x^k, t_x^k), are parameterized by neural networks

L2HMC: Generalized Leapfrog

- Complete (generalized) update:

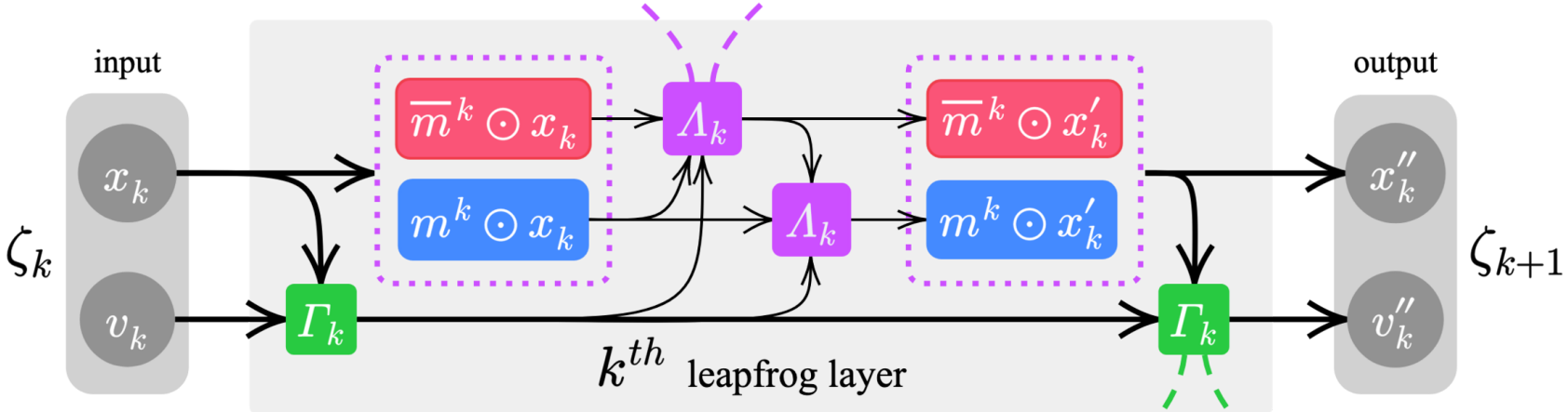
1. Half-step v update: $v'_k = \Gamma^{\pm}(v_k; \zeta_{v_k})$
2. Full-step $\frac{1}{2}x$ update: $x'_k = \bar{m}^t \odot x_k + m^t \odot \Lambda^{\pm}(x_k; \zeta_{x_k})$
3. Full-step $\frac{1}{2}x$ update: $x''_k = \bar{m}^t \odot \Lambda^{\pm}_k(x'_k; \zeta_{x'_k}) + m^t \odot x'_k$
4. Half-step v update: $v''_k = \Gamma^{\pm}(v'_k; \zeta_{v'_k})$



Leapfrog Layer

$$x_k \in U(1) \longrightarrow x_k = [\cos\theta, \sin\theta]$$

$$s_x^k = \alpha_{s,x} \tanh\left(w_{s,x}^T \cdot z_d + b_{s,x}\right)$$
$$t_x^k = w_{t,x}^T \cdot z_d + b_{t,x}$$
$$q_x^k = \alpha_{q,x} \tanh\left(w_{q,x}^T \cdot z_d + b_{q,x}\right)$$



$$s_v^k = \alpha_{s,v} \tanh\left(w_{s,v}^T \cdot z_d + b_{s,v}\right)$$
$$t_v^k = w_{t,v}^T \cdot z_d + b_{t,v}$$
$$q_v^k = \alpha_{q,v} \tanh\left(w_{q,v}^T \cdot z_d + b_{q,v}\right)$$

masks:

$$\overline{m}^k + m^k = 1$$

Stack of fully-connected layers

Training Algorithm

input :

1. Loss function, $\mathcal{L}_\theta(\xi', \xi, A(\xi'|\xi))$
2. Batch of initial states, x
3. Learning rate schedule, $\{\alpha_t\}_{t=0}^{N_{\text{train}}}$
4. Annealing schedule, $\{\gamma_t\}_{t=0}^{N_{\text{train}}}$
5. Target distribution, $p_t(x) \propto e^{-\gamma_t S_\beta(x)}$

Initialize weights θ

for $0 \leq t < N_{\text{train}}$ **:**

- resample $v \sim \mathcal{N}(0, \mathbb{1})$
- resample $d \sim \mathcal{U}(+, -)$
- construct $\xi_0 \equiv (x_0, v_0, d_0)$
- **for** $0 \leq k < N_{\text{LF}}$ **:**
  - propose (leapfrog layer) $\xi'_k \leftarrow \xi_k$
- compute $A(\xi'|\xi) = \min\left\{1, \frac{p(\xi')}{p(\xi)} \left|\frac{\partial \xi'}{\partial \xi^T}\right|\right\}$
- update $\mathcal{L} \leftarrow \mathcal{L}_\theta(\xi', \xi, A(\xi'|\xi))$
- backprop $\theta \leftarrow \theta - \alpha_t \nabla_\theta \mathcal{L}$
- assign $x_{t+1} \leftarrow \begin{cases} x' & \text{with probability } A(\xi'|\xi) \\ x & \text{with probability } (1 - A(\xi'|\xi)). \end{cases}$

re-sample momentum + direction

construct trajectory

Compute loss + backprop

Metropolis-Hastings accept/reject

Example: GMM $\in \mathbb{R}^2$

- Define the *squared jump distance*:
  $\delta(\xi', \xi) = \|x' - x\|_2^2$
- Maximize
  expected squared jump distance:
  $\mathcal{L}_\theta(\theta) \equiv \mathbb{E}_{p(\xi)}\left[A(\xi'|\xi) \cdot \delta(\xi', \xi)\right]$

Note:

$A(\xi', \xi)$ = acceptance probability

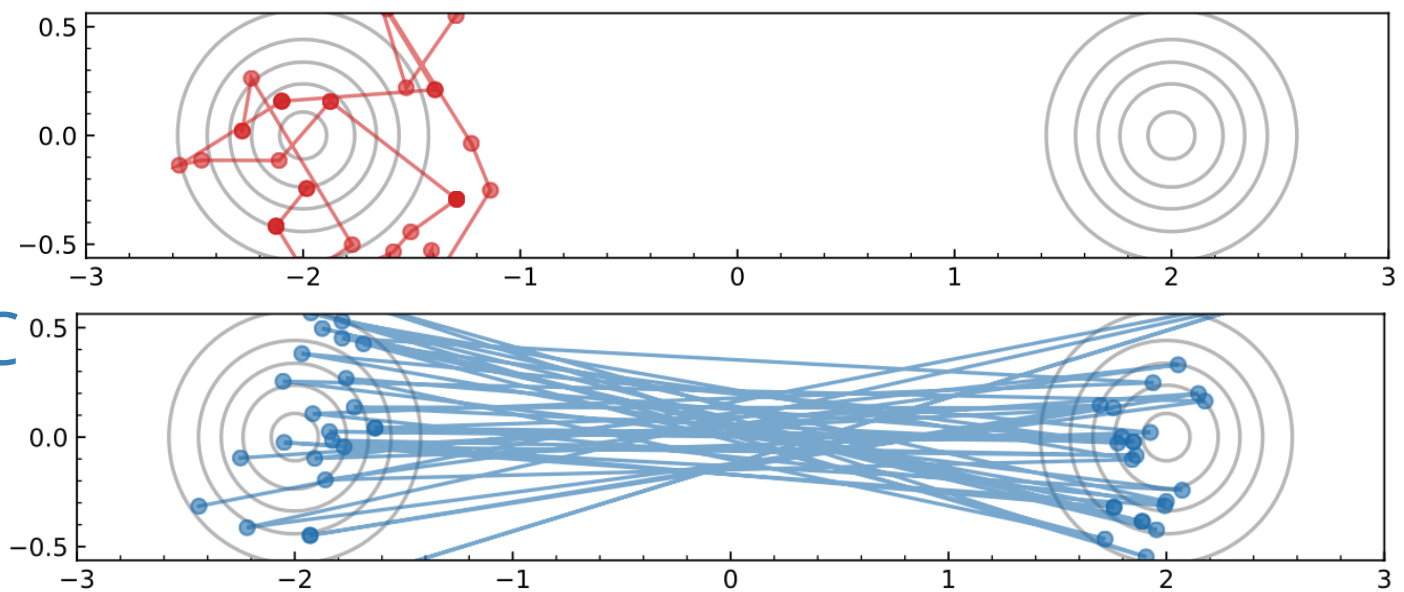
$A(\xi'|\xi) \cdot \delta(\xi', \xi)$= avg. distance

ξ = initial state

ξ = initial state

HMC

L2HMC



Annealing Schedule

- Introduce an annealing schedule during the **training** phase:

$$\{\gamma_t\}_{t=0}^{N} = \{\gamma_0, \gamma_1, \ldots, \gamma_{N-1}, \gamma_N\}, \quad \text{e.g. } \{0.1, 0.2, \ldots, 0.9, 1.0\}$$

$$\gamma_0 < \gamma_1 < \ldots < \gamma_N \equiv 1 \quad \text{(increasing)}$$

$$\gamma_{t+1} - \gamma_t \ll 1 \quad \text{(varied *slowly*)}$$

- For $\|\gamma_t\| < 1$, this helps to rescale (*shrink)* the energy barriers between isolated modes
  - Allows our sampler to explore previously inaccessible regions of the target distribution

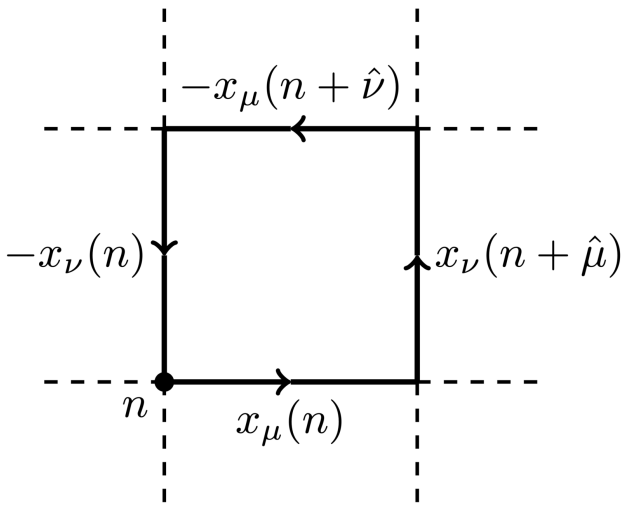
- Target distribution becomes:

$$p_t(x) \propto e^{-\gamma_t S(x)}, \quad \text{for} \quad t = 0, 1, \ldots, N$$

Lattice Gauge Theory

- Link variables:

$$U_\mu(x) = e^{ix_\mu(n)} \in U(1)$$
$$x_\mu(n) \in [-\pi, \pi]$$



- Wilson action:

$$S_\beta(x) = \beta \sum_P 1 - \cos x_P$$
$$x_P = x_\mu(n) + x_\nu(n+\hat{\mu}) - x_\mu(n+\hat{\nu}) - x_\nu(n)$$

- Topological charge:

$$\mathcal{Q}_\mathbb{R} = \frac{1}{2\pi} \sum_P \sin x_P \in \mathbb{R}$$

✅ continuous, differentiable

$$\mathcal{Q}_\mathbb{Z} = \frac{1}{2\pi} \sum_P \lfloor x_P \rfloor \in \mathbb{Z}$$

❌ discrete, hard to work with

$$\lfloor x_P \rfloor = x_P - 2\pi \left\lfloor \frac{x_P + \pi}{2\pi} \right\rfloor$$

Non-Compact Projection [1.]

- Project $[-\pi, \pi]$ onto $\mathbb{R}$ using a transformation: $z = g(x)$, $g : [-\pi, \pi] \to \mathbb{R}$
  - $z = \tan\left(\frac{x}{2}\right)$
- Perform the update in $\mathbb{R}$
  - $z' = m^t \odot z + \bar{m}^t \odot [\alpha z + \beta]$
- Project back to $[-\pi, \pi]$ using the inverse transformation $x = g^{-1}(z)$, $g^{-1} : \mathbb{R} \to [-\pi, \pi]$
  - $x = 2\tan^{-1}(z)$
- These steps can be combined into a single update equation
  - $x' = m^t \odot x + \bar{m}^t \odot \left[2\tan^{-1}\left(\alpha \tan\left(\frac{x}{2}\right)\right) + \beta\right]$
  - with corresponding Jacobian factor
    - $\frac{\partial x'}{\partial x} = \frac{\exp(\varepsilon s_x)}{\cos^2(x/2) + \exp(2\varepsilon s_x)\sin(x/2)}$

$$x_k \in U(1) \longrightarrow x_k = [\cos\theta, \sin\theta]$$

[1.] "Normalizing Flows on Tori and Spheres" arXiv:2002.02428

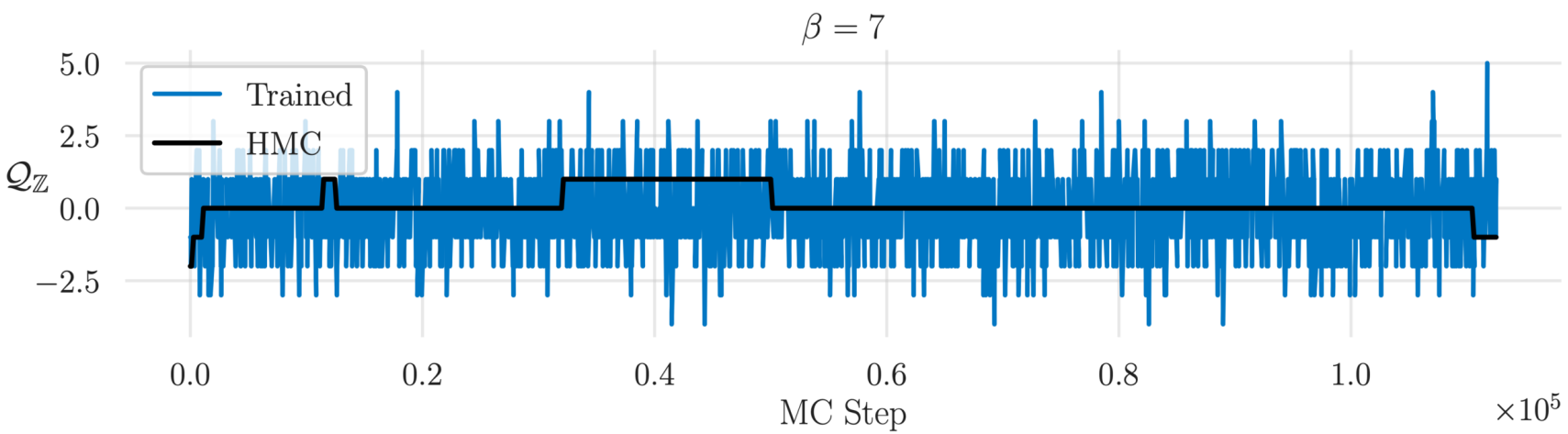
Loss function: $\mathcal{L}(\theta)$

- We maximize the *expected squared charge difference:*

$$\mathcal{L}(\theta) = \mathbb{E}_{p(\xi)}\left[-\delta\mathcal{Q}^2_{\mathbb{R}}(\xi', \xi) \cdot A(\xi'|\xi)\right]$$

$$\delta\mathcal{Q}^2_{\mathbb{R}}(\xi', \xi) \equiv \left(\mathcal{Q}_{\mathbb{R}}(x') - \mathcal{Q}_{\mathbb{R}}(x)\right)^2$$

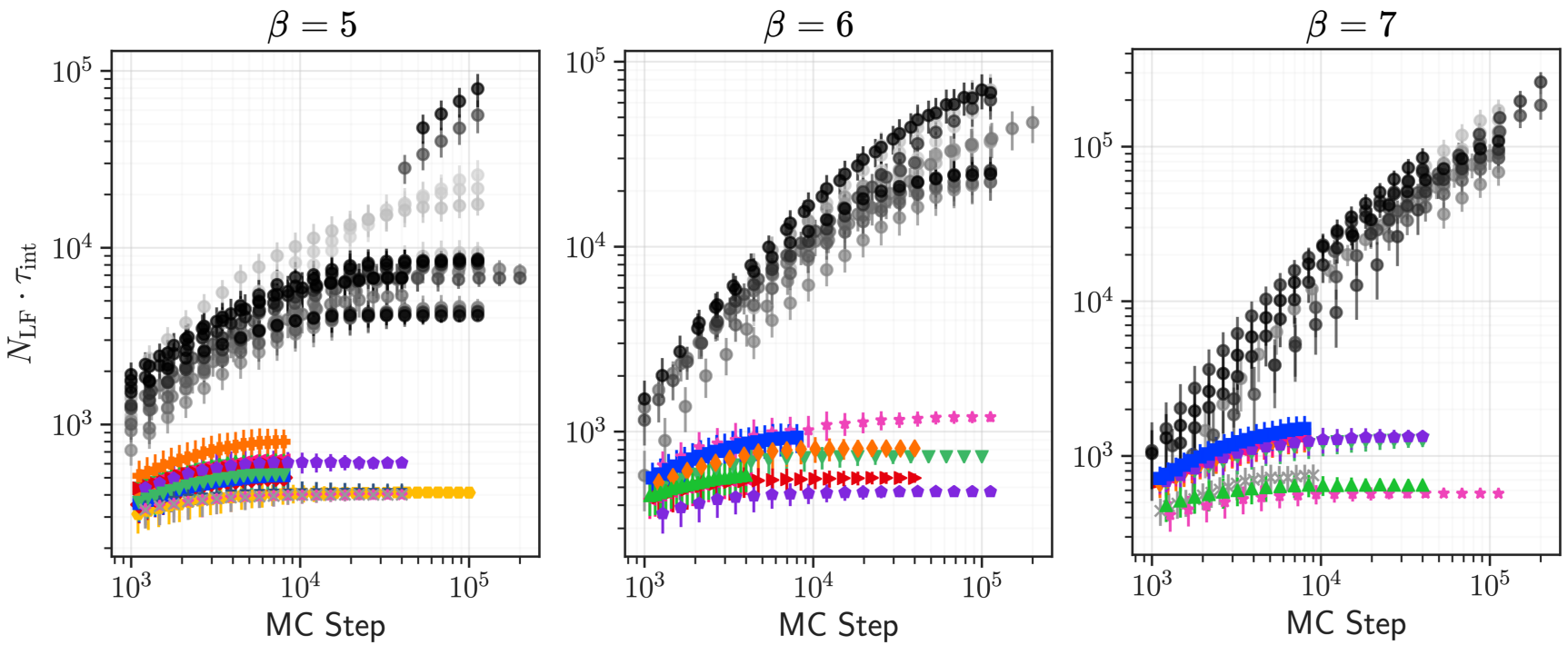
$$A(\xi'|\xi) = \min\left\{1, \frac{p(\xi')}{p(\xi)}\left|\frac{\partial\xi'}{\partial\xi^T}\right|\right\}$$



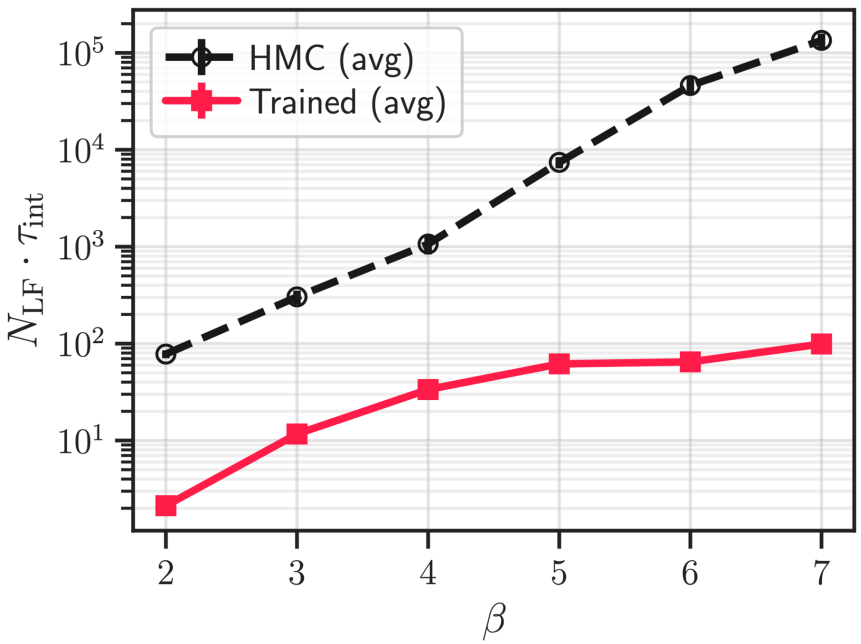
Results: $\tau_{\text{int}}^{\mathcal{Q}_\mathbb{Z}}$

- Want to calculate: $\langle \mathcal{O} \rangle \propto \int [\mathcal{D}x]\, \mathcal{O}(x) e^{-S[x]}$
- If we had *independent* configurations, we could approximate by $\langle \mathcal{O} \rangle \simeq \frac{1}{N} \sum_{n=1}^{N} \mathcal{O}(x_n) \longrightarrow \sigma^2 = \frac{1}{N} \mathrm{Var}\left[\mathcal{O}(x)\right] \propto \frac{1}{N}$
- Instead, we account for the *autocorrelation*, so the variance becomes: $\sigma^2 = \frac{\tau_{int}^{\mathcal{O}}}{N} \mathrm{Var}\left[\mathcal{O}(x)\right]$

Rescale: $N_{\text{LF}} \cdot \tau_{\text{int}}^{\mathcal{Q}_\mathbb{Z}}$ to account for different *trajectory lengths*



Results: $\tau_{\text{int}}^{\mathcal{Q}_{\mathbb{Z}}}$

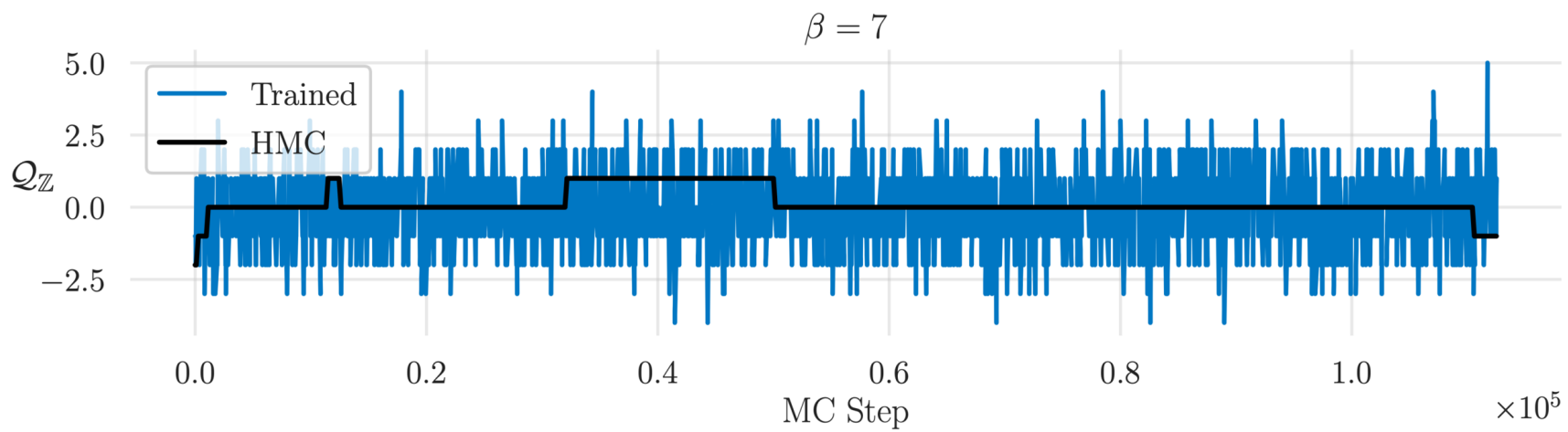


- We maximize the *expected squared charge difference:*

$$\mathcal{L}(\theta) = \mathbb{E}_{p(\xi)}\left[-\delta\mathcal{Q}^2_{\mathbb{R}}(\xi', \xi) \cdot A(\xi'|\xi)\right]$$

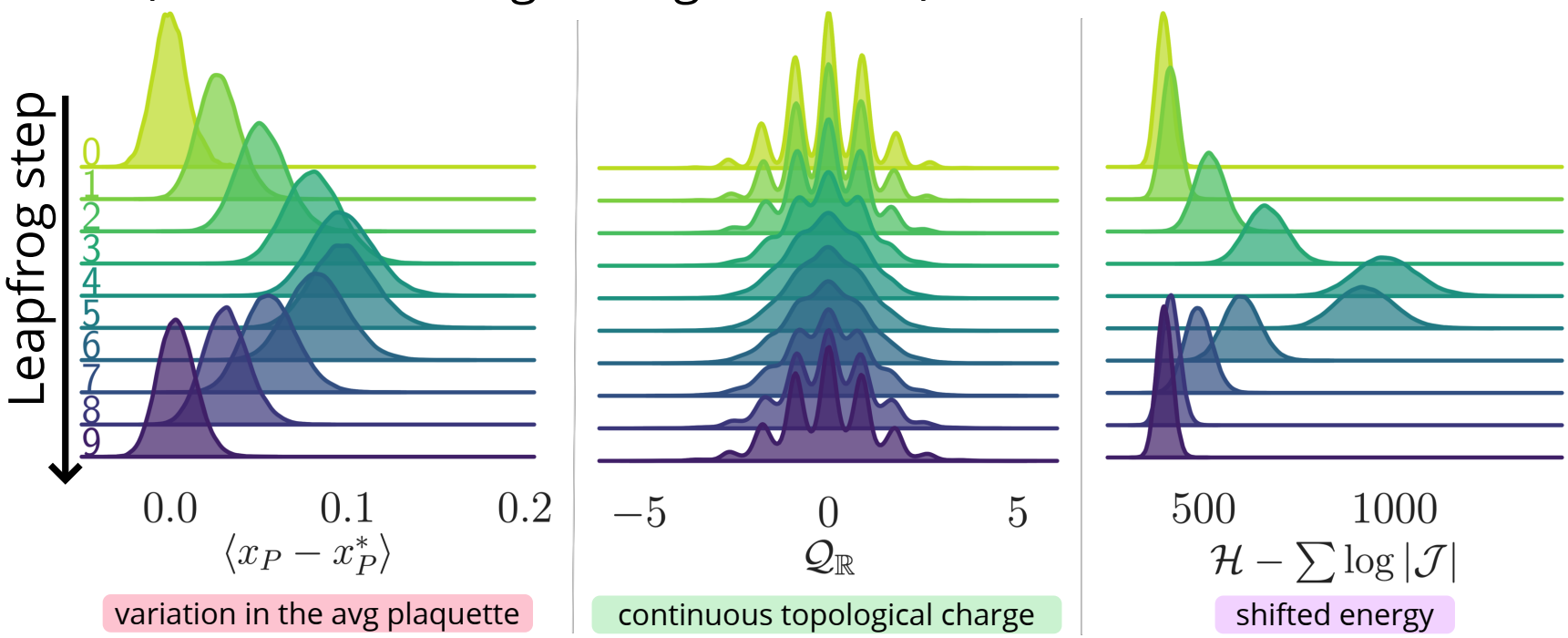
$$\delta\mathcal{Q}^2_{\mathbb{R}}(\xi', \xi) \equiv \left(\mathcal{Q}_{\mathbb{R}}(x') - \mathcal{Q}_{\mathbb{R}}(x)\right)^2$$

$$A(\xi'|\xi) = \min\left\{1, \frac{p(\xi')}{p(\xi)}\left|\frac{\partial\xi'}{\partial\xi^T}\right|\right\}$$



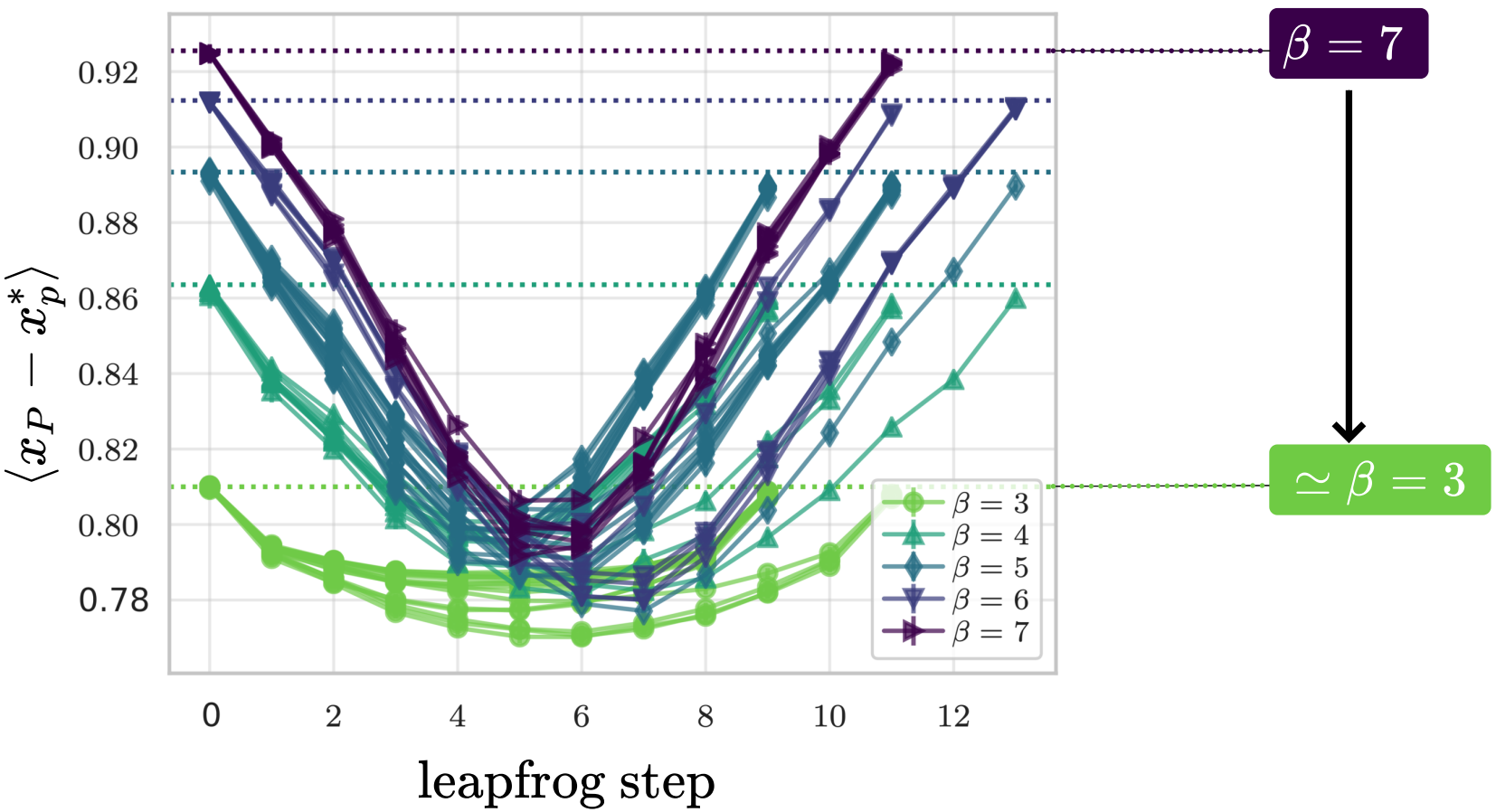
Interpretation

- Look at how different quantities evolve over a single trajectory
  - See that the sampler artificially *increases the energy* during the first half of the trajectory (before returning to original value)



Interpretation

- Look at how the variation in $\langle \delta x_P \rangle$ varies for different values of β



Training Costs

- We trained our model(s) using Horovod with TensorFlow on the ThetaGPU supercomputer at the Argonne Leadership Computing Facility.
- A typical training run:
  - 1 node (8$\times$ NVIDIA A100 GPUs)
  - Batch size $M = 2048$
  - Hidden layer shapes $= \{256, 256, 256\}$
  - Leapfrog layers $N_{\mathrm{LF}} = 10$
  - Lattice volume $= 16 \times 16$
  - Training steps $= 5 \times 10^5$
  - $\simeq$ 24 hours to complete.

Next Steps

- Going forward, we plan to:
  - Continue testing on larger lattice volumes to better understand scaling efficiency
  - Generalize to 2D / 4D $SU(3)$
  - Test alternative network architectures
    - Gauge Equivariant layers

Thanks for listening!

Interested?

arXiv:2105.03418


saforem2/l2hmc-qcd

slides.com/samforeman/dlhmc