PROCEEDINGS OF SCIENCE

LeapfrogLayers: A Trainable Framework for Effective Topological Sampling

Sam Foreman,[a,*] **Xiao-Yong Jin**[a,b] **and James C. Osborn**[a,b]

[a] *Leadership Computing Facility, Argonne National Laboratory, Lemont, IL, USA*

[b] *Computational Science Division, Argonne National Laboratory, Lemont, IL, USA*

E-mail: foremans@anl.gov, xjin@anl.gov, osborn@alcf.anl.gov

We introduce LeapfrogLayers, an invertible neural network architecture that can be trained to efficiently sample the topology of a 2D $U(1)$ lattice gauge theory. We show an improvement in the integrated autocorrelation time of the topological charge when compared with traditional HMC, and propose methods for scaling our model to larger lattice volumes. Our implementation is open source, and is publicly available on github at github.com/saforem2/l2hmc-qcd.

The 38th International Symposium on Lattice Field Theory
26-30 July 2021
Zoom / Gather @ MIT, Cambridge MA, USA

*Speaker

1. Introduction

The main task in lattice field theory calculations is to evaluate integrals of the form

$$\langle \mathcal{O} \rangle \propto \int [\mathcal{D}\mathbf{x}]\, \mathcal{O}(\mathbf{x}) p(\mathbf{x}), \tag{1}$$

for some multivariate target distribution $p(\mathbf{x}) \propto e^{-S(\mathbf{x})}/Z$. We can approximate the integral using Markov Chain Monte Carlo (MCMC) sampling techniques. This is done by sequentially generating a chain of configurations $\{\mathbf{x}_1, \mathbf{x}_2, \dots \mathbf{x}_N\}$, with $\mathbf{x}_i \sim p(\mathbf{x})$ and averaging the value of the function $\mathcal{O}(\mathbf{x})$ over the chain

$$\overline{\mathcal{O}} = \frac{1}{N} \sum_{n=1}^{N} \mathcal{O}(\mathbf{x}_n). \tag{2}$$

Accounting for correlations between states in the chain, the sampling variance is given by

$$\sigma^2 = \frac{\tau_{\mathcal{O}}^{\text{int}}}{N} \sum_{n=1}^{N} \left[\mathcal{O}(\mathbf{x}_n) - \overline{\mathcal{O}} \right]^2 \tag{3}$$

where $\tau_{\mathcal{O}}^{\text{int}}$ is the integrated autocorrelation time. This quantity can be interpreted as the additional time required for these induced correlations to become negligible.

While our main target for improved gauge field generation is full QCD, here we restrict our work to the simpler 2D U(1) pure-gauge lattice theory. There are already many efficient alternatives for simulating this theory (see e.g. [1] and references therein for a comparison), however, they either are not generally applicable to full QCD or haven't demonstrated a significant advantage over HMC there. For this reason we only consider HMC as a baseline method for performance comparison. The method presented here can be extended to full QCD in a relatively straightforward manner, though the cost of training and efficiency are still open questions.

1.1 Charge Freezing

The ability to efficiently generate independent configurations is currently a major bottleneck for lattice simulations. The 2D U(1) gauge theory considered here has sectors of topological charge that become separated by large potential barriers for increasing β, similar to full QCD.

The theory is defined in terms of the link variables $U_\mu(x) = e^{ix} \in U(1)$ with $x \in [-\pi, \pi)$. Our target distribution is given by $p(\mathbf{x}) \propto e^{-S_\beta(\mathbf{x})}$, where $S_\beta(\mathbf{x})$ is the Wilson action

$$S_\beta(\mathbf{x}) = \beta \sum_P 1 - \cos x_P, \tag{4}$$

defined in terms of the sum of the gauge variables around the elementary plaquette, $x_P \equiv x_\mu(n) + x_\nu(n+\hat{\mu}) - x_\mu(n+\hat{\nu}) - x_\nu(n)$, starting at site n of the lattice, and $\sum_P$ denotes the sum over all such plaquettes. For a given lattice configuration, we can calculate the topological charge $Q \in \mathbb{Z}$ using

$$Q_{\mathbb{Z}} = \frac{1}{2\pi} \sum_P \lfloor x_P \rfloor\,, \text{ where } \lfloor x_P \rfloor \equiv x_P - 2\pi \left\lfloor \frac{x_P + \pi}{2\pi} \right\rfloor. \tag{5}$$

As $\beta \to \infty$, we see that the value of $Q_{\mathbb{Z}}$ remains fixed for large durations of the simulation. This freezing can be quantatively measured by introducing the *tunneling rate* (as measured between subsequent states in our chain i and $i+1$)

$$\overline{\delta Q}_{\mathbb{Z}} \text{ , with } \delta Q_{\mathbb{Z},i} \equiv \left| Q_{\mathbb{Z},i+1} - Q_{\mathbb{Z},i} \right| \in \mathbb{Z}, \tag{6}$$

which serves as a measure for how efficiently our chain is able to jump (tunnel) between sectors of distinct topological charge. From Figure 1, we can see that $\overline{\delta Q}_{\mathbb{Z}} \to 0$ as $\beta \to \infty$.

2. Hamiltonian Monte Carlo (HMC)

The Hamiltonian Monte Carlo algorithm begins by introducing a fictitious momentum, v, for each coordinate varible, x, typically taken from an independent normal distribution. This allows us to write the joint target density of the $\xi \equiv (\mathbf{x}, \mathbf{v})$ system as

$$p(\mathbf{x}, \mathbf{v}) = p(\mathbf{x})p(\mathbf{v}) = e^{-S_\beta(\mathbf{x})} e^{-\mathbf{v}^2/2} = e^{-\mathcal{H}(\mathbf{x},\mathbf{v})} \tag{7}$$

where $\mathcal{H}(\xi) = \mathcal{H}(\mathbf{x}, \mathbf{v}) = S_\beta(\mathbf{x}) + \frac{1}{2}\mathbf{v}^2$ is the Hamiltonian of the system. We use the *leapfrog integrator* to approximately numerically integrate Hamilton's equations

$$\dot{\mathbf{x}} = \frac{\partial \mathcal{H}}{\partial \mathbf{v}} \quad , \quad \dot{\mathbf{v}} = -\frac{\partial \mathcal{H}}{\partial \mathbf{x}} \tag{8}$$

along iso-probability contours of $\mathcal{H} = \text{const.}$ from $\xi = (\mathbf{x}, \mathbf{v}) \to (\mathbf{x}^*, \mathbf{v}^*) = \xi^*$. The error in this integration is then corrected by a Metropolis-Hastings (MH) accept/reject step.

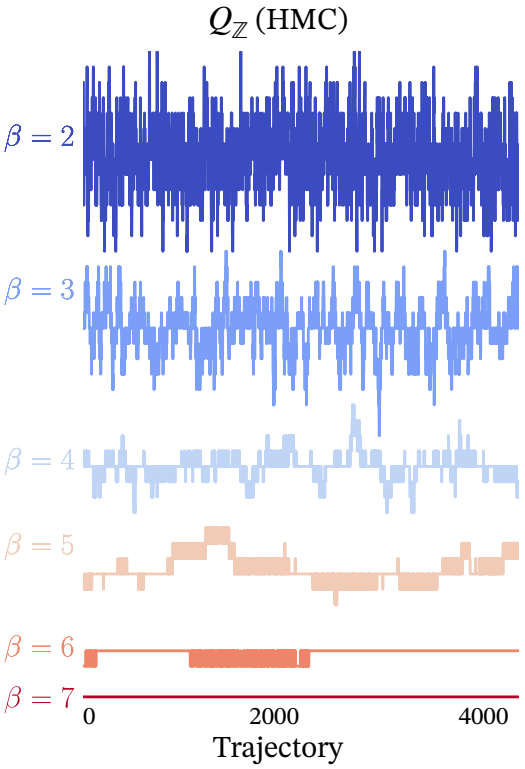


Figure 1: Illustration of the topological charge, $Q_{\mathbb{Z}}$, freezing as $\beta : 2 \to 7$ for traditional HMC.

2.1 Leapfrog Integrator

1. Starting from $\mathbf{x}_i$, resample the momentum $\mathbf{v}_i \sim \mathcal{N}(0, \mathbb{1})$ and construct the state $\xi = (\mathbf{x}_i, \mathbf{v}_i)$.
2. Generate a *proposal configuration* ξ^* by integrating $\dot{\xi}$ along $\mathcal{H} = \text{const.}$ for N leapfrog steps. i.e.

$$\xi \to \xi_1 \to \ldots \to \xi_N \equiv \xi^*, \tag{9}$$

where a single leapfrog step $\xi_i \to \xi_{i+1}$ above consists of:

$$\textbf{(a.)}\ \mathbf{v}' \leftarrow \mathbf{v} - \frac{\varepsilon}{2}\partial_{\mathbf{x}} S(\mathbf{x}), \qquad \textbf{(b.)}\ \mathbf{x}' \leftarrow \mathbf{x} + \varepsilon \mathbf{v}', \qquad \textbf{(c.)}\ \mathbf{v}'' \leftarrow \mathbf{v}' - \frac{\varepsilon}{2}\partial_{\mathbf{x}} S(\mathbf{x}). \tag{10}$$

3. At the end of the trajectory, accept or reject the proposal configuration ξ^* using the MH test

$$\mathbf{x}_{i+1} \leftarrow \begin{cases} \mathbf{x}^* \text{ with probability } A(\xi^*|\xi) \equiv \min\left\{1, \frac{p(\xi^*)}{p(\xi)}\right\} \\ \mathbf{x}_i \text{ with probability } 1 - A(\xi^*|\xi). \end{cases} \tag{11}$$

An illustration of this procedure can be seen in Figure 2.

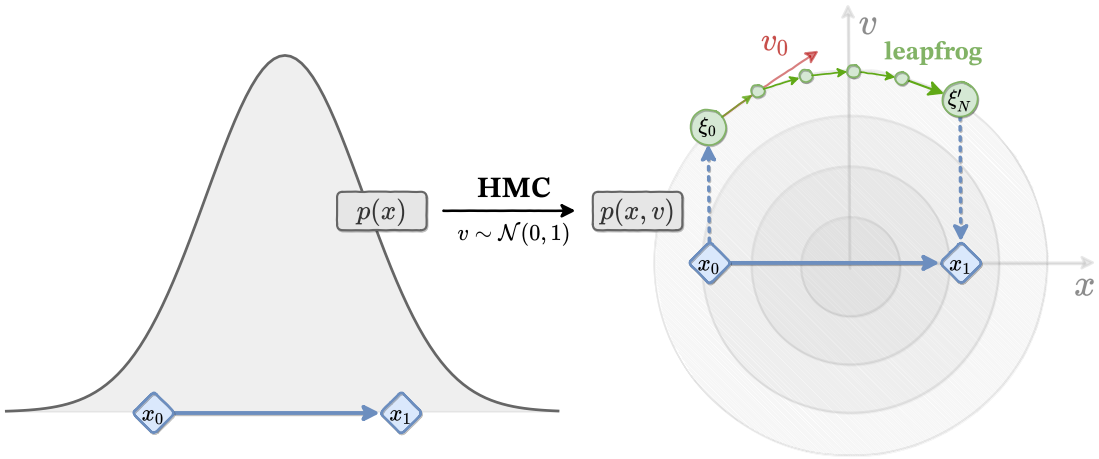


Figure 2: High-level overview of the HMC algorithm.

2.2 Issues with HMC

Re-sampling the momentum at the start of each trajectory causes the energy levels we explore to be randomly selected. This is reminiscient of the "random-walk" behavior of traditional MCMC and leads to a slow exploration of our target distribution (i.e. long autocorrelations). Additionally, the HMC sampler is known to have difficulty traversing low-density zones, resulting in poor performance for distributions which have multiple isolated modes. This is particularly relevant in the case of sampling topological quantities in lattice gauge models.

3. Generalizing HMC: LeapfrogLayers

In order to preserve the asymptotic behavior of HMC, our update must be explicitly reversible with a tractable Jacobian determinant. To simplify notation, we introduce two functions, Γ (Λ) to denote the $\mathbf{v}$ ($\mathbf{x}$) updates. As in HMC, we follow the general pattern of performing alternating updates of $\mathbf{v}$ and $\mathbf{x}$.

We can ensure the Jacobian is simple by splitting the $\mathbf{x}$ update into two parts and sequentially updating complementary subsets using a binary mask m and its complement $\bar{m}$. As in [2], we introduce $d \sim \mathcal{U}(+,-)$, distributed independently of both $\mathbf{x}$ and $\mathbf{v}$, to determine the "direction" of our update[1]. Here, we associate $+$ ($-$) with the forward (backward) direction and note that running sequential updates in opposite directions has the effect of inverting the update. We denote the complete state by $\xi = (\mathbf{x}, \mathbf{v}, \pm)$, with target density given by $p(\xi) = p(\mathbf{x})p(\mathbf{v})p(\pm)$.

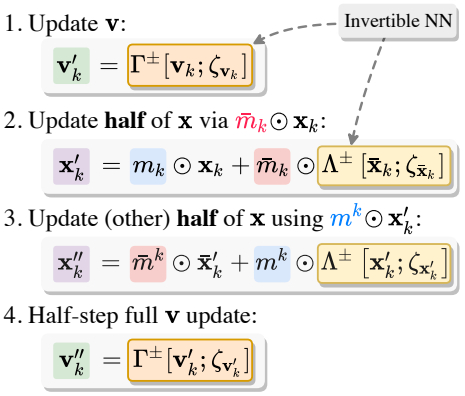
Explicitly, we can write this series of updates as[2]

$$\textbf{(a.)}\ \mathbf{v}' \leftarrow \Gamma^{\pm}[\mathbf{v};\, \zeta_{\mathbf{v}}] \qquad \textbf{(b.)}\ \mathbf{x}' \leftarrow m \odot \mathbf{x} + \bar{m} \odot \Lambda^{\pm}[x_{\bar{m}}; \zeta_{\bar{\mathbf{x}}}]$$
$$\textbf{(c.)}\ \mathbf{x}'' \leftarrow \bar{m} \odot \mathbf{x}' + m \odot \Lambda^{\pm}[x_m; \zeta_{\mathbf{x}'}] \qquad \textbf{(d.)}\ \mathbf{v}'' \leftarrow \Gamma^{\pm}[\mathbf{v}', \zeta_{\mathbf{v}'}]$$

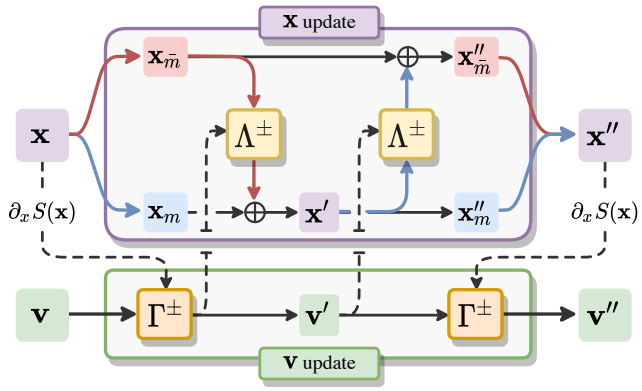
where $\zeta_{\bar{\mathbf{x}}} = [\bar{m} \odot \mathbf{x}, \mathbf{v}]$, $\zeta_{\mathbf{x}} = [m \odot \mathbf{x}, \mathbf{v}]$ ($\zeta_{\mathbf{v}} = [\mathbf{x}, \partial_{\mathbf{x}} S(\mathbf{x})]$) is independent of $\mathbf{x}$ ($\mathbf{v}$) and is passed as input to the update functions $\Lambda^{\pm}$ ($\Gamma^{\pm}$).

[1] As indicated by the superscript $\pm$ on $\Gamma^{\pm}, \Lambda^{\pm}$ in the update functions.

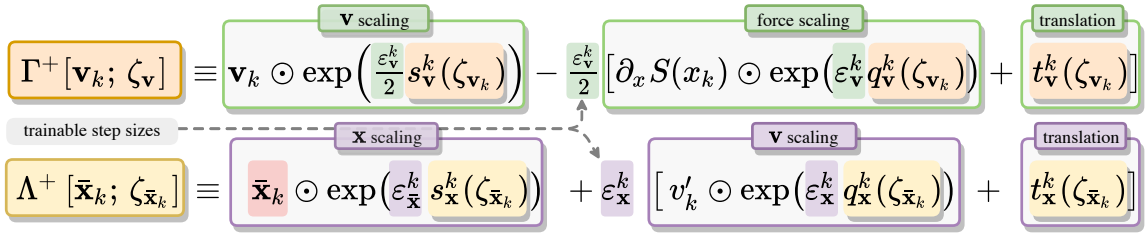
[2] Here we denote by $x_m = m \odot \mathbf{x}$ and $x_{\bar{m}} = \bar{m} \odot \mathbf{x}$ with $\mathbb{1} = m + \bar{m}$.



(a) Generalized leapfrog update.



(b) Illustration of the data flow through a leapfrog layer.



(c) Detailed view of the update functions Γ^+, Λ^+ for the k^{th} leapfrog step.

Figure 3: Illustrations of the generalized leapfrog update. In Figure 3a, 3c, we denote $\bar{\mathbf{x}}_k = \bar{m} \odot \mathbf{x}_k$.

3.1 Network Details

Normalizing flows [3] are an obvious choice for the structure of the update functions. These architectures are easily invertible while maintaining a tractable Jacobian determinant, and have also been shown to be effective at approximating complicated target distributions in high-dimensional spaces [2–14].

We maintain separate networks Γ, Λ with identical architectures for updating $\mathbf{v}$ and $\mathbf{x}$, respectively. Without loss of generality, we describe below the details of the $\mathbf{x}$ update for the forward $(d = +)$ direction, $\Lambda^+[x_{\bar{m}}; \zeta_{\bar{\mathbf{x}}}]$[3]. For simplicity, we describe the data flow through a single leapfrog layer, which takes as input $\zeta_{\bar{\mathbf{x}}} = (x_{\bar{m}}, \mathbf{v})$. For the 2D $U(1)$ model, the gauge links are encoded as $[\cos(x), \sin(x)]$ for $x \in [-\pi, \pi]$. Explicitly[4],

$$h_1 = \sigma\left(w_x \mathbf{x} + w_v \mathbf{v} + b_1\right) \quad \in \mathbb{R}^{n_1} \tag{12}$$

$$h_2 = \sigma\left(w_2 h_1 + b_2\right) \quad \in \mathbb{R}^{n_2} \tag{13}$$

$$\vdots$$

$$h_k = \sigma\left(w_k h_{k-1} + b_{k-1}\right) \quad \in \mathbb{R}^{n_k} \implies \tag{14}$$

$$\textbf{(a.)}\ s_x = \lambda_s \tanh\left(w_s h_k + b_s\right); \quad \textbf{(b.)}\ t_x = w_t h_k + b_t; \quad \textbf{(c.)}\ q_x = \lambda_q \tanh\left(w_q h_k + b_q\right);$$

where the outputs s_x, t_x, q_x are of the same dimensionality as $\mathbf{x}$, and λ_s, λ_q are trainable parameters. These outputs are then used to update $\mathbf{x}$, as shown in Figure 3.

[3] To get the update for the $d = -$ direction, we invert the update functions and run them in the opposite order.

[4] Here $\sigma(\cdot)$ is a generic nonlinear activation function acting independently on the elements of the output vector.

3.2 Training Details

Our goal in training the nework is to find a sampler that efficiently jumps between different topological charge sectors. This can be done by constructing a loss function that maximizes the expected squared charge difference between the initial (ξ) and proposal (ξ^*) configuration generated by the sampler. Explicitly, we define

$$\mathcal{L}_\theta(\xi^*, \xi) = A(\xi^*|\xi)(Q^*_{\mathbb{R}} - Q_{\mathbb{R}})^2 \tag{15}$$

where $Q_{\mathbb{R}} = \frac{1}{2\pi}\sum_P \sin x_P \in \mathbb{R}$ is a real-valued approximation of the usual (integer-valued) topological charge $Q_{\mathbb{Z}} \in \mathbb{Z}$. This ensures that our loss function is continuous which helps to simplify the training procedure. The acceptance probability

$$A(\xi^*|\xi) = \min\left\{1, \frac{p(\xi^*)}{p(\xi)}\mathcal{J}(\xi^*, \xi)\right\} \tag{16}$$

now includes an extra Jacobian factor $\mathcal{J}(\xi^*, \xi)$ which allows the inclusion of non-symplectic update steps. The Jacobian comes from the $v\,(x)$ scaling term in the $v\,(x)$ update, and is easily calculated. For completeness, the details of an individual training step are summarized in Sec 3.2.1.

3.2.1 Training Step

1. Resample $\mathbf{v} \sim \mathcal{N}(0, \mathbb{1})$, $d \sim \mathcal{U}(+, -)$, and construct initial state $\xi = (\mathbf{x}, \mathbf{v}, \pm)$
2. Generate the proposal configuration ξ^* by passing the initial state sequentially through N_{LF} *leapfrog layers*: $\xi \to \xi_1 \to \ldots \to \xi_{N_{\text{LF}}} = \xi^*$
3. Compute the Metropolis-Hastings acceptance $A(\xi^*|\xi) = \min\left\{1, \frac{p(\xi^*)}{p(\xi)}\mathcal{J}(\xi^*, \xi)\right\}$
4. Evaluate the loss function $\mathcal{L} \leftarrow \mathcal{L}_\theta(\xi^*, \xi)$, and backpropagate gradients to update weights
5. Evaluate Metropolis-Hastings criteria and assign the next state in the chain according to $\mathbf{x}_{t+1} \leftarrow \begin{cases}\mathbf{x}^* \text{ with prob. } A(\xi^*|\xi) \\ \mathbf{x} \text{ with prob. } 1 - A(\xi^*|\xi).\end{cases}$

3.3 Annealing

As an additional tool to help improve the quality of the trained sampler, we scale the action during the N_{T} training steps using the target distribution $p_t(x) \propto e^{-\gamma_t S(x)}$ for $t = 0, 1, \ldots, N_{\text{T}}$. The scale factors, γ_t, monotonically increase according to an *annealing schedule* up to $\gamma_{N_{\text{T}}} \equiv 1$, with $|\gamma_{i+1} - \gamma_i| \ll 1$. For $\gamma_i < 1$, this helps to rescale (shrink) the energy barriers between isolated modes, allowing the training to experience sufficient tunneling even when the final distribution is difficult to sample.

4. Results

We can measure the performance of our approach by comparing the integrated autocorrelation time against traditional HMC. We see in Figure 4 that our estimate of the integrated autocorrelation time is much shorter for the trained model across $\beta \in [2, 7]$. To better understand how these transformations effect the HMC update, we can look at how various quantities change over the course of a trajectory, as shown in Figure 6. We see from the plot of $\mathcal{H} - \sum \log |\mathcal{J}|$ in Figure 6c that the trained sampler artificially increases the energy towards the middle of the trajectory. This is analogous to reducing the coupling β during the trajectory, as can be seen in Figure 7b. In particular, we can see that for $\beta = 7$, the value of the plaquette at in the middle of the trajectory roughly corresponds to the expected value at $\beta = 3$, indicated by the horizontal dashed line. This effective reduction in the coupling constant allows our sampler to mix topological charge values in the middle of the trajectory, before returning to the stationary distribution at the end, as can be seen in Figure 6b. By looking at the variation in the average plaquette $\langle x_P - x_P^* \rangle$ over a trajectory for models trained at multiple values of beta we are able to better understand how our sampler behaves. This allows our trajectories to explore new regions of the target distribution which may have been previously inaccessible.

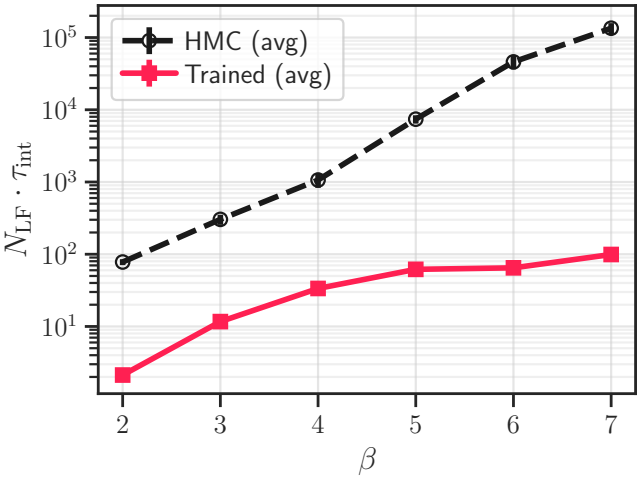


Figure 4: Estimated $\tau_{\mathrm{int}}^{Q_{\mathbb{R}}}$ vs β.

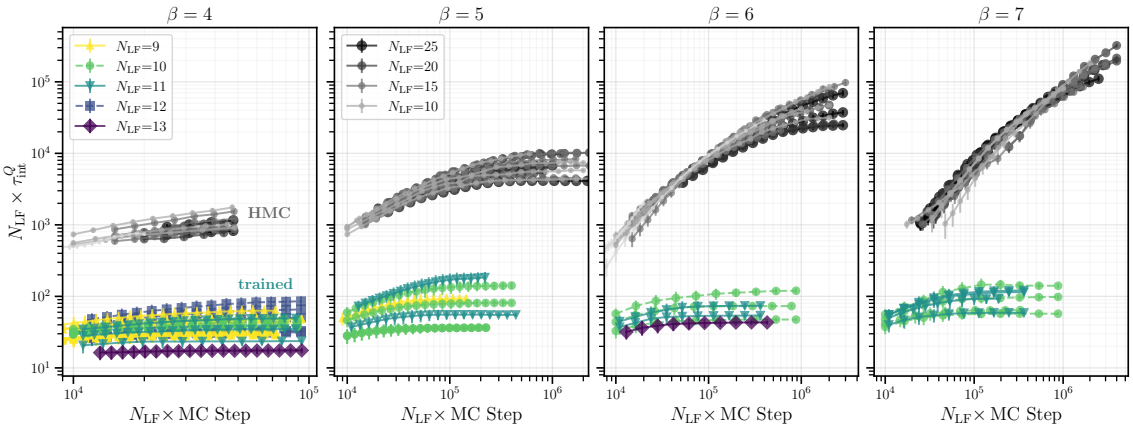
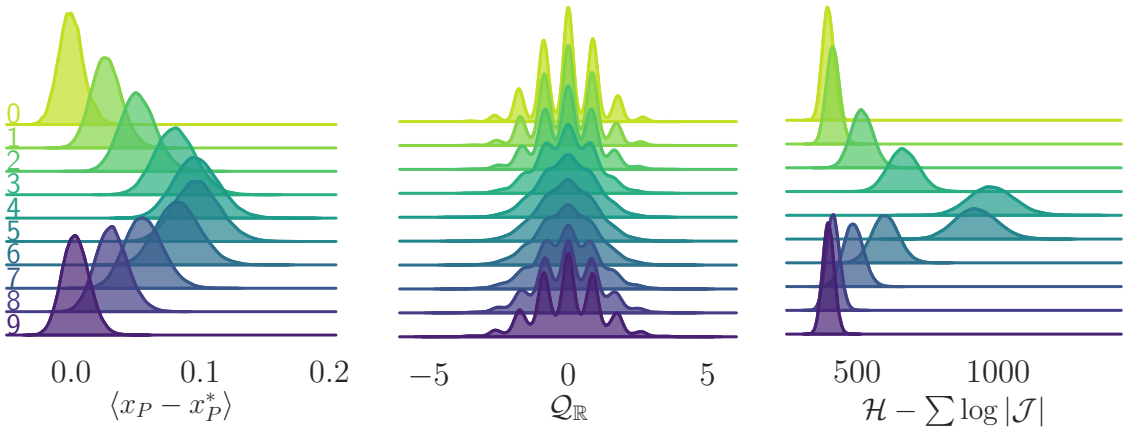
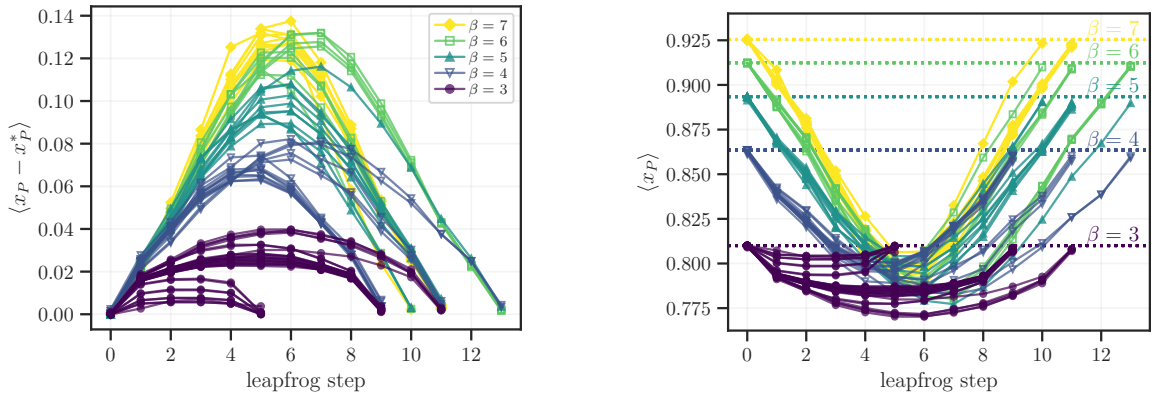


Figure 5: Comparison of the integrated autocorrelation time for trained models vs HMC with different trajectory lengths, N_{LF}, at $\beta = 4, 5, 6, 7$.



(a) Deviation in x_P. **(b)** Topological charge mixing $Q_{\mathbb{R}}$. **(c)** Artificial influx of energy.

Figure 6: Illustration of how different quantities evolve over a single trajectory.



(a) The deviation of x_P from the $V \to \infty$ limit, x_P^*. **(b)** Note x_P^* is indicated by the horizontal dashed lines.

Figure 7: Plots showing how the plaquette varies over a single trajectory for models trained at $\beta = 3, 4, \ldots, 7$.

5. Conclusion

In this work we have proposed a generalized sampling procedure for HMC that can be used for generating gauge configurations in the 2D $U(1)$ lattice model. We showed that our trained model is capable of outperforming traditional sampling techniques across a range of inverse coupling constants, as measured by the integrated autocorrelation time of the topological charge. By looking at the evolution of different quantities over the generalized leapfrog trajectory, we are able to gain insight into the mechanism driving this improved behavior.

6. Acknowledgments

This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration. This research was performed using resources of the Argonne Leadership Com-

puting Facility (ALCF), which is a DOE Office of Science User Facility supported under Contract DE_AC02–06CH11357. This work describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the work do not necessarily represent the views of the U.S. DOE or the United States Government. Results presented in this research were obtained using the Python [15], programming language and its many data science libraries [16–19].

References

[1] T. Eichhorn and C. Hoelbling, *Comparison of topology changing update algorithms*, 2112.05188.

[2] D. Levy, M.D. Hoffman and J. Sohl-Dickstein, *Generalizing Hamiltonian Monte Carlo with Neural Networks*, *arXiv e-prints* (2017) arXiv:1711.09268 [1711.09268].

[3] L. Dinh, J. Sohl-Dickstein and S. Bengio, *Density estimation using Real NVP*, *arXiv e-prints* (2016) arXiv:1605.08803 [1605.08803].

[4] S. Foreman, T. Izubuchi, L. Jin, X.-Y. Jin, J.C. Osborn and A. Tomiya, *HMC with Normalizing Flows*, 2112.01586.

[5] Z. Li, Y. Chen and F.T. Sommer, *A neural network MCMC sampler that maximizes proposal entropy*, 2010.03587.

[6] S. Foreman, X.-Y. Jin and J.C. Osborn, *Deep Learning Hamiltonian Monte Carlo*, 2105.03418.

[7] G. Kanwar, M.S. Albergo, D. Boyda, K. Cranmer, D.C. Hackett, S. Racanière et al., *Equivariant flow-based sampling for lattice gauge theory*, *Phys. Rev. Lett.* **125** (2020) 121601 [2003.06413].

[8] M.S. Albergo, D. Boyda, D.C. Hackett, G. Kanwar, K. Cranmer, S. Racanière et al., *Introduction to normalizing flows for lattice field theory*, 2101.08176.

[9] D. Boyda, G. Kanwar, S. Racanière, D.J. Rezende, M.S. Albergo, K. Cranmer et al., *Sampling using $SU(N)$ gauge equivariant flows*, *Phys. Rev. D* **103** (2021) 074504 [2008.05456].

[10] K. Neklyudov and M. Welling, *Orbital MCMC*, *arXiv e-prints* (2020) arXiv:2010.08047 [2010.08047].

[11] K. Neklyudov, M. Welling, E. Egorov and D. Vetrov, *Involutive mcmc: A unifying framework*, in *International Conference on Machine Learning*, p. arXiv:2006.16653, PMLR, 2020 [2006.16653].

[12] J. Dumas, A. Wehenkel, D. Lanaspeze, B. Cornélusse and A. Sutera, *You say normalizing flows i see bayesian networks*, 2006.00866.

[13] S.-H. Li and L. Wang, *Neural network renormalization group*, *Physical Review Letters* **121** (2018) .

[14] K.A. Nicoli, C.J. Anders, L. Funcke, T. Hartung, K. Jansen, P. Kessel et al., *Estimation of Thermodynamic Observables in Lattice Field Theories with Deep Generative Models*, 2007.07115.

[15] G. Van Rossum and F.L. Drake Jr, *Python Tutorial*, Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands (1995).

[16] T.A. Caswell, M. Droettboom, J. Hunter, E. Firing, A. Lee, J. Klymak et al., *matplotlib/matplotlib v3.1.0*, May, 2019. 10.5281/zenodo.2893252.

[17] C.R. Harris, K.J. Millman, S.J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau et al., *Array programming with NumPy*, *Nature* **585** (2020) 357.

[18] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro et al., "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems."

[19] F. Perez and B.E. Granger, *IPython: A System for Interactive Scientific Computing*, *Computing in Science and Engineering* **9** (2007) 21.