MLMC: Machine Learning Monte Carlo
for Lattice Gauge Theory

Sam Foreman
Xiao-Yong Jin, James C. Osborn
saforem2/lattice23

Background: MCMC

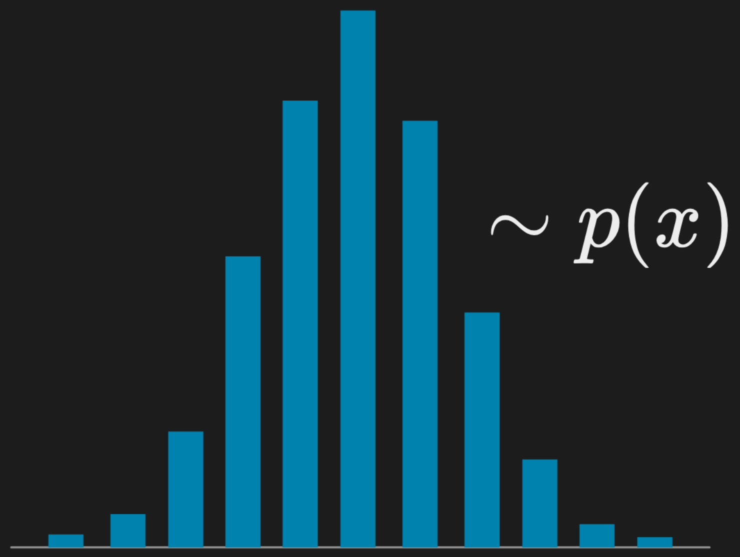
Markov Chain Monte Carlo (MCMC)

Goal

Generate **independent** samples $\{x_i\}$, such that[1]

$$\{x_i\} \sim p(x) \propto e^{-S(x)}$$

where $S(x)$ is the *action* (or potential energy)



- Want to calculate observables $\mathcal{O}$:
  $\langle\mathcal{O}\rangle \propto \int [\mathcal{D}x] \; \mathcal{O}(x)\, p(x)$

If these were independent, we could approximate: $\langle\mathcal{O}\rangle \simeq \frac{1}{N}\sum_{n=1}^{N}\mathcal{O}(x_n)$

$$\sigma_{\mathcal{O}}^2 = \frac{1}{N}\mathrm{Var}[\mathcal{O}(x)] \Longrightarrow \sigma_{\mathcal{O}} \propto \frac{1}{\sqrt{N}}$$

1. Here, $\sim$ means "is distributed according to"

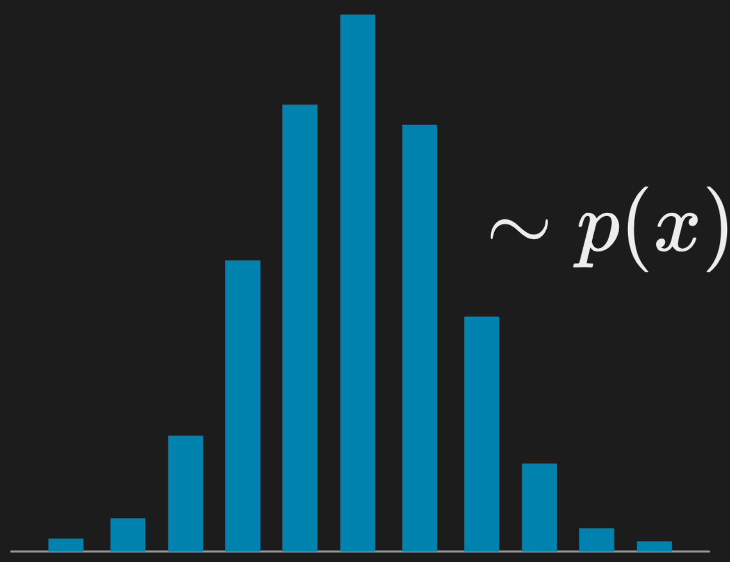
Markov Chain Monte Carlo (MCMC)

Goal

Generate **independent** samples $\{x_i\}$, such that[1]

$$\{x_i\} \sim p(x) \propto e^{-S(x)}$$

where $S(x)$ is the *action* (or potential energy)



- Want to calculate observables $\mathcal{O}$:
  $\langle\mathcal{O}\rangle \propto \int [\mathcal{D}x] \;\; \mathcal{O}(x)\, p(x)$

Instead, nearby configs are correlated, and we incur a factor of $\tau_{\mathrm{int}}^{\mathcal{O}}$:

$$\sigma_{\mathcal{O}}^2 = \frac{\tau_{\mathrm{int}}^{\mathcal{O}}}{N} \mathrm{Var}[\mathcal{O}(x)]$$

1. Here, $\sim$ means “is distributed according to”

Background: HMC

Hamiltonian Monte Carlo (HMC)

- Want to (sequentially) construct a chain of states:

$$x_0 \to x_1 \to x_i \to \cdots \to x_N$$

such that, as $N \to \infty$:

$$\{x_i, x_{i+1}, x_{i+2}, \cdots, x_N\} \xrightarrow{N\to\infty} p(x) \propto e^{-S(x)}$$

</> **Trick**

- Introduce fictitious momentum $v \sim \mathcal{N}(0, 1)$
  - Normally distributed **independent** of x, i.e.

$$p(x, v) = p(x)\, p(v) \propto e^{-S(x)} e^{-\frac{1}{2}v^T v} = e^{-\left[S(x) + \frac{1}{2}v^T v\right]} = e^{-H(x,v)}$$

Hamiltonian Monte Carlo (HMC)

- **Idea**: Evolve the $(\dot{x}, \dot{v})$ system to get new states $\{x_i\}$ **!**
- Write the **joint distribution** $p(x, v)$:

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

</> Hamiltonian Dynamics

$$H = S[x] + \frac{1}{2}v^T v \Longrightarrow$$

$$\dot{x} = +\partial_v H, \;\; \dot{v} = -\partial_x H$$

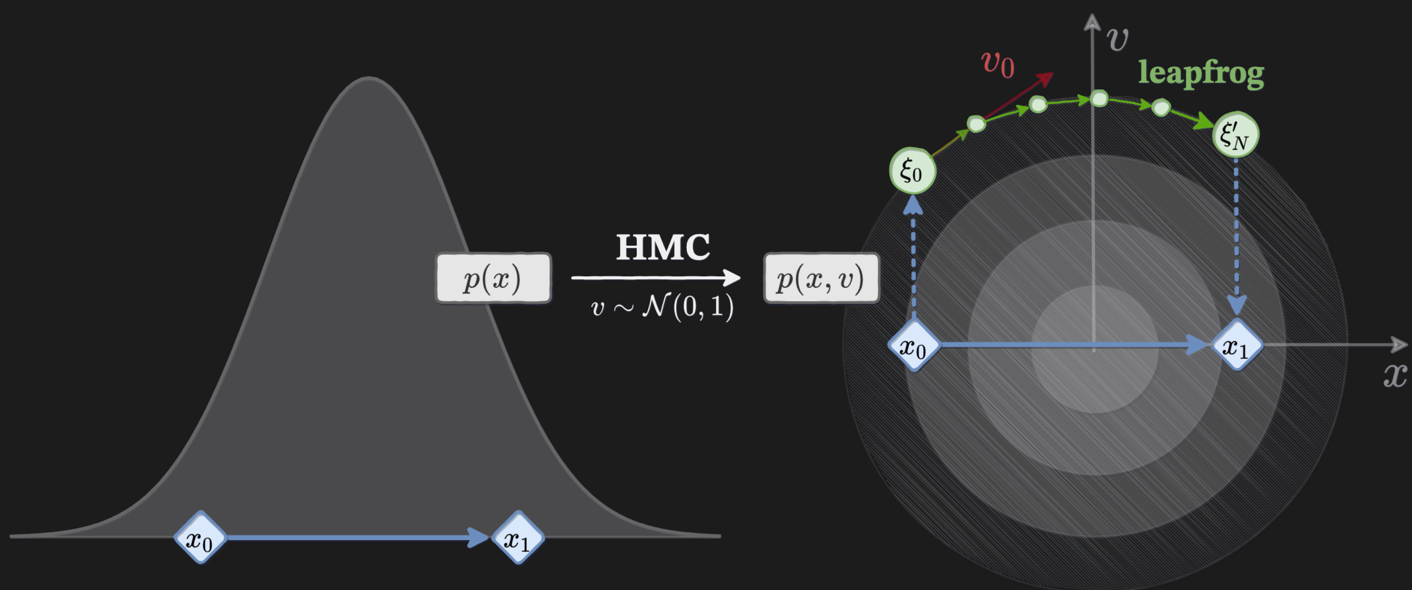


Figure 1: Overview of HMC algorithm

Leapfrog Integrator (HMC)

Hamiltonian Dynamics

$$(\dot{x}, \dot{v}) = (\partial_v H, -\partial_x H)$$

Leapfrog Step

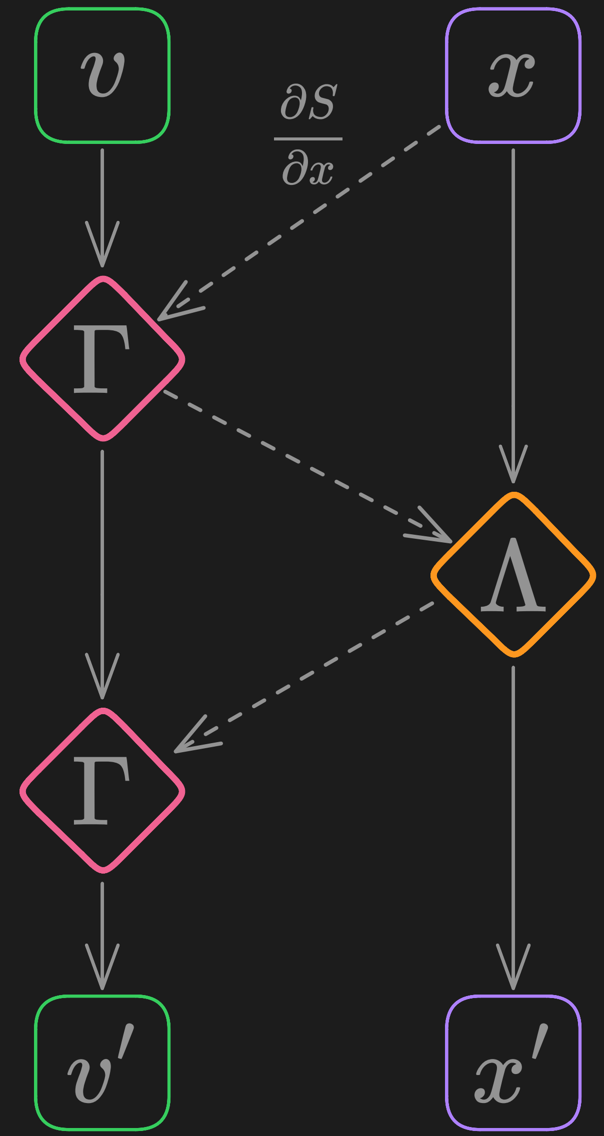
`input` $(x, v) \rightarrow (x', v')$ `output`

$$\tilde{v} := \Gamma(x, v) \; = v - \frac{\varepsilon}{2}\partial_x S(x)$$
$$x' := \Lambda(x, \tilde{v}) \; = x + \varepsilon\, \tilde{v}$$
$$v' := \Gamma(x', \tilde{v}) = \tilde{v} - \frac{\varepsilon}{2}\partial_x S(x')$$

Warning!

Resample $v_0 \sim \mathcal{N}(0, 1)$
at the beginning of each trajectory

Note: $\partial_x S(x)$ is the *force*



HMC Update

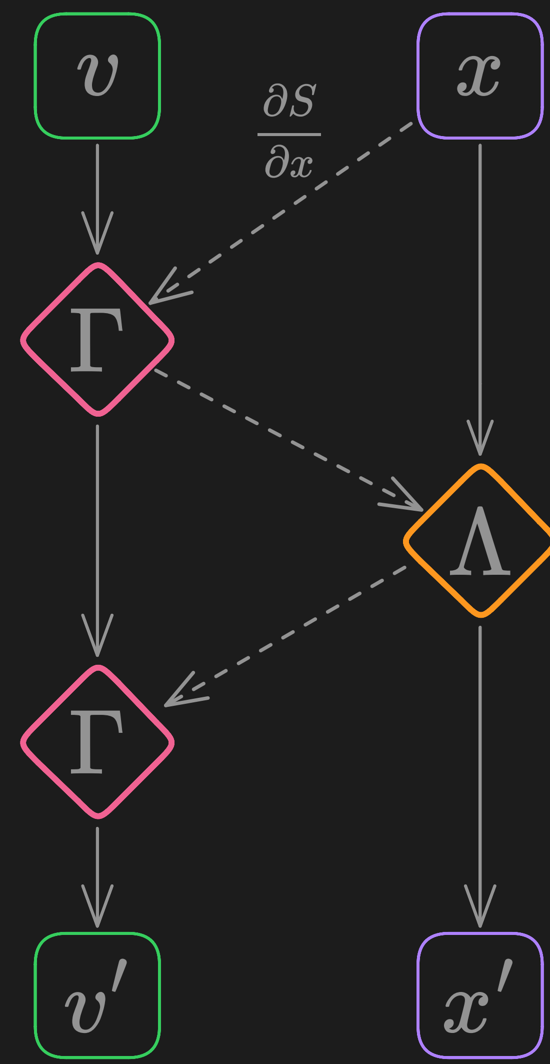
- We build a trajectory of N_{LF} **leapfrog steps**[1]

$$(x_0, v_0) \to (x_1, v_1) \to \cdots \to (x', v')$$

- And propose x' as the next state in our chain

$$\Gamma : (x, v) \to v' := v - \frac{\varepsilon}{2}\partial_x S(x)$$
$$\Lambda : (x, v) \to x' := x + \varepsilon v$$

- We then accept / reject x' using Metropolis-Hastings criteria,
$A(x'|x) = \min\left\{1, \frac{p(x')}{p(x)}\left|\frac{\partial x'}{\partial x}\right|\right\}$



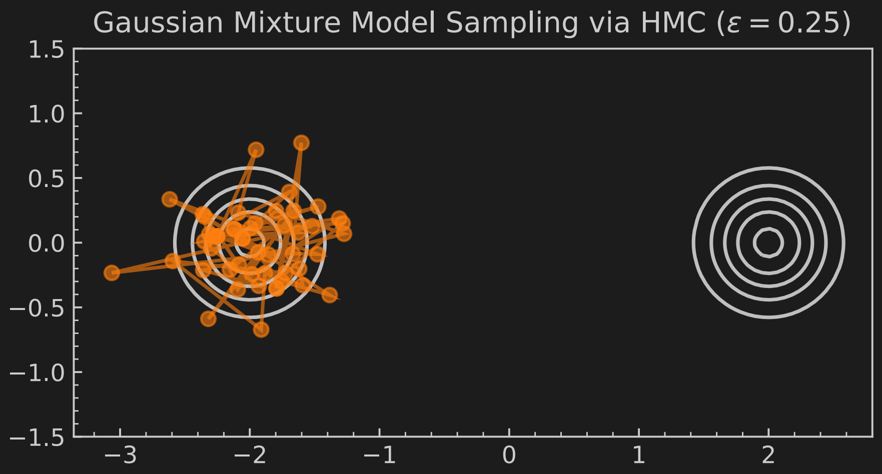
1. We **always** start by resampling the momentum, $v_0 \sim \mathcal{N}(0, 1)$

HMC Demo

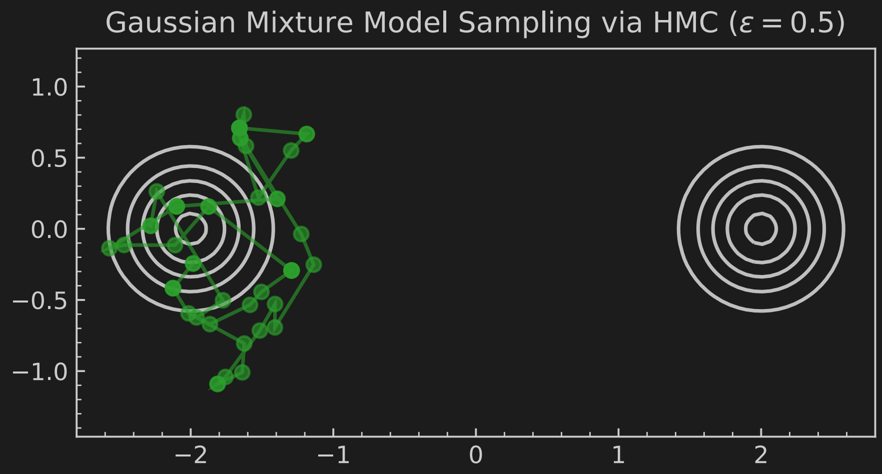
Figure 2: HMC Demo

Issues with HMC

- What do we want in a good sampler?
  - **Fast mixing** (small autocorrelations)
  - **Fast burn-in** (quick convergence)
- Problems with HMC:
  - Energy levels selected randomly $\longrightarrow$ **slow mixing**
  - Cannot easily traverse low-density zones $\longrightarrow$ **slow convergence**



HMC Samples with $\varepsilon = 0.25$



HMC Samples with $\varepsilon = 0.5$

Figure 3: HMC Samples generated with varying step sizes ε

Topological Freezing

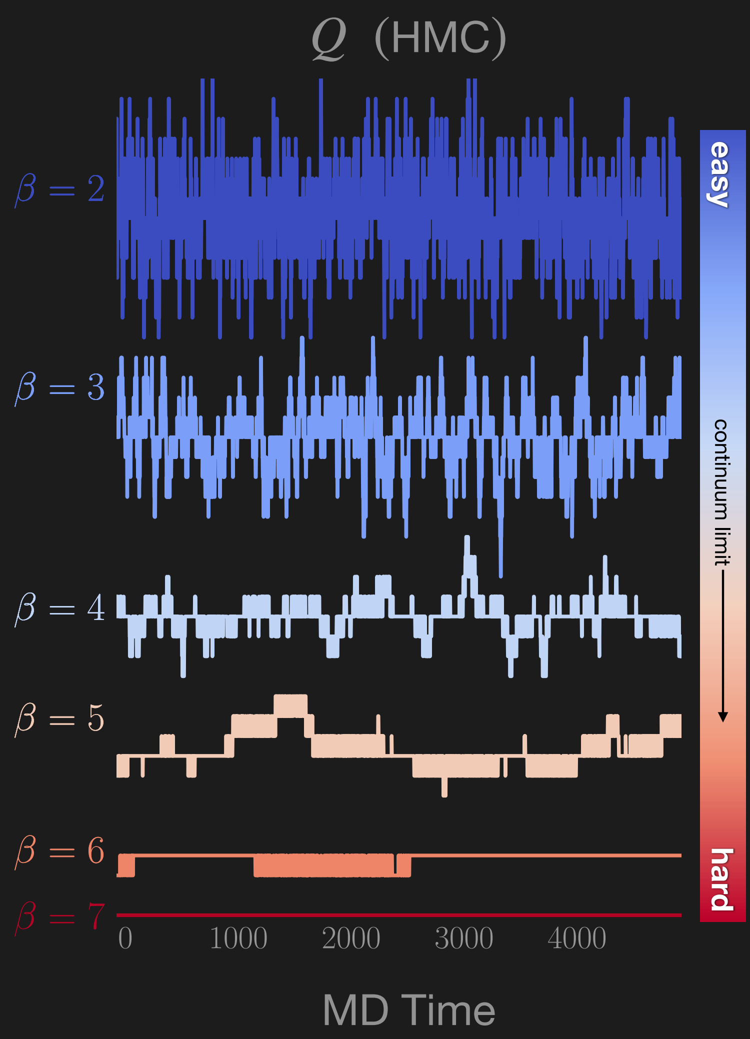
Topological Charge:

$$Q = \frac{1}{2\pi}\sum_{P}\left\lfloor x_P\right\rfloor \in \mathbb{Z}$$

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

Critical Slowing Down

- Q gets stuck!
  - as $\beta \longrightarrow \infty$:
    - $Q \longrightarrow \text{const.}$
    - $\delta Q = (Q^* - Q) \rightarrow 0 \Longrightarrow$
  - # configs required to estimate errors **grows exponentially**: $\tau_{\mathrm{int}}^{Q} \longrightarrow \infty$



Note $\delta Q \rightarrow 0$ at increasing β

Can we do better?

- Introduce two (**invertible NNs**) `vNet` and `xNet`[1]:
  - `vNet:` $(x, F) \longrightarrow (s_v,\ t_v,\ q_v)$
  - `xNet:` $(x, v) \longrightarrow (s_x,\ t_x,\ q_x)$
- Use these (s, t, q) in the *generalized* MD update:
  - $\Gamma_\theta^\pm : (x, v) \xrightarrow{s_v, t_v, q_v} (x, v')$
  - $\Lambda_\theta^\pm : (x, v) \xrightarrow{s_x, t_x, q_x} (x', v)$

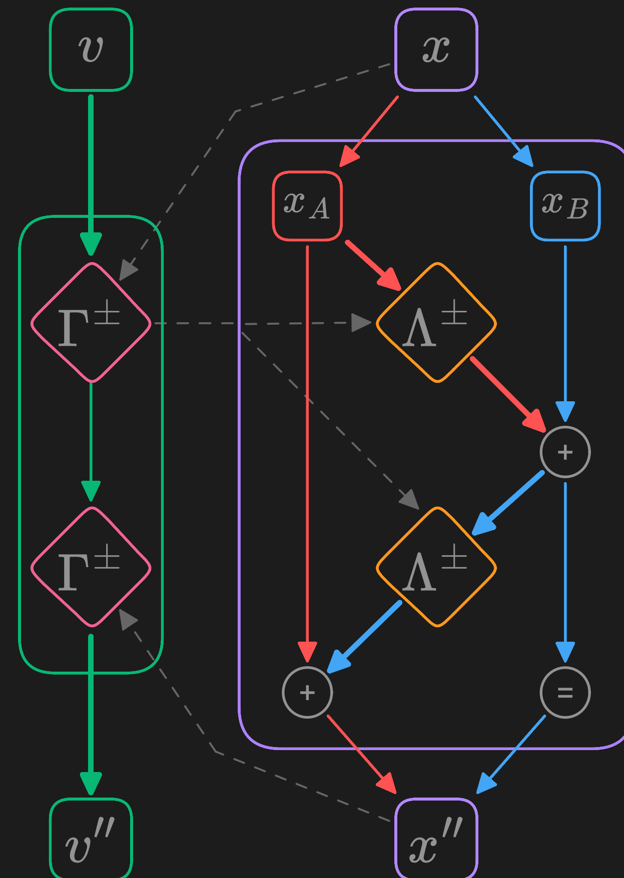


Figure 4: Generalized MD update where $\Lambda_\theta^\pm$, $\Gamma_\theta^\pm$ are **invertible NNs**

1. L2HMC: (Foreman, Jin, and Osborn 2021, 2022)

L2HMC: Generalizing the MD Update

L2HMC Update

- Introduce $d \sim \mathcal{U}(\pm)$ to determine the direction[1] of our update

1. $v' = \Gamma^{\pm}(x, v)$ — update v
2. $x' = x_B + \Lambda^{\pm}(x_A, v')$ — update first **half**: x_A
3. $x'' = x'_A + \Lambda^{\pm}(x'_B, v')$ — update other half: x_B
4. $v'' = \Gamma^{\pm}(x'', v')$ — update v

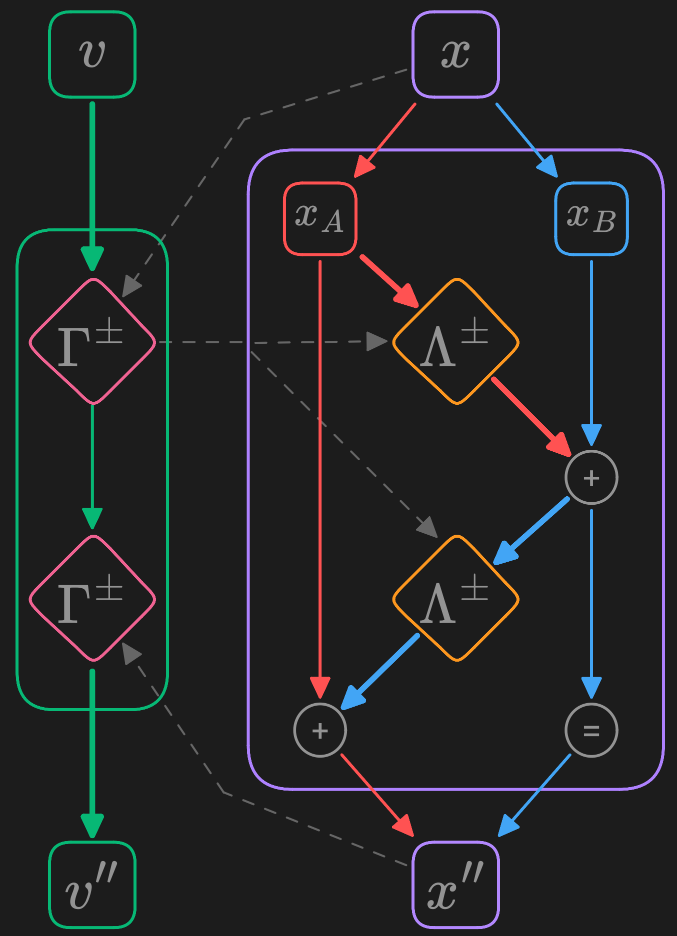


Figure 5: Generalized MD update with $\Lambda_\theta^{\pm}$, $\Gamma_\theta^{\pm}$ **invertible NNs**

1. d resampled at start of each trajectory, to ensure reversibility

L2HMC: Leapfrog Layer

Invertible NN

1. Update $\mathbf{v}$:

$$\mathbf{v}' = \Gamma^{\pm}[\mathbf{v};\zeta_{\mathbf{v}}]$$

2. Update **half** of $\mathbf{x}$ via $\bar{m}_k \odot \mathbf{x}_k$:

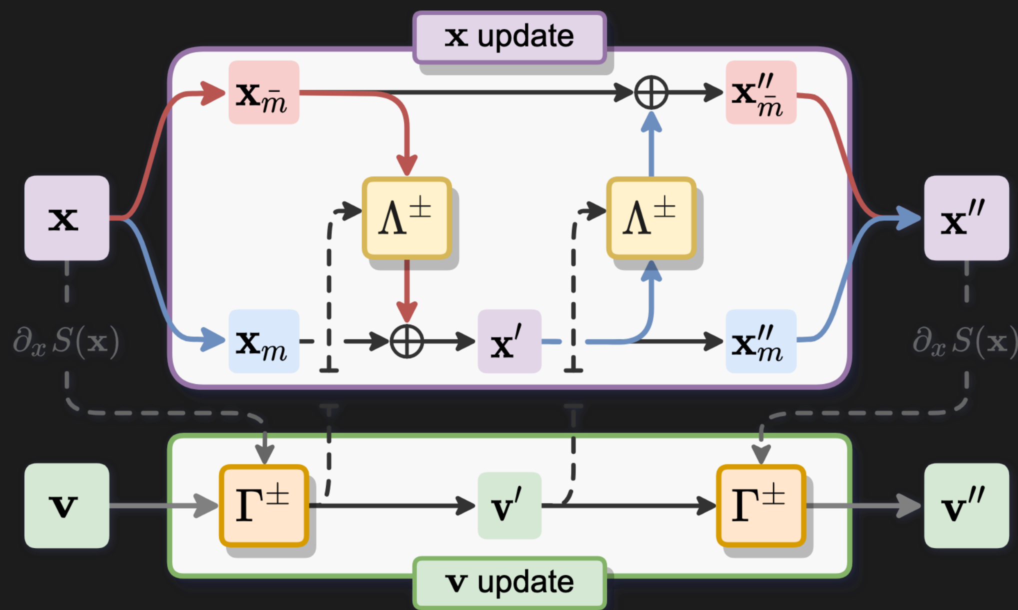
$$\mathbf{x}' = \mathbf{x}_m + \bar{m}\odot\Lambda^{\pm}\left[\mathbf{x}_{\bar{m}};\zeta_{\bar{\mathbf{x}}_k}\right]$$

3. Update (other) **half** via $m^k \odot \mathbf{x}'_k$:

$$\mathbf{x}'' = \mathbf{x}'_m + \bar{m}\odot\Lambda^{\pm}\left[\mathbf{x}'_m;\zeta_{\mathbf{x}'}\right]$$

4. Half-step full $\mathbf{v}$ update:

$$\mathbf{v}'' = \Gamma^{\pm}[\mathbf{v}';\zeta_{\mathbf{v}'}]$$



v scaling | force scaling | translation

$$\Gamma^{+}[\mathbf{v}_k;\ \zeta_{\mathbf{v}}] \equiv \mathbf{v}_k \odot \exp\left(\frac{\varepsilon^k_{\mathbf{v}}}{2} s^k_{\mathbf{v}}(\zeta_{\mathbf{v}_k})\right) - \frac{\varepsilon^k_{\mathbf{v}}}{2}\left[\partial_x S(x_k)\odot\exp\left(\varepsilon^k_{\mathbf{v}} q^k_{\mathbf{v}}(\zeta_{\mathbf{v}_k})\right) + t^k_{\mathbf{v}}(\zeta_{\mathbf{v}_k})\right]$$

trainable step sizes

x scaling | v scaling | translation

$$\Lambda^{+}[\bar{\mathbf{x}}_k;\ \zeta_{\bar{\mathbf{x}}_k}] \equiv \bar{\mathbf{x}}_k \odot \exp\left(\varepsilon^k_{\bar{\mathbf{x}}} s^k_{\mathbf{x}}(\zeta_{\bar{\mathbf{x}}_k})\right) + \varepsilon^k_{\mathbf{x}}\left[v'_k \odot \exp\left(\varepsilon^k_{\mathbf{x}} q^k_{\mathbf{x}}(\zeta_{\bar{\mathbf{x}}_k})\right) + t^k_{\mathbf{x}}(\zeta_{\bar{\mathbf{x}}_k})\right]$$

L2HMC Update

Algorithm

1. `input`: x
   - Resample: $v \sim \mathcal{N}(0, 1)$; $d \sim \mathcal{U}(\pm)$
   - Construct initial state: $\xi = (x, v, \pm)$
2. `forward`: Generate **proposal** ξ' by passing initial ξ through N_{LF} leapfrog layers

$$\xi \xrightarrow{\text{LF layer}} \xi_1 \longrightarrow \cdots \longrightarrow \xi_{N_{\mathrm{LF}}} = \xi' := (x'', v'')$$

   - Accept / Reject:

$$A(\xi'|\xi) = \min\left\{1, \frac{\pi(\xi')}{\pi(\xi)} \left|\mathcal{J}(\xi', \xi)\right|\right\}$$

3. `backward` (if training):
   - Evaluate the **loss function**[1] $\mathcal{L} \leftarrow \mathcal{L}_\theta(\xi', \xi)$ and backprop
4. `return`: x_{i+1}
   Evaluate MH criteria (1) and return accepted config,

$$x_{i+1} \leftarrow \begin{cases} x'' & \text{w/ prob } A(\xi''|\xi) \quad ✅ \\ x & \text{w/ prob } 1 - A(\xi''|\xi) \quad 🚫 \end{cases}$$

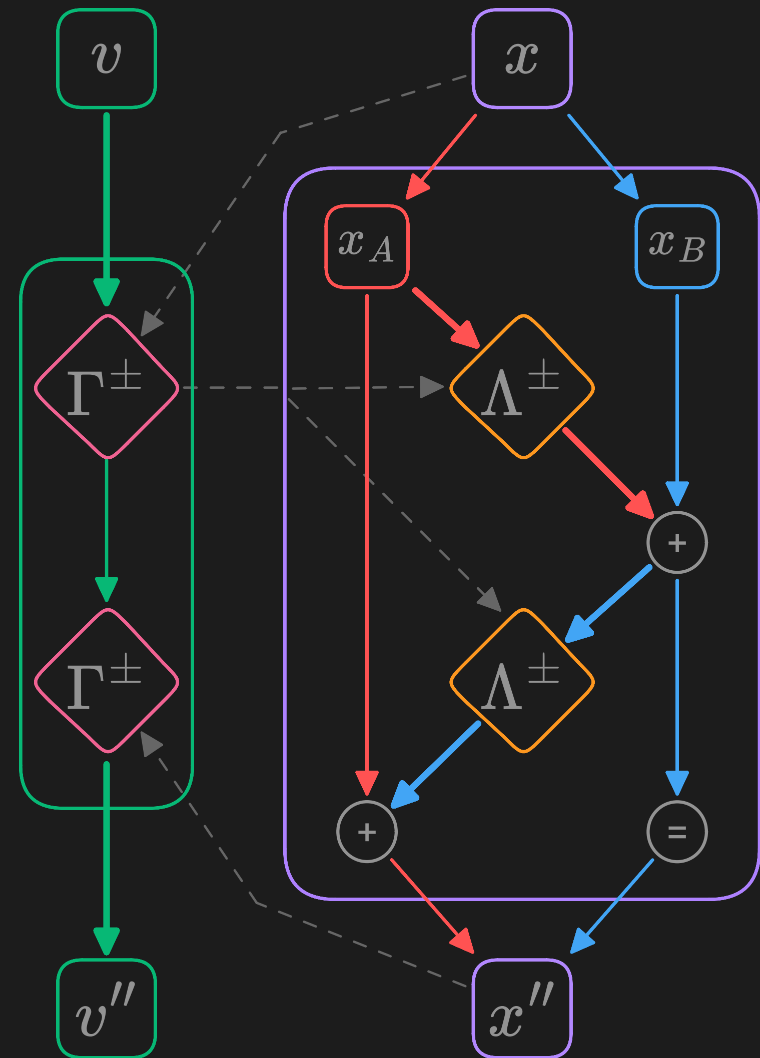


Figure 6: **Leapfrog Layer** used in generalized MD update

1. For simple $\mathbf{x} \in \mathbb{R}^2$ example, $\mathcal{L}_\theta = A(\xi^*|\xi) \cdot (\mathbf{x}^* - \mathbf{x})^2$

4D $SU(3)$ Model

Link Variables

- Write link variables $U_\mu(x) \in SU(3)$:

$$U_\mu(x) = \exp\left[i\,\omega_\mu^k(x)\lambda^k\right] = e^{iQ}, \quad \text{with} \quad Q \in \mathfrak{su}(3)$$

where $\omega_\mu^k(x) \in \mathbb{R}$, and λ^k are the generators of $SU(3)$

Conjugate Momenta

- Introduce $P_\mu(x) = P_\mu^k(x)\lambda^k$ conjugate to $\omega_\mu^k(x)$

Wilson Action

$$S_G = -\frac{\beta}{6}\sum \mathrm{Tr}\left[U_{\mu\nu}(x) + U_{\mu\nu}^\dagger(x)\right]$$

where $U_{\mu\nu}(x) = U_\mu(x)U_\nu(x+\hat{\mu})U_\mu^\dagger(x+\hat{\nu})U_\nu^\dagger(x)$

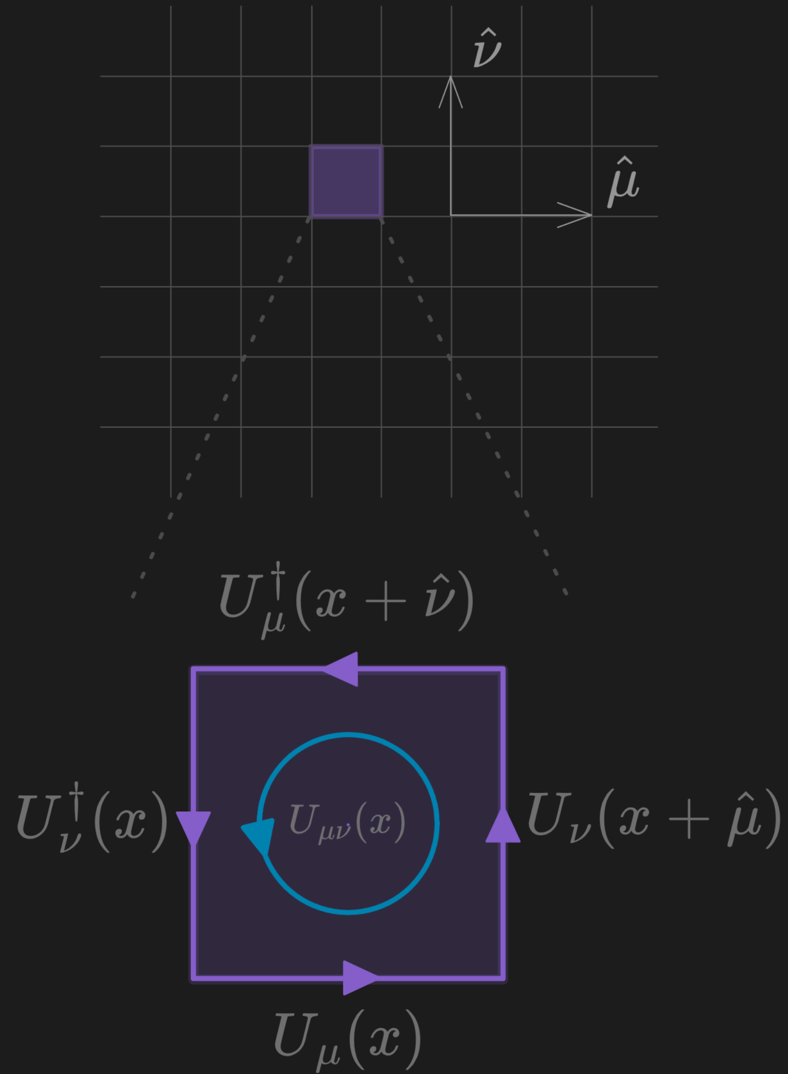


Figure 7: Illustration of the lattice

HMC: 4D $SU(3)$

Hamiltonian: $H[P,U] = \frac{1}{2}P^2 + S[U] \Longrightarrow$

- U update: $\frac{d\omega^k}{dt} = \frac{\partial H}{\partial P^k}$

$$\frac{d\omega^k}{dt}\lambda^k = P^k\lambda^k \Longrightarrow \frac{dQ}{dt} = P$$

$$Q(\varepsilon) = Q(0) + \varepsilon P(0) \Longrightarrow$$

$$-i\,\log U(\varepsilon) = -i\,\log U(0) + \varepsilon P(0)$$

$$U(\varepsilon) = e^{i\,\varepsilon P(0)}U(0) \Longrightarrow$$

$$\Lambda:\ U \longrightarrow U' := e^{i\varepsilon P'}U$$

- P update: $\frac{dP^k}{dt} = -\frac{\partial H}{\partial \omega^k}$

$$\frac{dP^k}{dt} = -\frac{\partial H}{\partial \omega^k} = -\frac{\partial H}{\partial Q} = -\frac{dS}{dQ} \Longrightarrow$$

$$P(\varepsilon) = P(0) - \varepsilon \left.\frac{dS}{dQ}\right|_{t=0}$$

$$= P(0) - \varepsilon\, F[U]$$

$$\Gamma:\ P \longrightarrow P' := P - \frac{\varepsilon}{2}F[U]$$

ε is the step size

$F[U]$ is the force term

HMC: 4D $SU(3)$

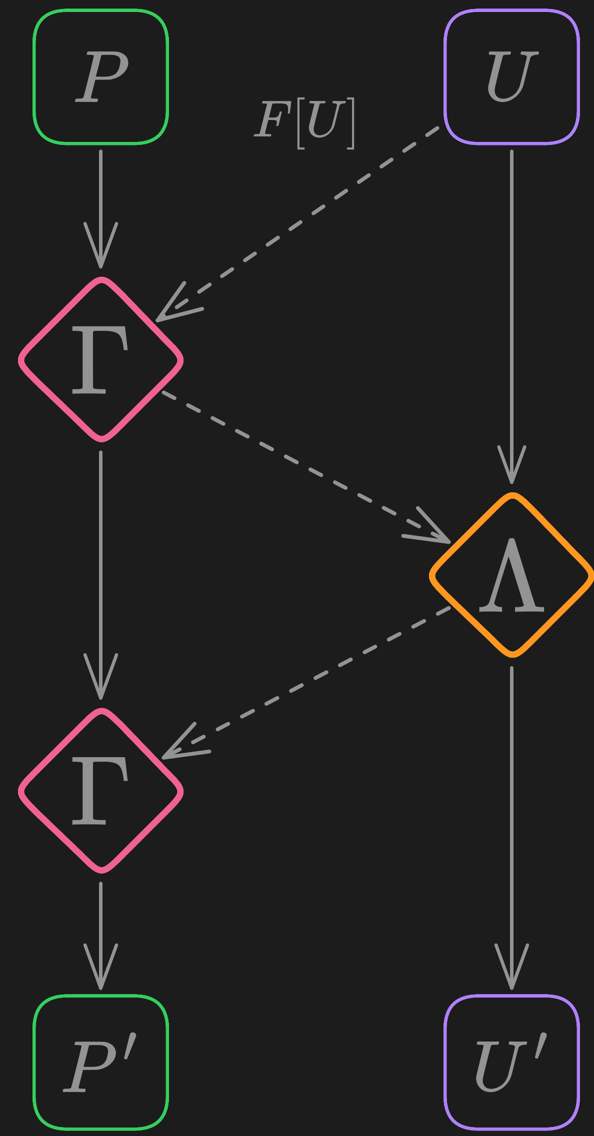
- Momentum Update:

$$\Gamma : P \longrightarrow P' := P - \frac{\varepsilon}{2} F[U]$$

- Link Update:

$$\Lambda : U \longrightarrow U' := e^{i\varepsilon P'} U$$

- We maintain a batch of `Nb` lattices, all updated in parallel
  - U`.dtype = complex128`
  - U`.shape`
    `= [Nb, 4, Nt, Nx, Ny, Nz, 3, 3]`



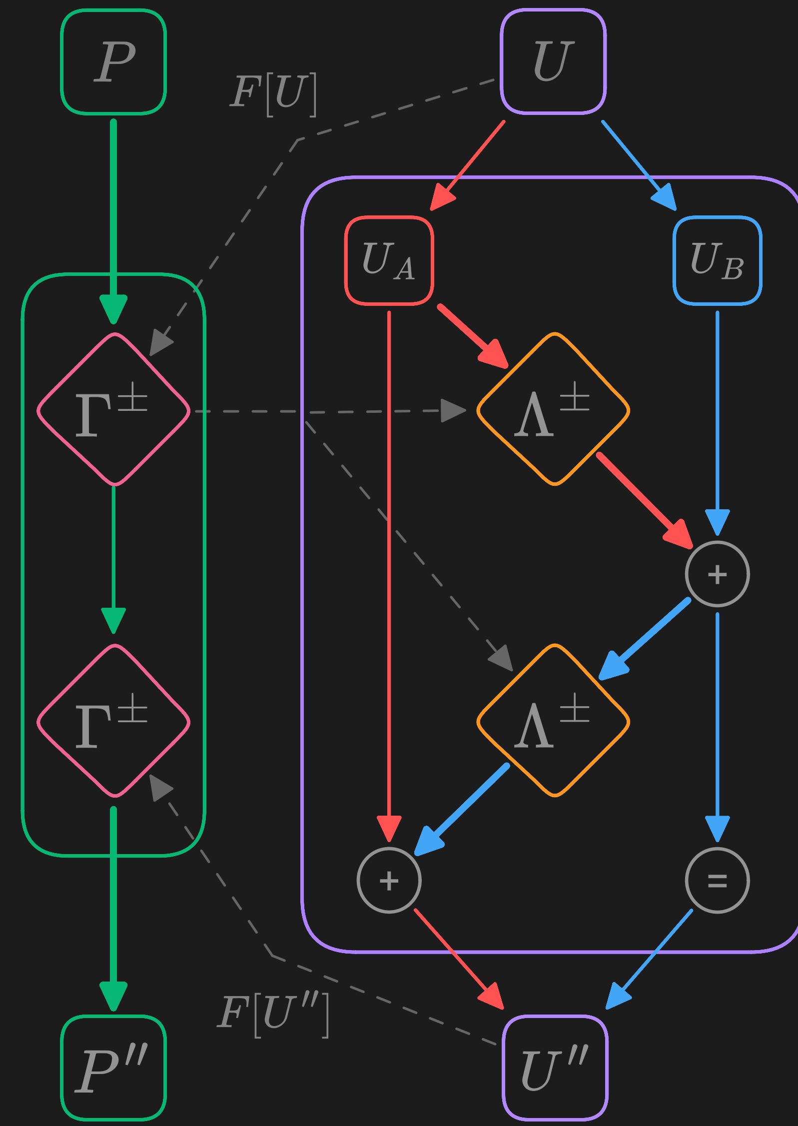
Networks 4D $SU(3)$

U-Network:

`UNet:` $(U, P) \longrightarrow (s_U,\, t_U,\, q_U)$

P-Network:

`PNet:` $(U, P) \longrightarrow (s_P,\, t_P,\, q_P)$



Networks 4D $SU(3)$

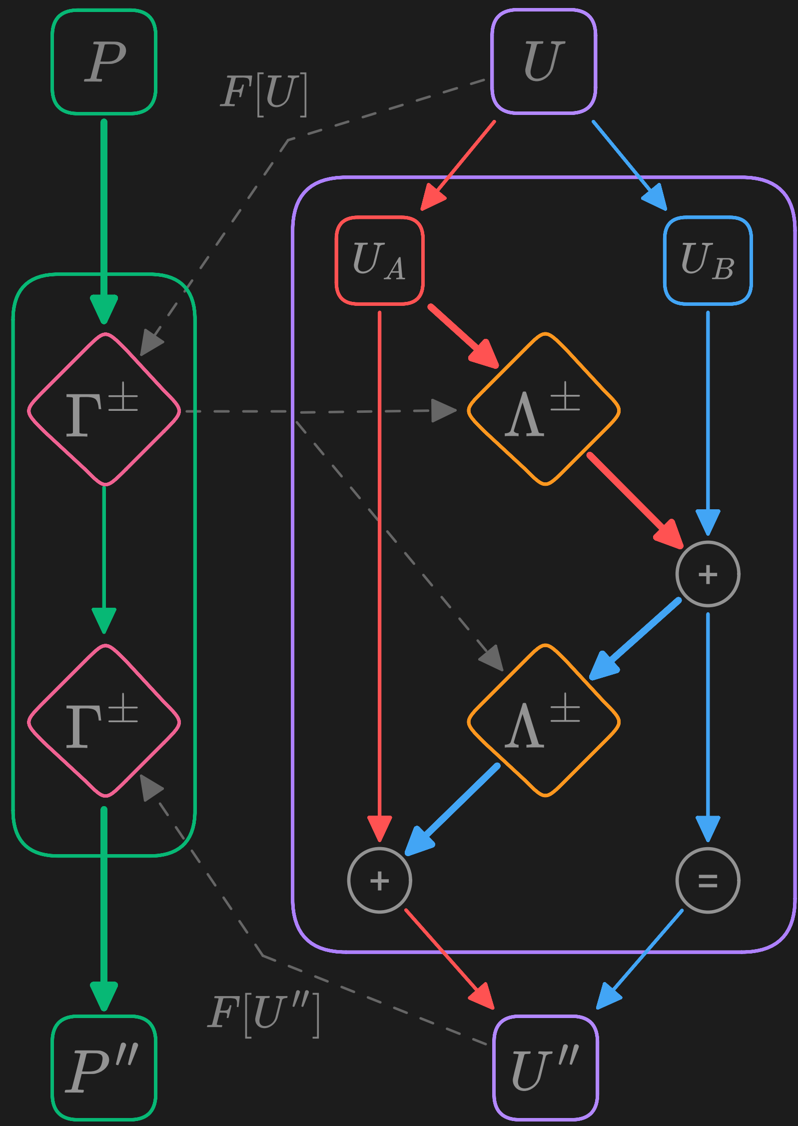
U-Network:

`UNet:` $(U, P) \longrightarrow (s_U,\, t_U,\, q_U)$

P-Network:

`PNet:` $(U, P) \longrightarrow (s_P,\, t_P,\, q_P)$

↑
let's look at this



P-Network (pt. 1)

$(U, F) \longrightarrow$ P-Network $\longrightarrow (s_P, t_P, q_P)$

- input[1]: $(U, F) := (e^{iQ}, F)$

$$h_0 = \sigma\left(w_Q Q + w_F F + b\right)$$
$$h_1 = \sigma\left(w_1 h_0 + b_1\right)$$
$$\vdots$$
$$h_n = \sigma\left(w_{n-1} h_{n-2} + b_n\right)$$
$$z := \sigma\left(w_n h_{n-1} + b_n\right) \longrightarrow$$

- output[2]: (s_P, t_P, q_P)
  - $s_P = \lambda_s \tanh(w_s z + b_s)$
  - $t_P = w_t z + b_t$
  - $q_P = \lambda_q \tanh(w_q z + b_q)$

1. $\sigma(\cdot)$ denotes an activation function
2. $\lambda_s,\ \lambda_q \in \mathbb{R}$ are trainable parameters

P-Network (pt. 2)

$$(U, F) \longrightarrow \boxed{\texttt{P-Network}} \longrightarrow (s_P, t_P, q_P)$$

- Use (s_P, t_P, q_P) to update $\Gamma^{\pm} : (U, P) \to (U, P_{\pm})$[1]:
  - forward $(d = +)$:

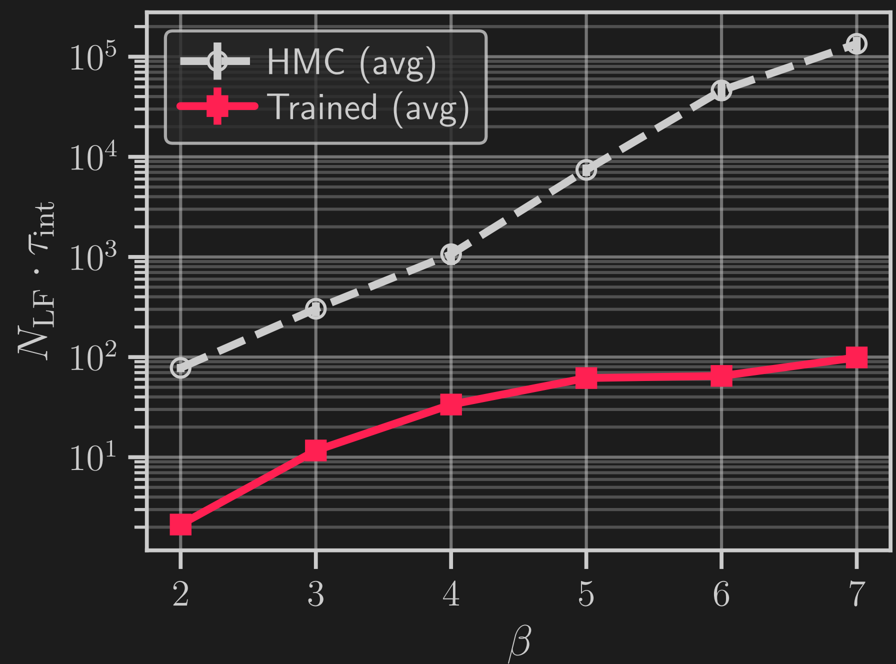
$$\Gamma^{+}(U, P) := P_{+} = P \cdot e^{\frac{\varepsilon}{2} s_P} - \frac{\varepsilon}{2}\left[F \cdot e^{\varepsilon q_P} + t_P\right]$$

  - backward $(d = -)$:

$$\Gamma^{-}(U, P) := P_{-} = e^{-\frac{\varepsilon}{2} s_P}\left\{P + \frac{\varepsilon}{2}\left[F \cdot e^{\varepsilon q_P} + t_P\right]\right\}$$

1. Note that $\left(\Gamma^{+}\right)^{-1} = \Gamma^{-}$, i.e. $\Gamma^{+}\left[\Gamma^{-}(U, P)\right] = \Gamma^{-}\left[\Gamma^{+}(U, P)\right] = (U, P)$

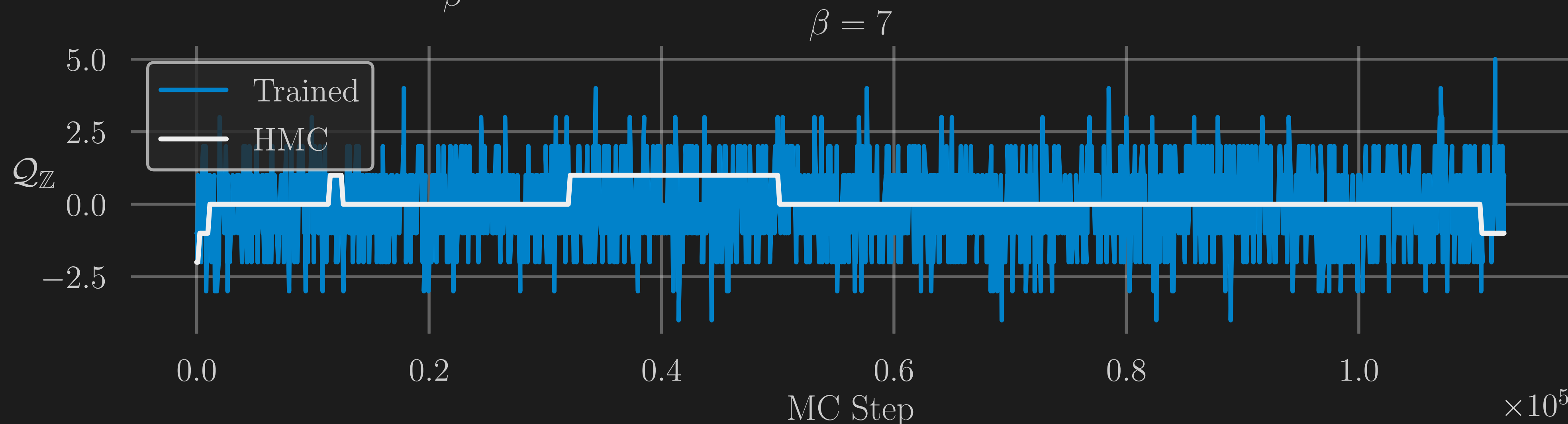
2D $U(1)$: Integrated Autocorrelation time: τ_{int}



Improvement

We can measure the performance by comparing τ_{int} for the **trained model** vs. **HMC**.

Note: lower is better



Interpretation

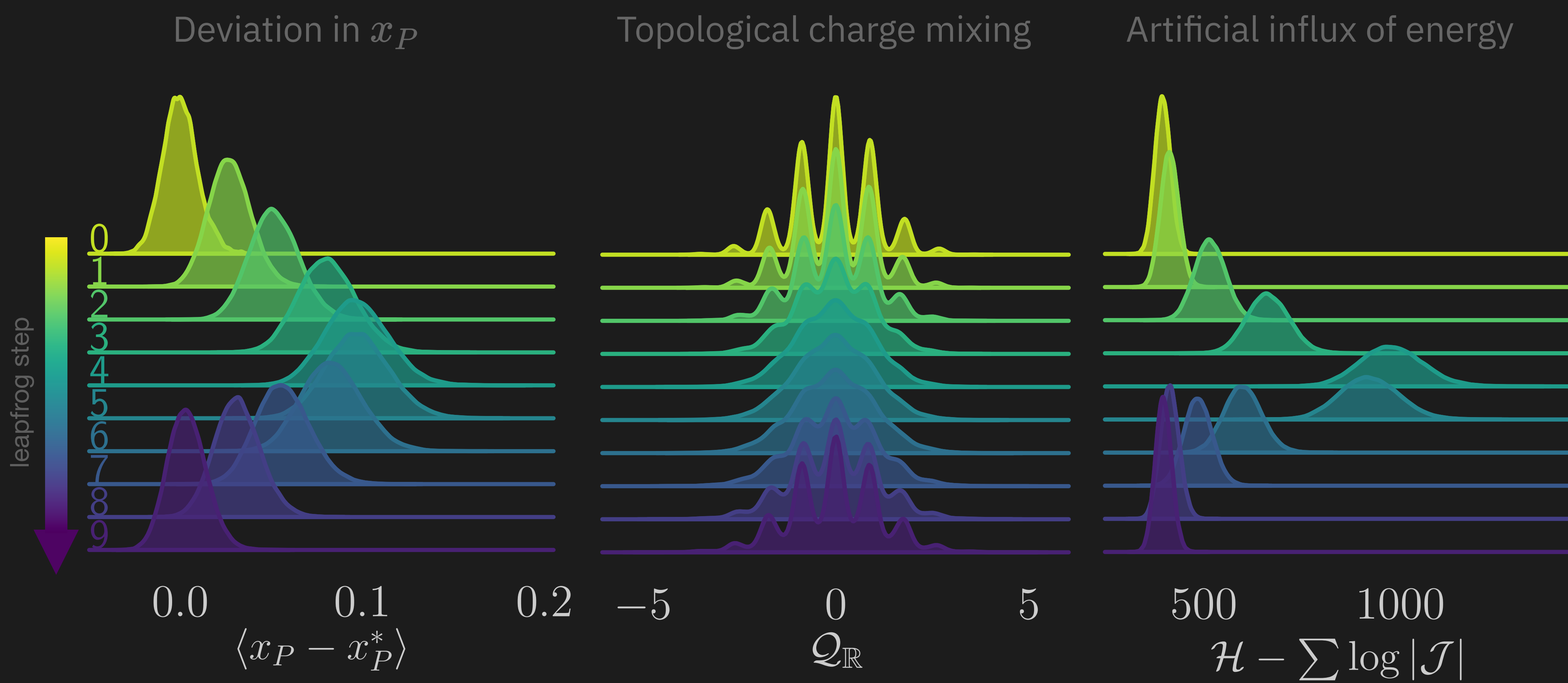
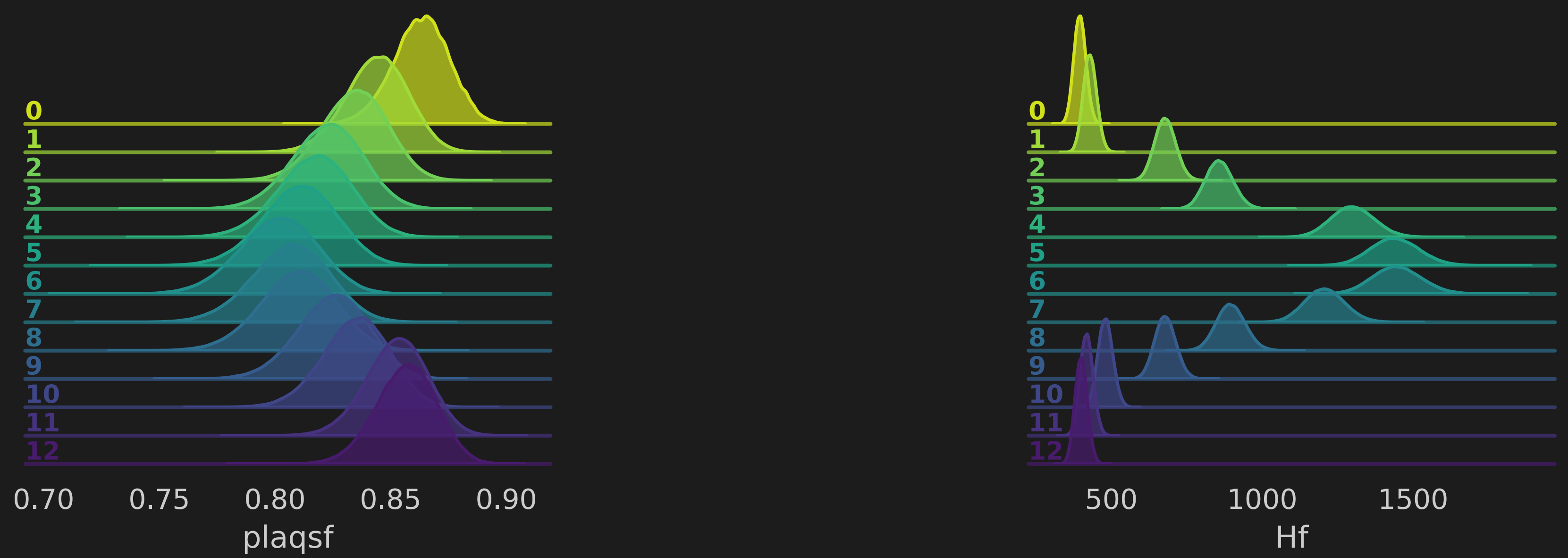


Figure 8: Illustration of how different observables evolve over a single L2HMC trajectory.

Interpretation

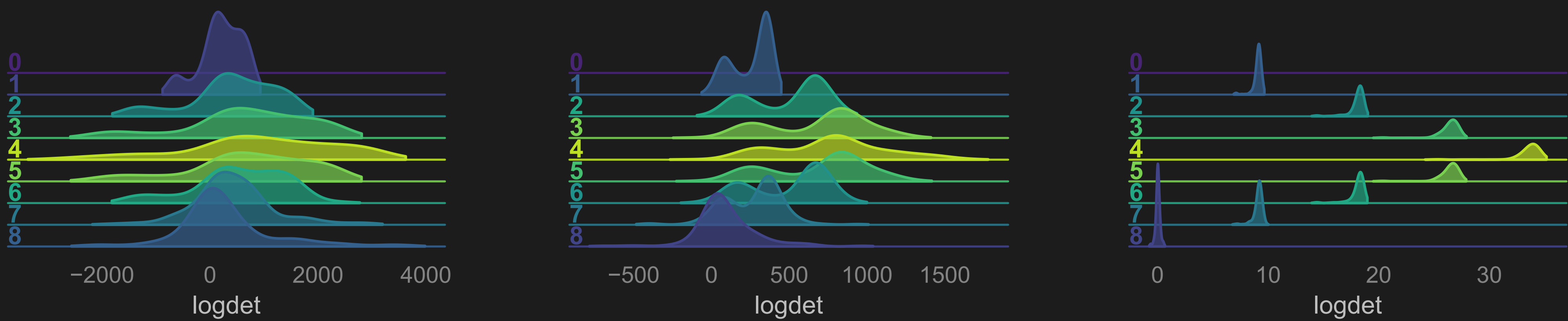


Average plaquette: $\langle x_P \rangle$ *vs LF step*

Average energy: $H - \sum \log |\mathcal{J}|$

Figure 9: The trained model artifically increases the energy towards the middle of the trajectory, allowing the sampler to tunnel between isolated sectors.

4D $SU(3)$ Results



(a) 100 train iters

(b) 500 train iters

(c) 1000 train iters

Figure 10: $\log|\mathcal{J}|$ vs. N_{LF} during training

4D $SU(3)$ Results: $\delta U_{\mu\nu}$

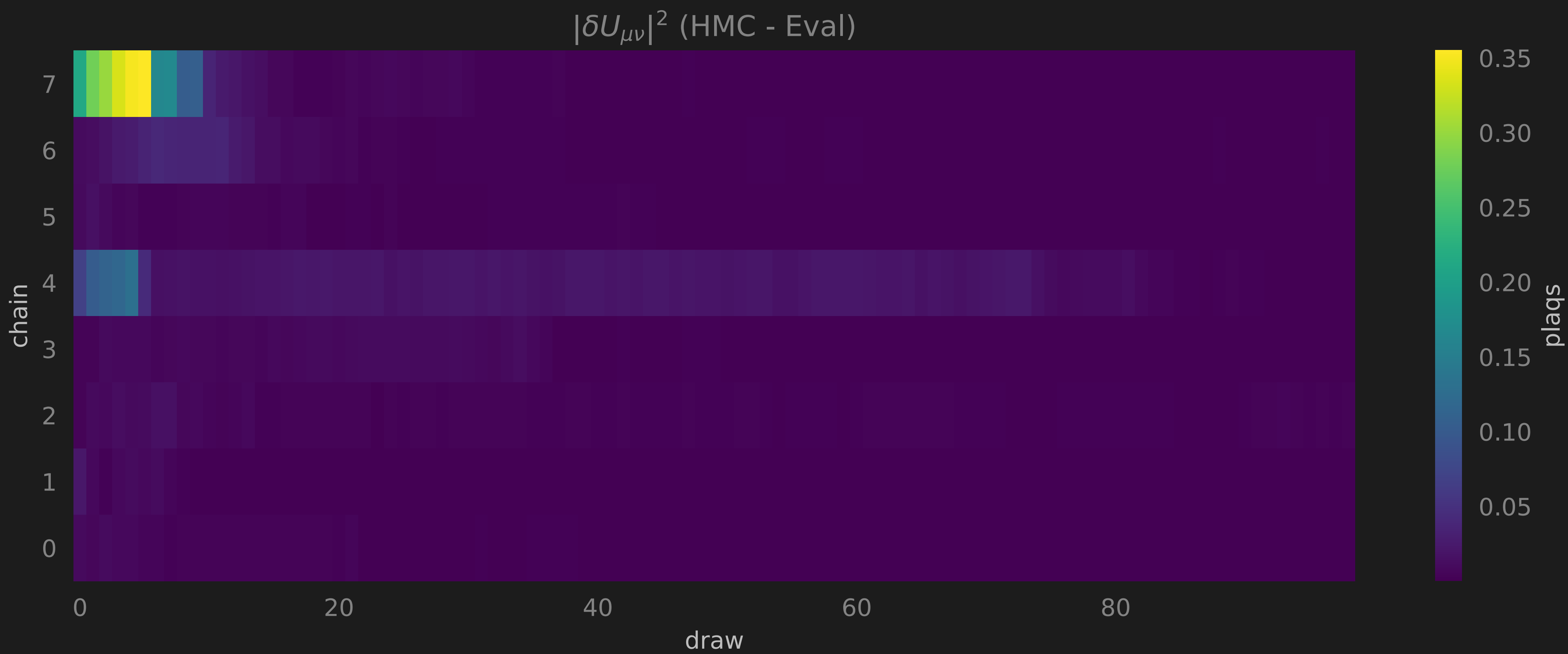


Figure 11: The difference in the average plaquette $|\delta U_{\mu\nu}|^2$ between the trained model and HMC

4D $SU(3)$ Results: $\delta U_{\mu\nu}$

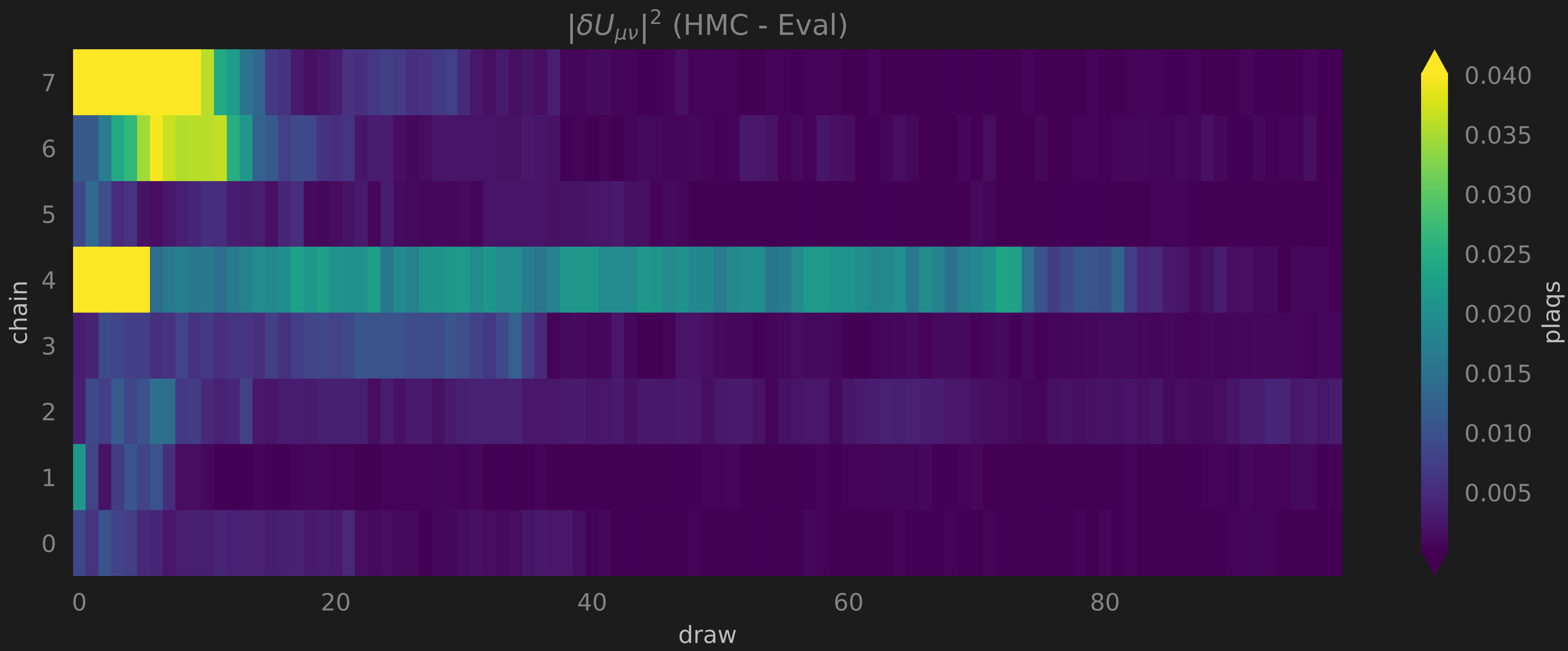


Figure 12: The difference in the average plaquette $|\delta U_{\mu\nu}|^2$ between the trained model and HMC

Next Steps

- Further code development
  - saforem2/l2hmc-qcd
- Continue to use / test different network architectures
  - Gauge equivariant NNs for $U_\mu(x)$ update
- Continue to test different loss functions for training
- Scaling:
  - Lattice volume
  - Network size
  - Batch size
  - # of GPUs

Thank you!

samforeman.me saforem2 @saforem2 foremans@anl.gov

l2hmc-qcd

👏 111 / 21003 | l2hmc-qcd | codefactor A

arXiv 2112.01582 | arXiv 2105.03418

Config Hydra | PyTorch | TensorFlow | Visualize in W&B

Acknowledgements

- **Links**:
  - Link to github
  - reach out!
- **References**:
  - Link to HMC demo
  - Link to slides
    - link to github with slides
  - (Foreman et al. 2022; Foreman, Jin, and Osborn 2022, 2021)
  - (Boyda et al. 2022; Shanahan et al. 2022)

- Huge thank you to:
  - Yannick Meurice
  - Norman Christ
  - Akio Tomiya
  - Luchang Jin
  - Chulwoo Jung
  - Peter Boyle
  - Taku Izubuchi
  - ECP-CSD group
  - **ALCF Staff + Datascience Group**

Acknowledgements

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.

Links + References

- This talk: `saforem2/lattice23`
- Code repo `saforem2/l2hmc-qcd`
- Slides saforem2.github.io/lattice23
- Title Slide Background (worms) animation

References

Boyda, Denis et al. 2022. "Applications of Machine Learning to Lattice Quantum Field Theory." In *Snowmass 2021*. https://arxiv.org/abs/2202.05838.

Foreman, Sam, Taku Izubuchi, Luchang Jin, Xiao-Yong Jin, James C. Osborn, and Akio Tomiya. 2022. "HMC with Normalizing Flows." *PoS* LATTICE2021: 073. https://doi.org/10.22323/1.396.0073.

Foreman, Sam, Xiao-Yong Jin, and James C. Osborn. 2021. "Deep Learning Hamiltonian Monte Carlo." In *9th International Conference on Learning Representations*. https://arxiv.org/abs/2105.03418.

———. 2022. "LeapfrogLayers: A Trainable Framework for Effective Topological Sampling." *PoS* LATTICE2021 (May): 508. https://doi.org/10.22323/1.396.0508.

Shanahan, Phiala et al. 2022. "Snowmass 2021 Computational Frontier CompF03 Topical Group Report: Machine Learning," September. https://arxiv.org/abs/2209.07559.

Extras

Integrated Autocorrelation Time

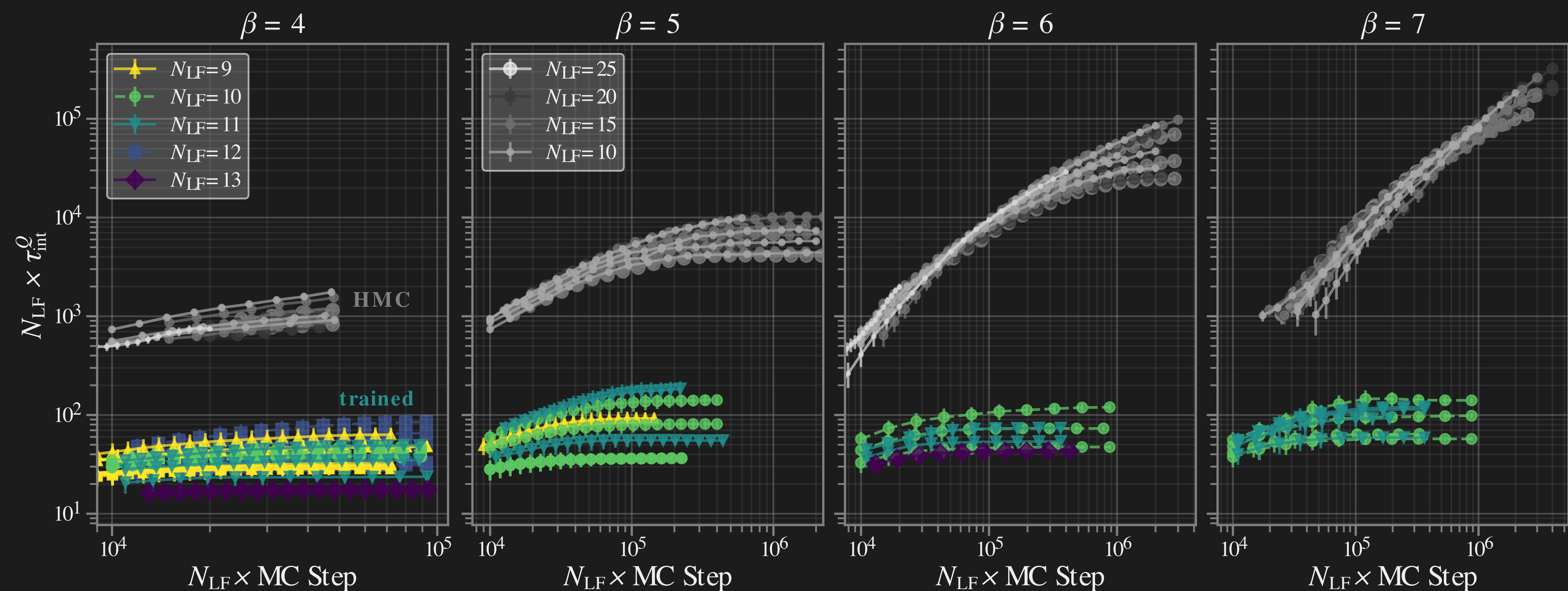
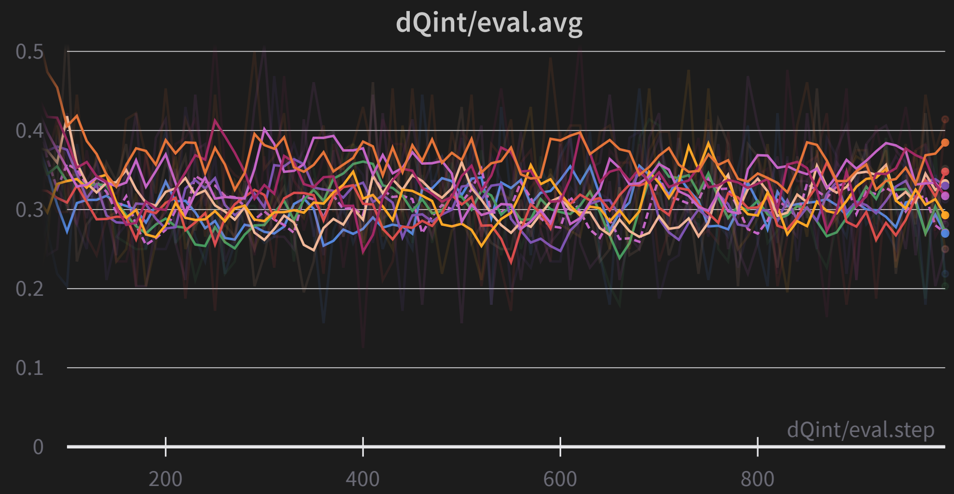
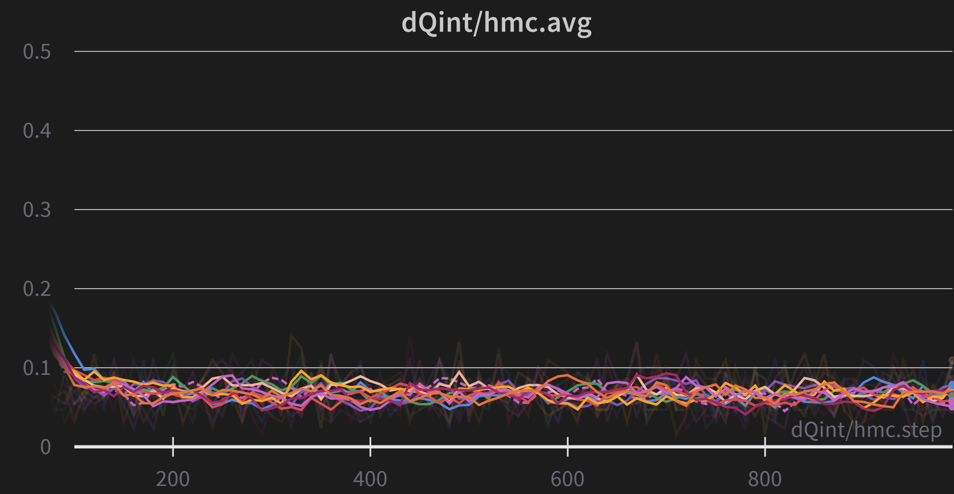


Figure 13: Plot of the integrated autocorrelation time for both the trained model (colored) and HMC (greyscale).

Comparison



(a) Trained model



(b) Generic HMC

Figure 14: Comparison of $\langle \delta Q \rangle = \frac{1}{N}\sum_{i=k}^{N} \delta Q_i$ for the trained model Figure 14 (a) vs. HMC Figure 14 (b)

Plaquette analysis: x_P

Deviation from $V \to \infty$ limit, x_P^*

Average $\langle x_P \rangle$, with x_P^* (dotted-lines)

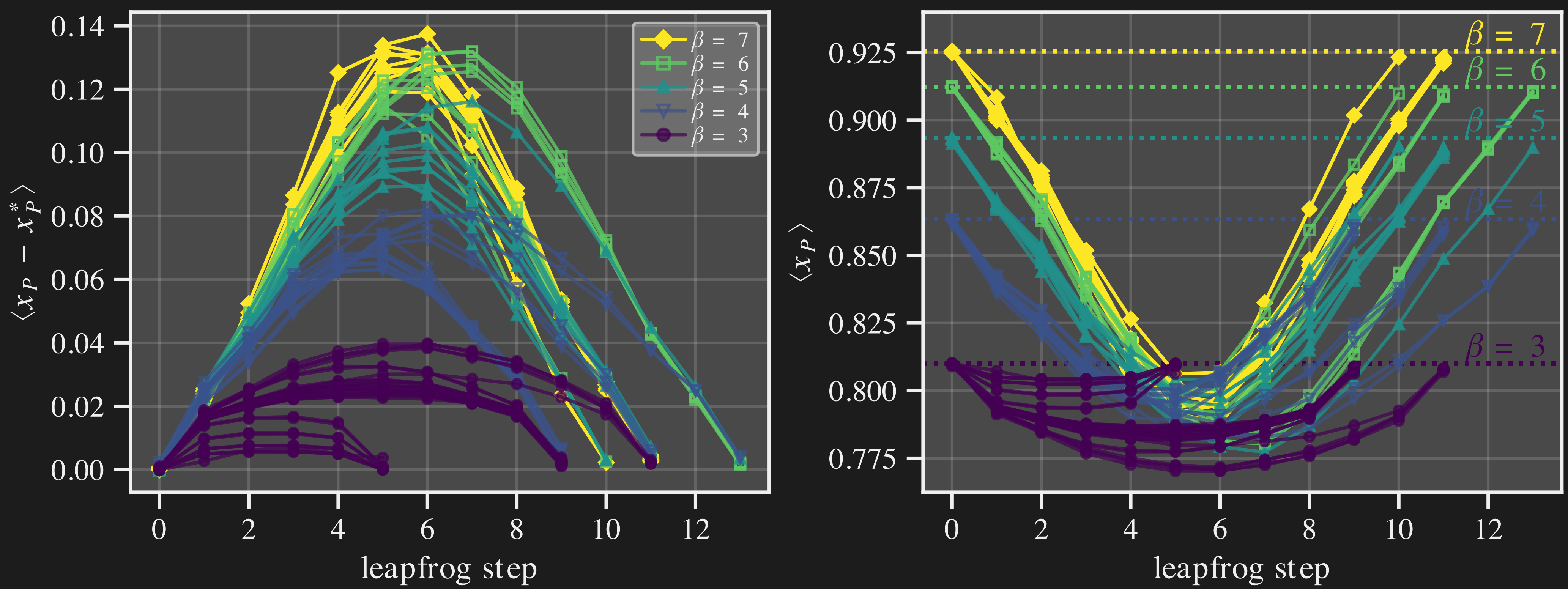


Figure 15: Plot showing how **average plaquette**, $\langle x_P \rangle$ varies over a single trajectory for models trained at different β, with varying trajectory lengths N_{LF}

Loss Function

- Want to maximize the *expected* squared charge difference[1]:

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

- Where:
  - δQ is the *tunneling rate*:

$$\delta Q(\xi^*, \xi) = |Q^* - Q|$$

  - $A(\xi^*|\xi)$ is the probability[2] of accepting the proposal ξ^*:

$$A(\xi^*|\xi) = \min\left(1, \frac{p(\xi^*)}{p(\xi)}\left|\frac{\partial \xi^*}{\partial \xi^T}\right|\right)$$

1. Where ξ^* is the *proposed* configuration (prior to Accept / Reject)
2. And $\left|\frac{\partial \xi^*}{\partial \xi^T}\right|$ is the Jacobian of the transformation from $\xi \to \xi^*$

v-Update[1]

- forward $(d = +)$:

$$\Gamma^{+} : (x, v) \rightarrow v' \coloneqq v \cdot e^{\frac{\varepsilon}{2} s_v} - \frac{\varepsilon}{2}\left[F \cdot e^{\varepsilon q_v} + t_v\right]$$

- backward $(d = -)$:

$$\Gamma^{-} : (x, v) \rightarrow v' \coloneqq e^{-\frac{\varepsilon}{2} s_v}\left\{v + \frac{\varepsilon}{2}\left[F \cdot e^{\varepsilon q_v} + t_v\right]\right\}$$

1. Note that $\left(\Gamma^{+}\right)^{-1} = \Gamma^{-}$, i.e. $\Gamma^{+}\left[\Gamma^{-}(x, v)\right] = \Gamma^{-}\left[\Gamma^{+}(x, v)\right] = (x, v)$

x-Update

- forward $(d = +)$:

$$\Lambda^{+}(x, v) = x \cdot e^{\frac{\varepsilon}{2} s_x} - \frac{\varepsilon}{2}\left[v \cdot e^{\varepsilon q_x} + t_x\right]$$

- backward $(d = -)$:

$$\Lambda^{-}(x, v) = e^{-\frac{\varepsilon}{2} s_x}\left\{x + \frac{\varepsilon}{2}\left[v \cdot e^{\varepsilon q_x} + t_x\right]\right\}$$

Lattice Gauge Theory (2D $U(1)$)

Link Variables

$$U_\mu(n) = e^{i x_\mu(n)} \in \mathbb{C}, \quad \text{where}$$

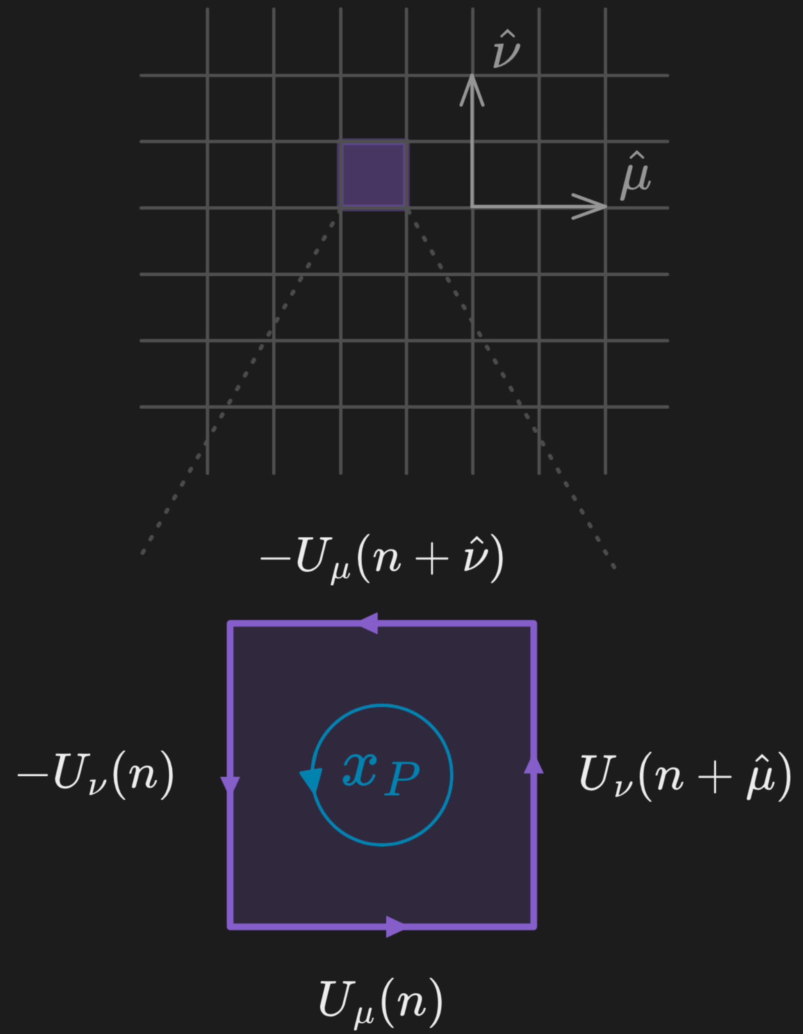
$$x_\mu(n) \in [-\pi, \pi)$$

Wilson Action

$$S_\beta(x) = \beta \sum_P \cos x_P,$$

$$x_P = [x_\mu(n) + x_\nu(n+\hat{\mu}) - x_\mu(n+\hat{\nu}) - x_\nu(n)]$$

Note: x_P is the product of links around 1×1 square, called a "plaquette"



2D Lattice

Figure 16: Jupyter Notebook

Annealing Schedule

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

$$\{\gamma_t\}_{t=0}^{N} = \{\gamma_0, \gamma_1, \ldots, \gamma_{N-1}, \gamma_N\}$$

  where $\gamma_0 < \gamma_1 < \cdots < \gamma_N \equiv 1$, and $\left|\gamma_{t+1} - \gamma_t\right| \ll 1$
- **Note**:
  - for $\left|\gamma_t\right| < 1$, this rescaling helps to reduce the height of the energy barriers $\Longrightarrow$
  - easier for our sampler to explore previously inaccessible regions of the phase space

Networks 2D $U(1)$

- Stack gauge links as `shape`(U_μ)`=[Nb, 2, Nt, Nx]` $\in \mathbb{C}$

$$x_\mu(n) := [\cos(x), \sin(x)]$$

  with `shape`(x_μ)`= [Nb, 2, Nt, Nx, 2]` $\in \mathbb{R}$

- x-Network:
  - $\psi_\theta : (x, v) \longrightarrow (s_x,\, t_x,\, q_x)$
- v-Network:
  - $\varphi_\theta : (x, v) \longrightarrow (s_v,\, t_v,\, q_v)$

Networks 2D $U(1)$

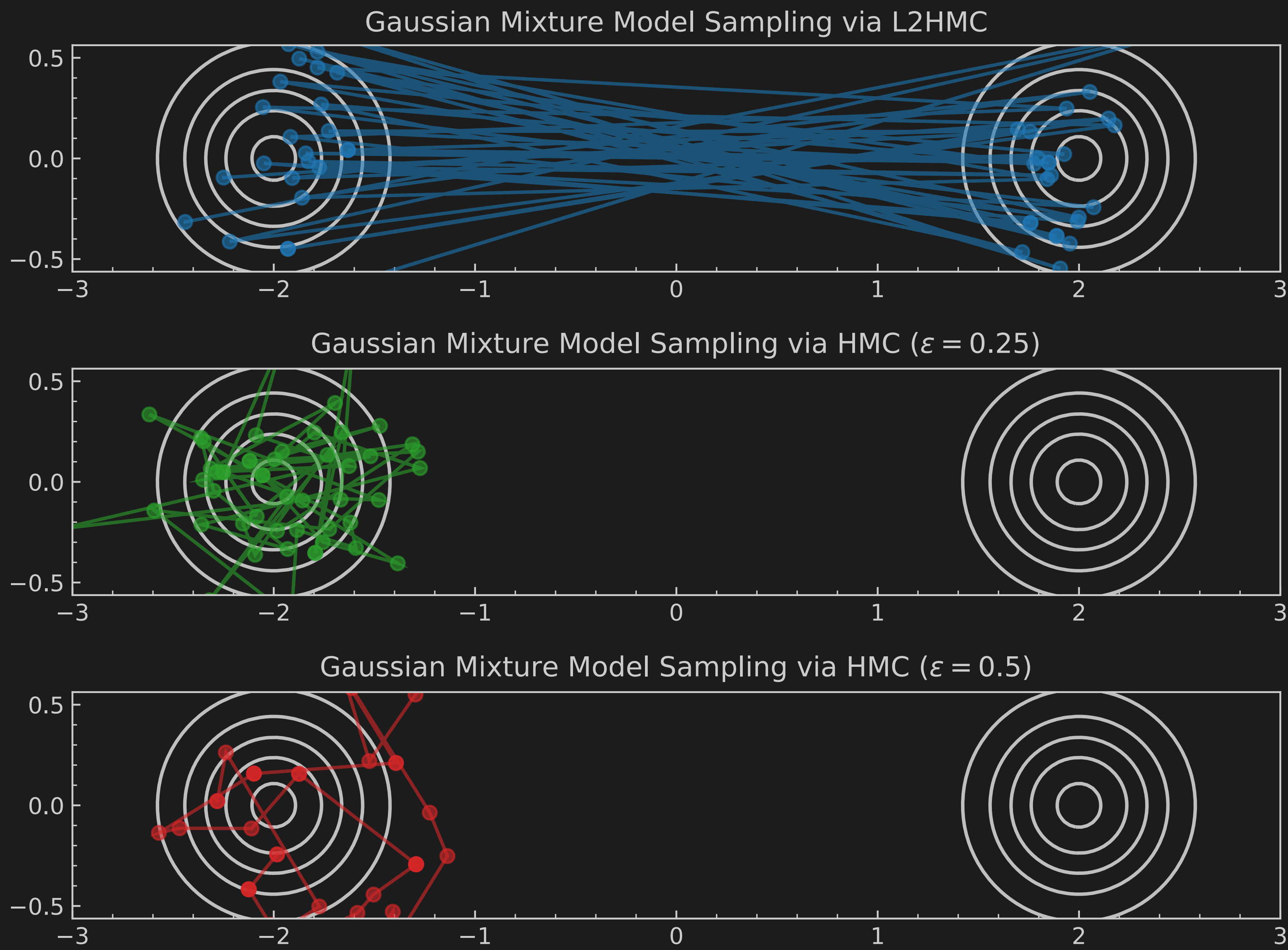
- Stack gauge links as `shape`(U_μ)`=[Nb, 2, Nt, Nx]` $\in \mathbb{C}$

$$x_\mu(n) \coloneqq [\cos(x), \sin(x)]$$

  with `shape`(x_μ)`= [Nb, 2, Nt, Nx, 2]` $\in \mathbb{R}$

- x-Network:
  - $\psi_\theta : (x, v) \longrightarrow (s_x,\, t_x,\, q_x)$
- v-Network:
  - $\varphi_\theta : (x, v) \longrightarrow (s_v,\, t_v,\, q_v)$ $\longleftarrow$ lets look at this

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



Physical Quantities

- To estimate physical quantities, we:
  - calculate physical observables at **increasing** spatial resolution
  - perform extrapolation to continuum limit

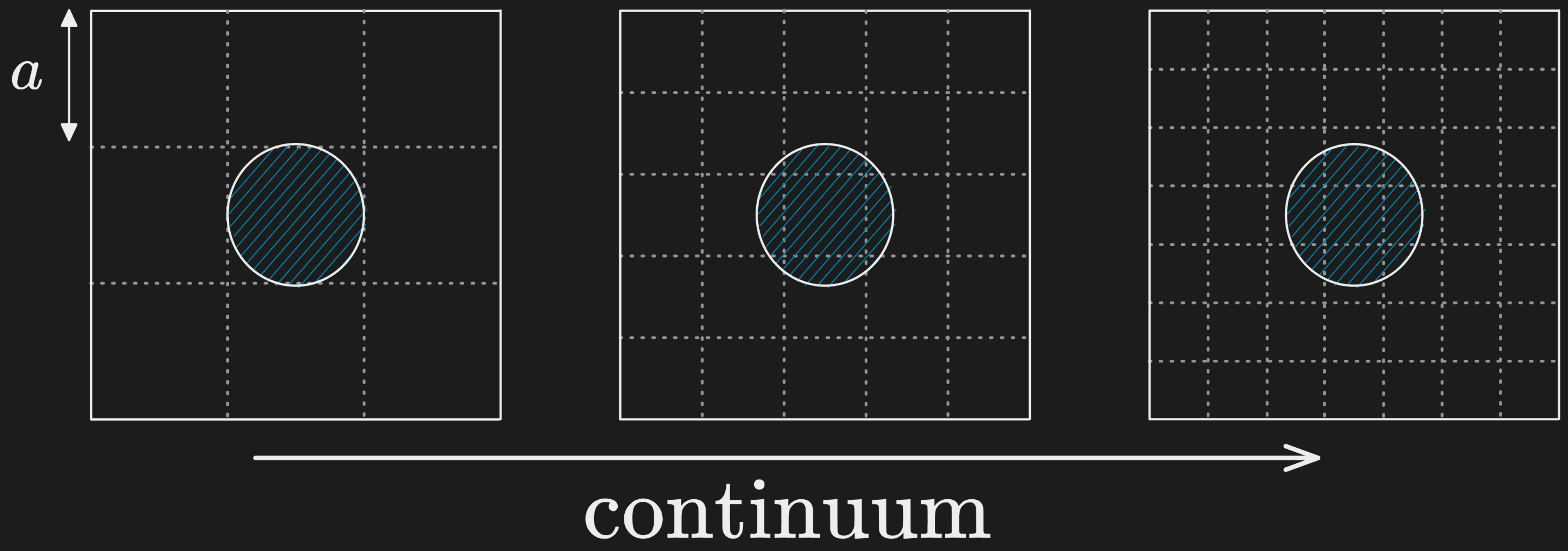


Figure 17: Increasing the physical resolution ($a \to 0$) allows us to make predictions about numerical values of physical quantities in the continuum limit.