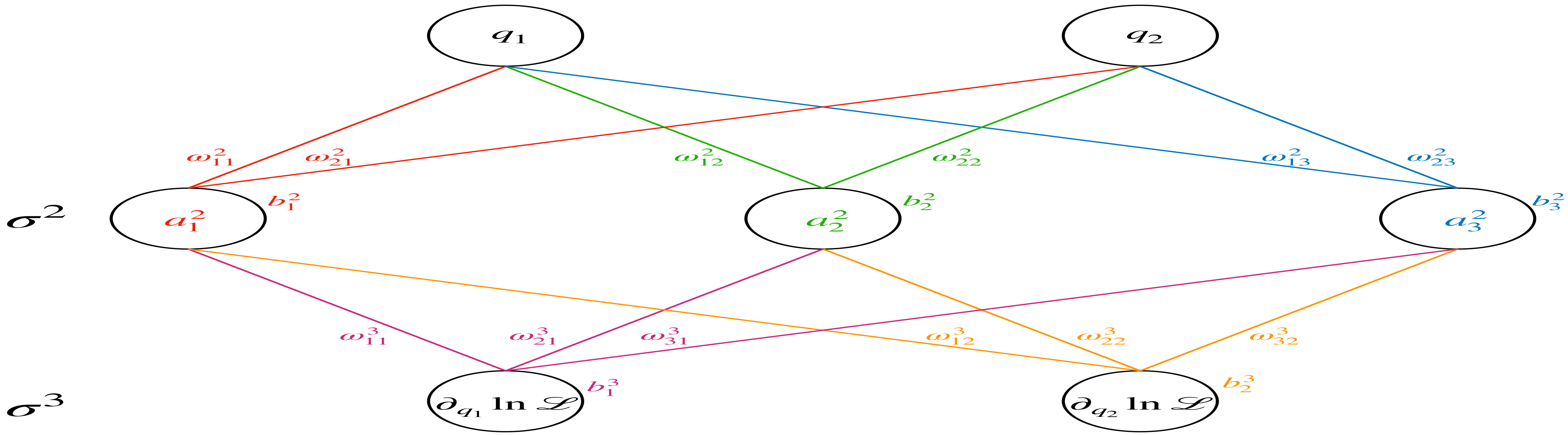
DeepHMC:a deep neural network acclerated Hamiltonian Monte Carlo algorithm for BNS parameter estimation



Jules Perret and **Ed Porter** (APC/CNRS)

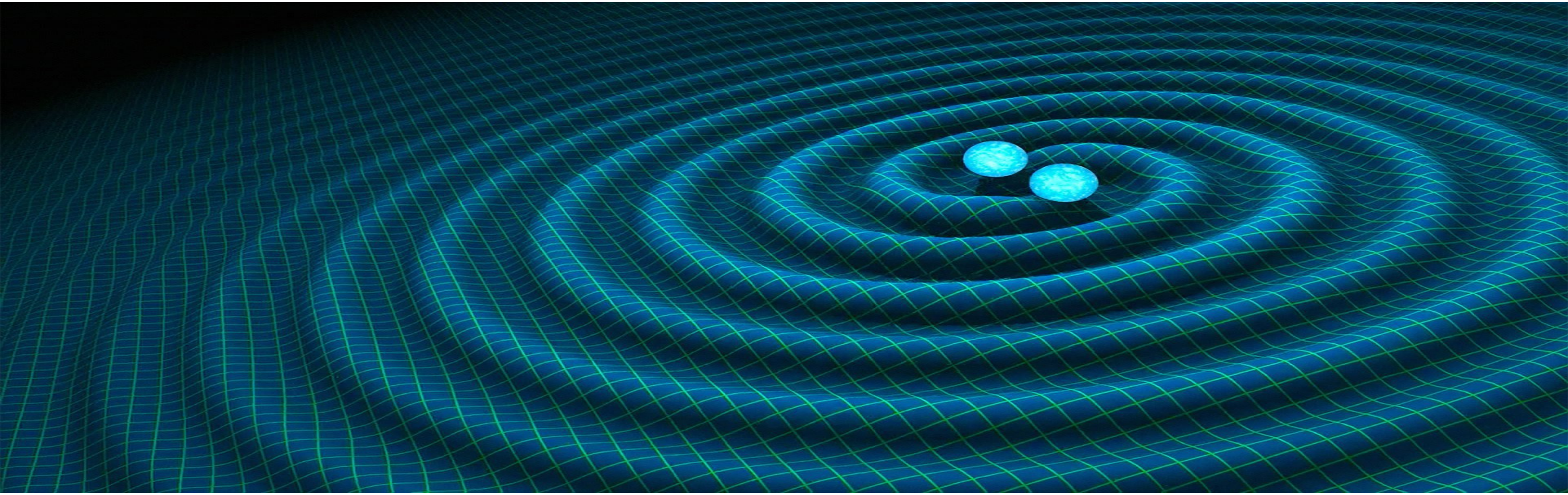
Paris workshop on Bayesian deep learning
20-23 May 2025

arXiv:2505.02589

Outline

- *Bayesian inference for GWs*
- *Motivation for using a DNN*
- *Constructing the DNN*
- *Application to GW170817/GW190425*

Bayesian Inference for GWs



Bayesian Inference for GWs

- *BNS inference currently takes many hours to days to complete using MCMC/ Nested sampling techniques*

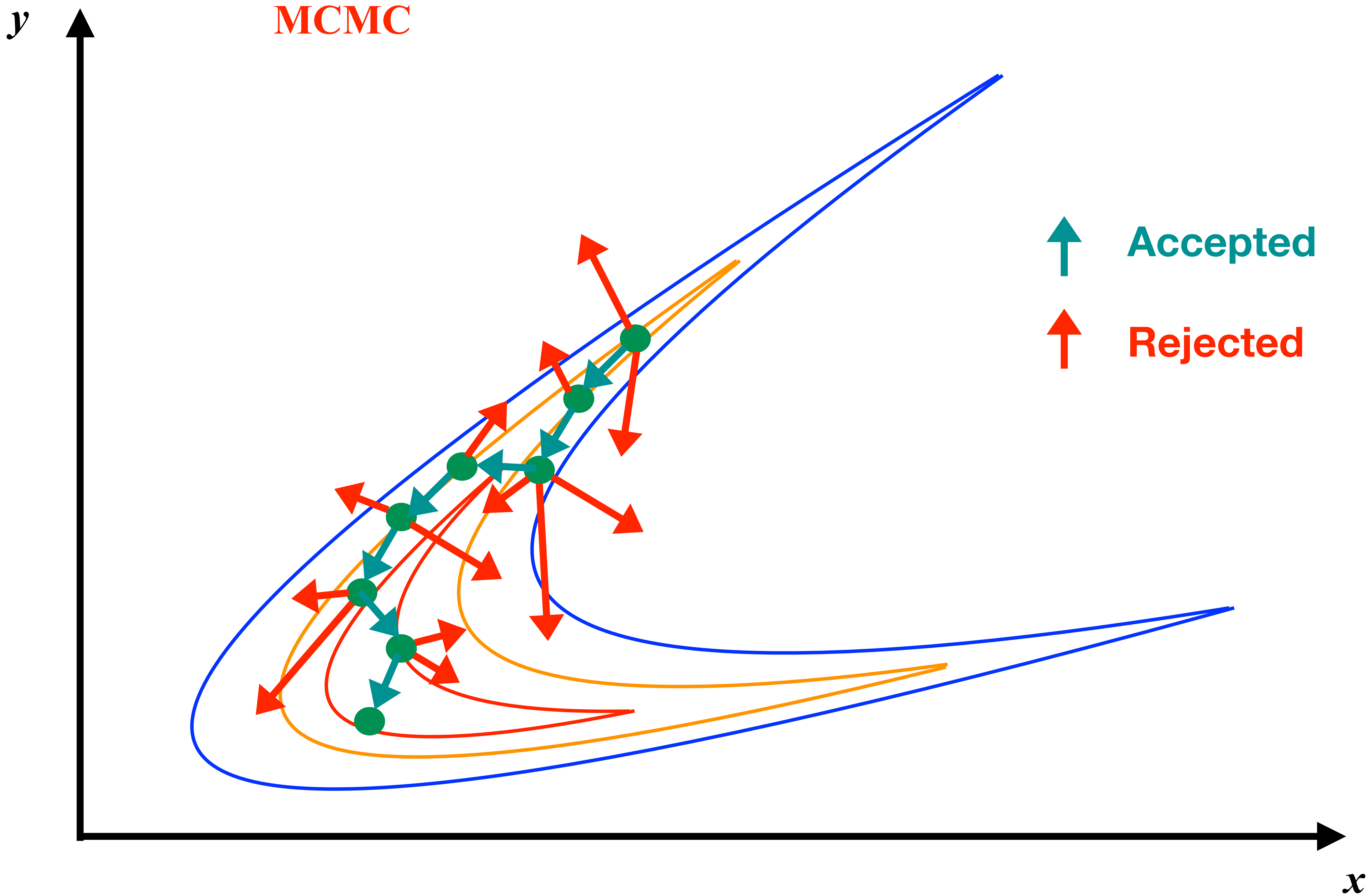
- *As the low frequency performance of the detectors improve, waveform durations will get longer, so…*

- *…we want to be able to do the inference as fast, and as efficiently, as possible*

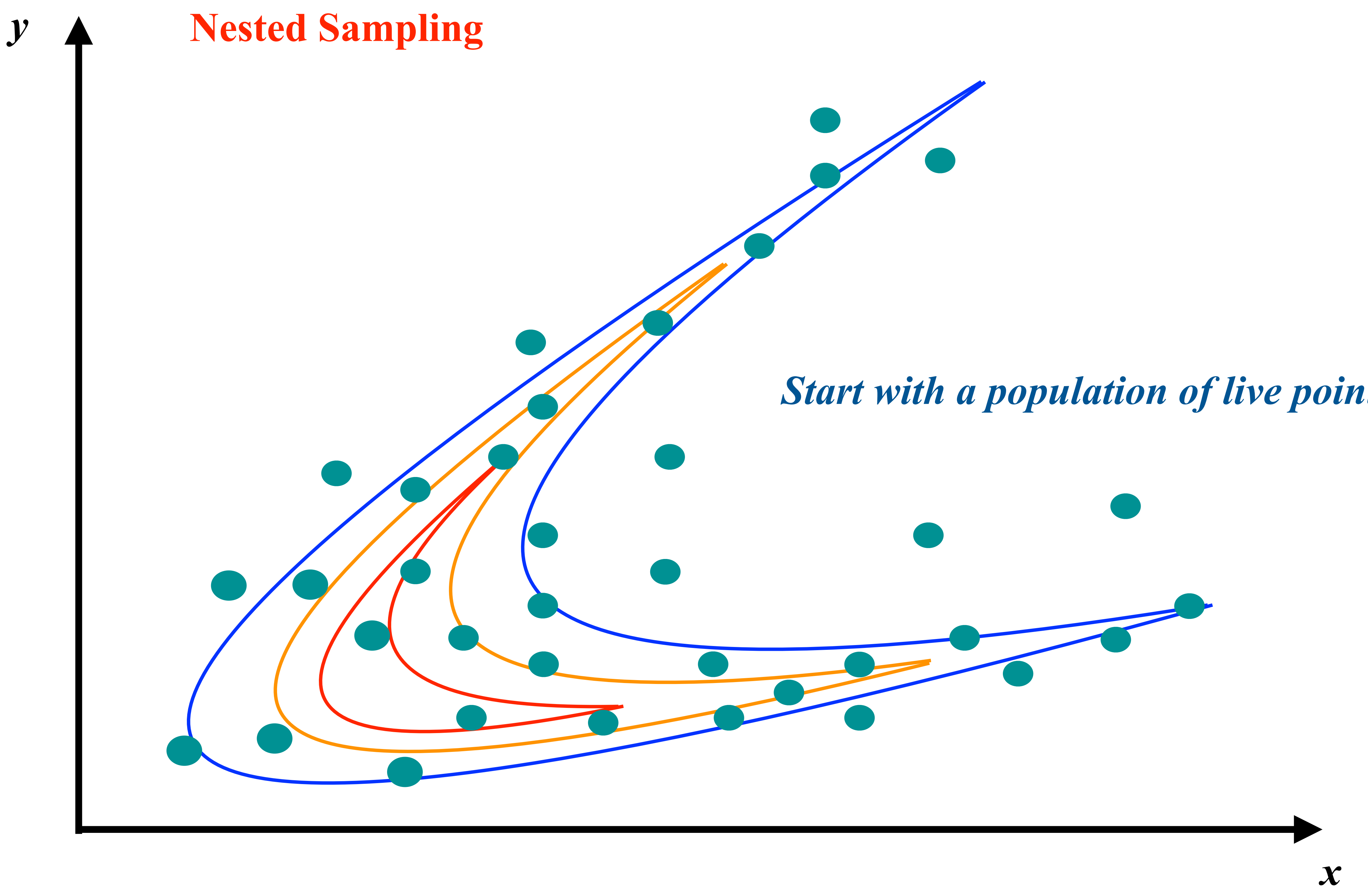
Bayesian Inference for GWs

- *Standard Bayesian inference samplers (e.g. MCMC, nested sampling) belong to a family of random walk samplers*
    - *require specifically designed proposal distributions to improve efficiency*
    - *known to have slow convergence properties, especially as D increases*
    - *GW parameter estimation can take hours to days to run.*
- *Markovian process - future states depend only on the current state and not on any past states, i.e. no history*
- *Pros*
    - *(relatively) easy to get going*
    - *will (eventually) converge*
- *Cons*
    - *High autocorrelations*
    - *Convergence is slow*
    - *Difficult to apply off-the-shelf versions to specific problems*
    - *Prone to getting stuck in local minima*

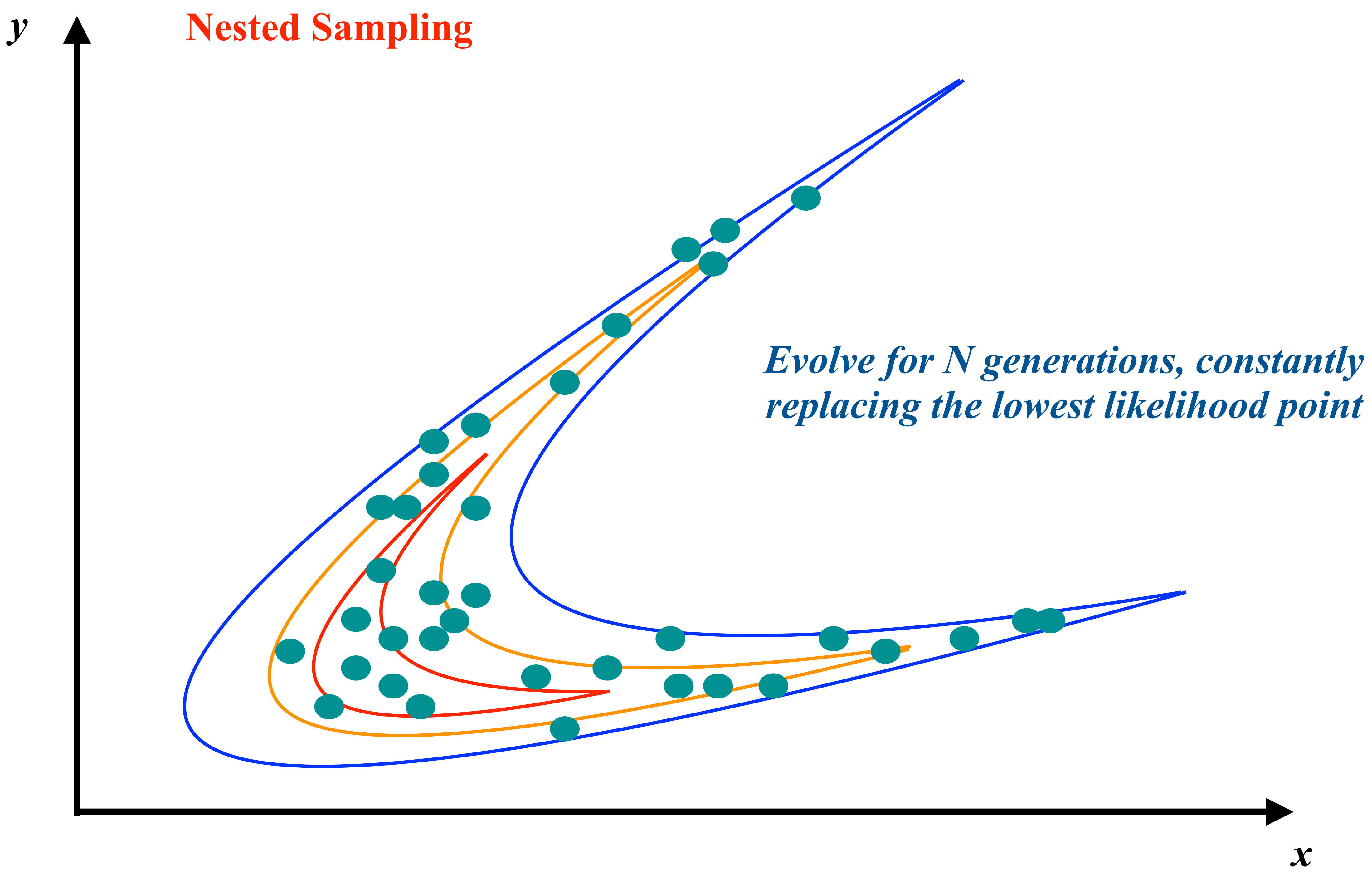
Bayesian Inference for GWs



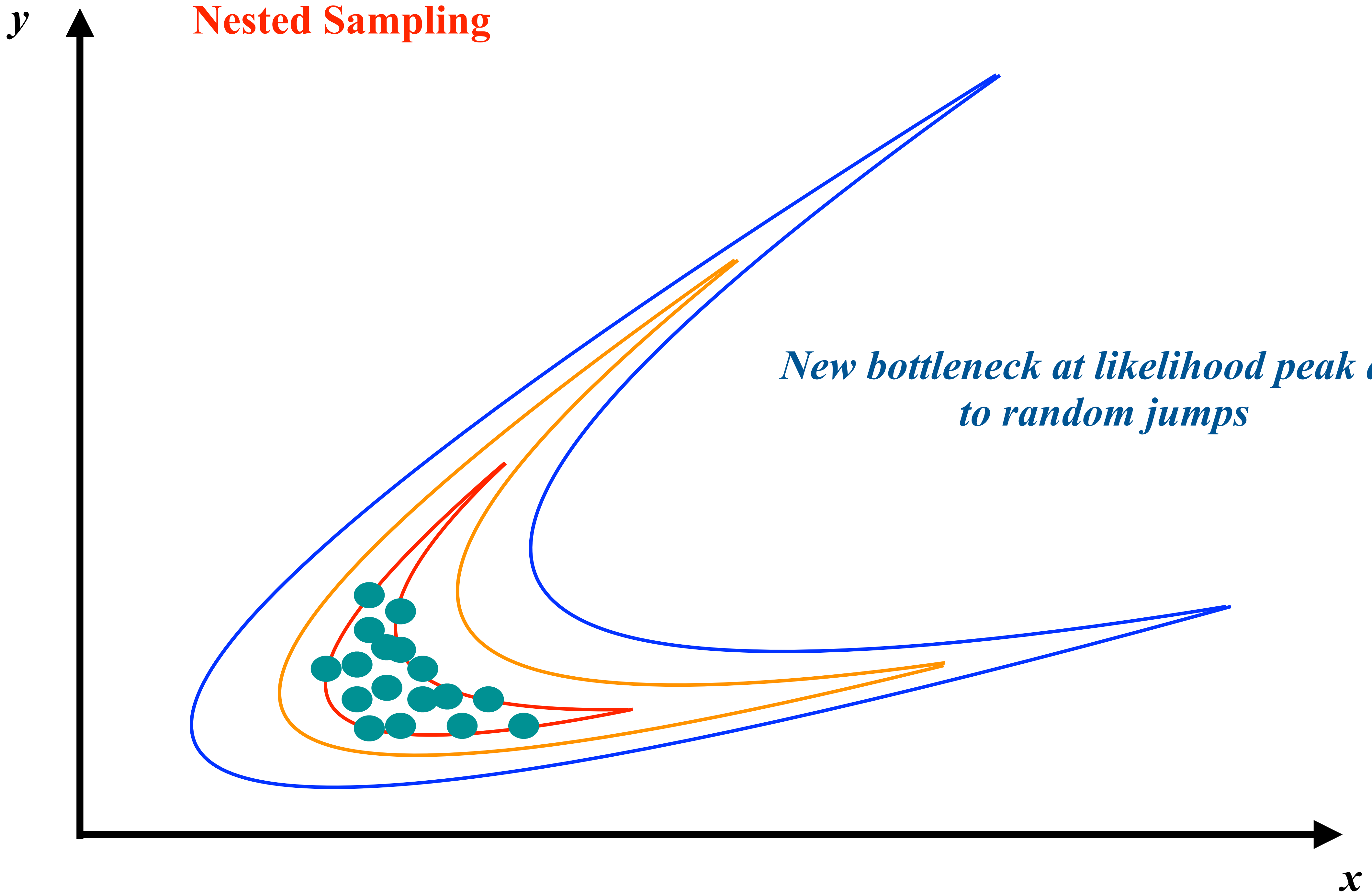
Bayesian Inference for GWs



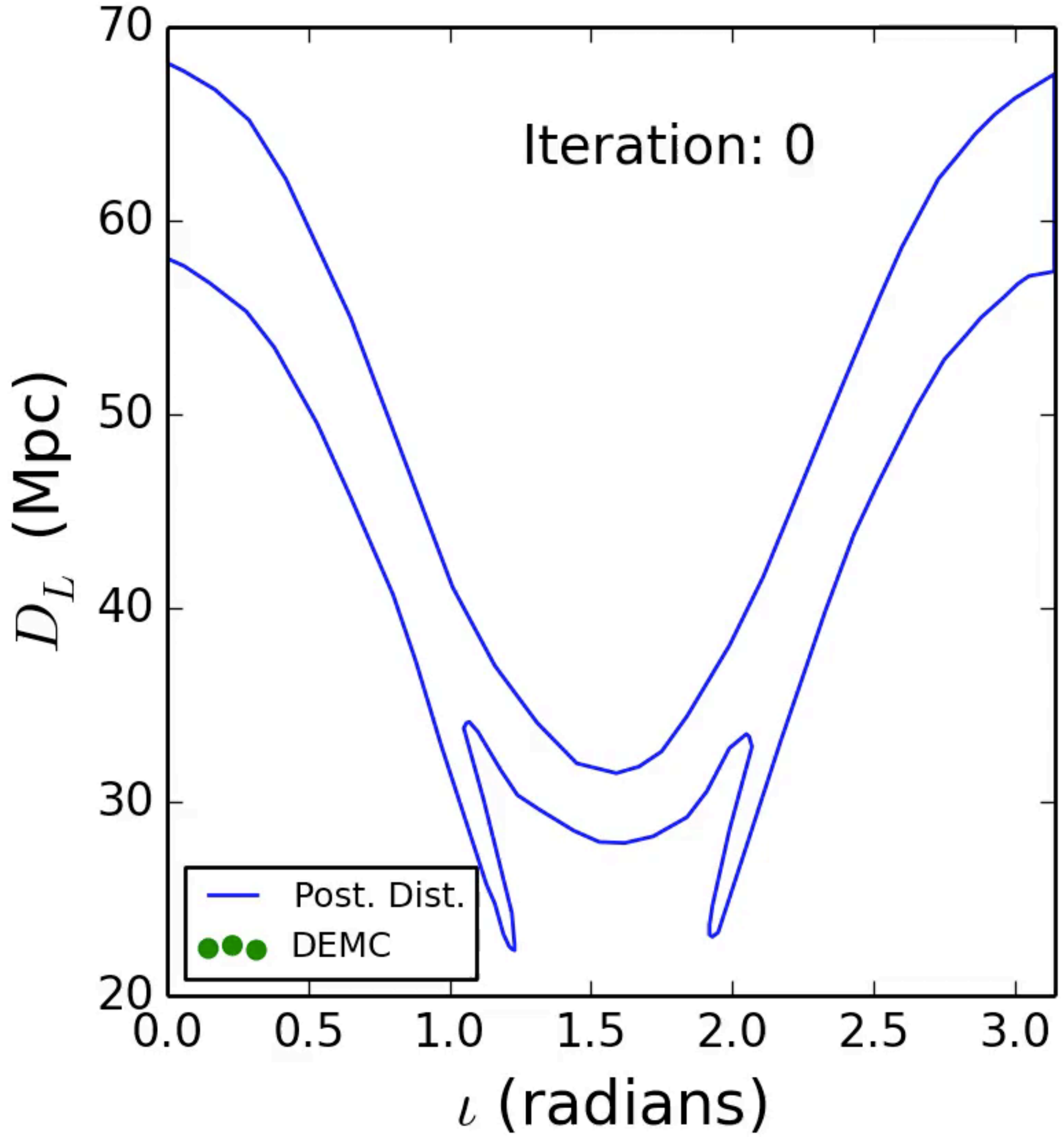
Bayesian Inference for GWs

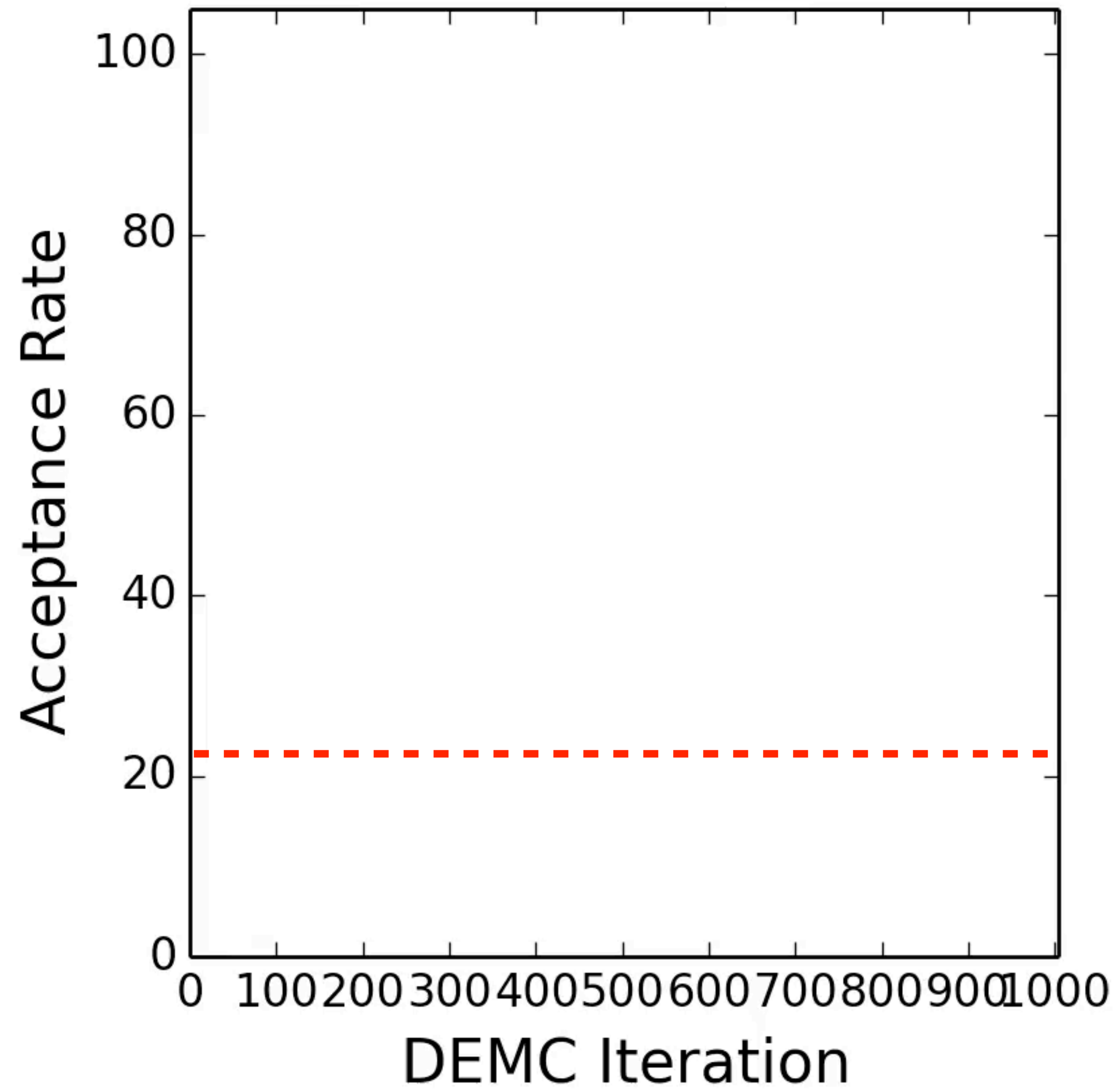


Bayesian Inference for GWs



Bayesian Inference for GWs





Hamiltonian Monte Carlo

- *Instead of a maximisation problem, make it a "gravitational problem"*

- *Equate GW template parameters to state space variables, i.e.* $q^\mu = \lambda^\mu$

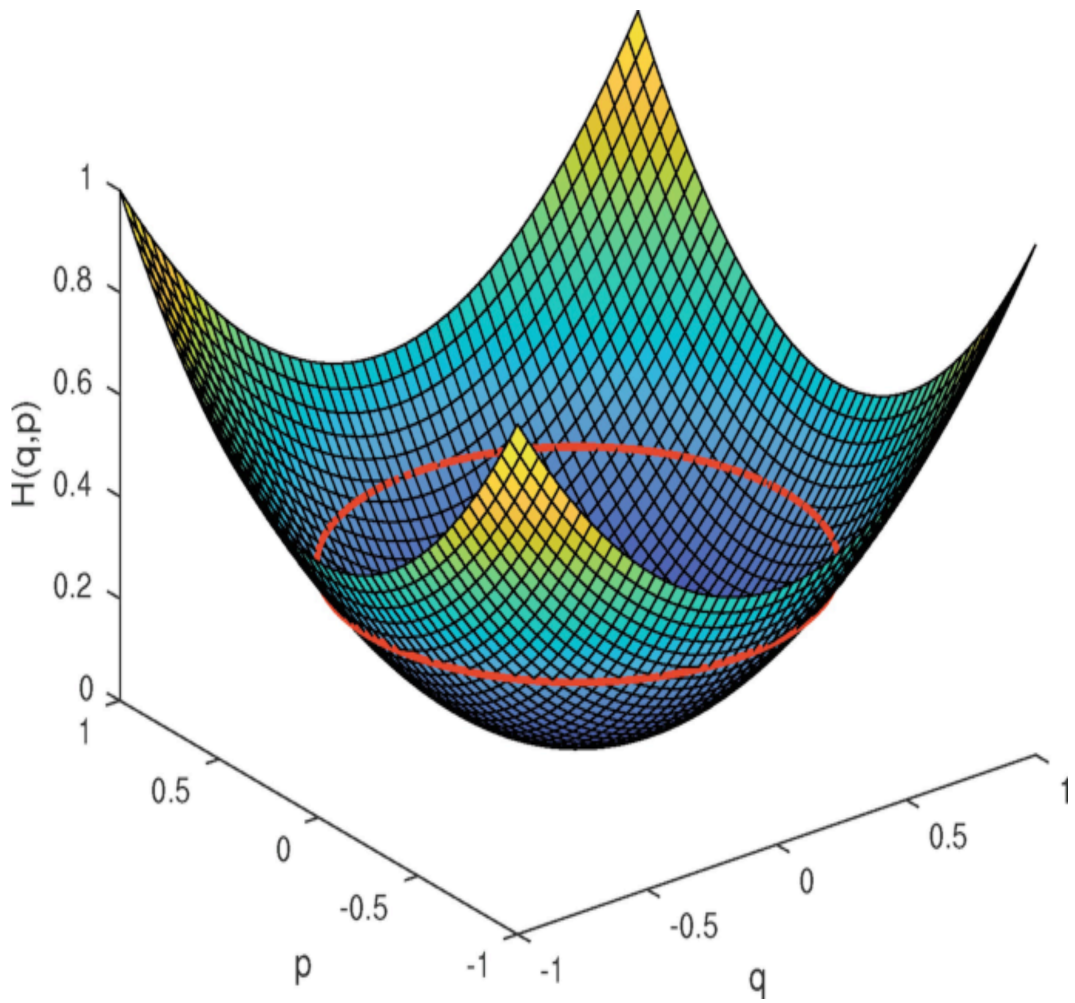
- *Construct a "gravitational" potential energy:*

$$\mathcal{U}(q^\mu) = -\ln[\pi(q^\mu)\mathcal{L}(q^\mu)]$$

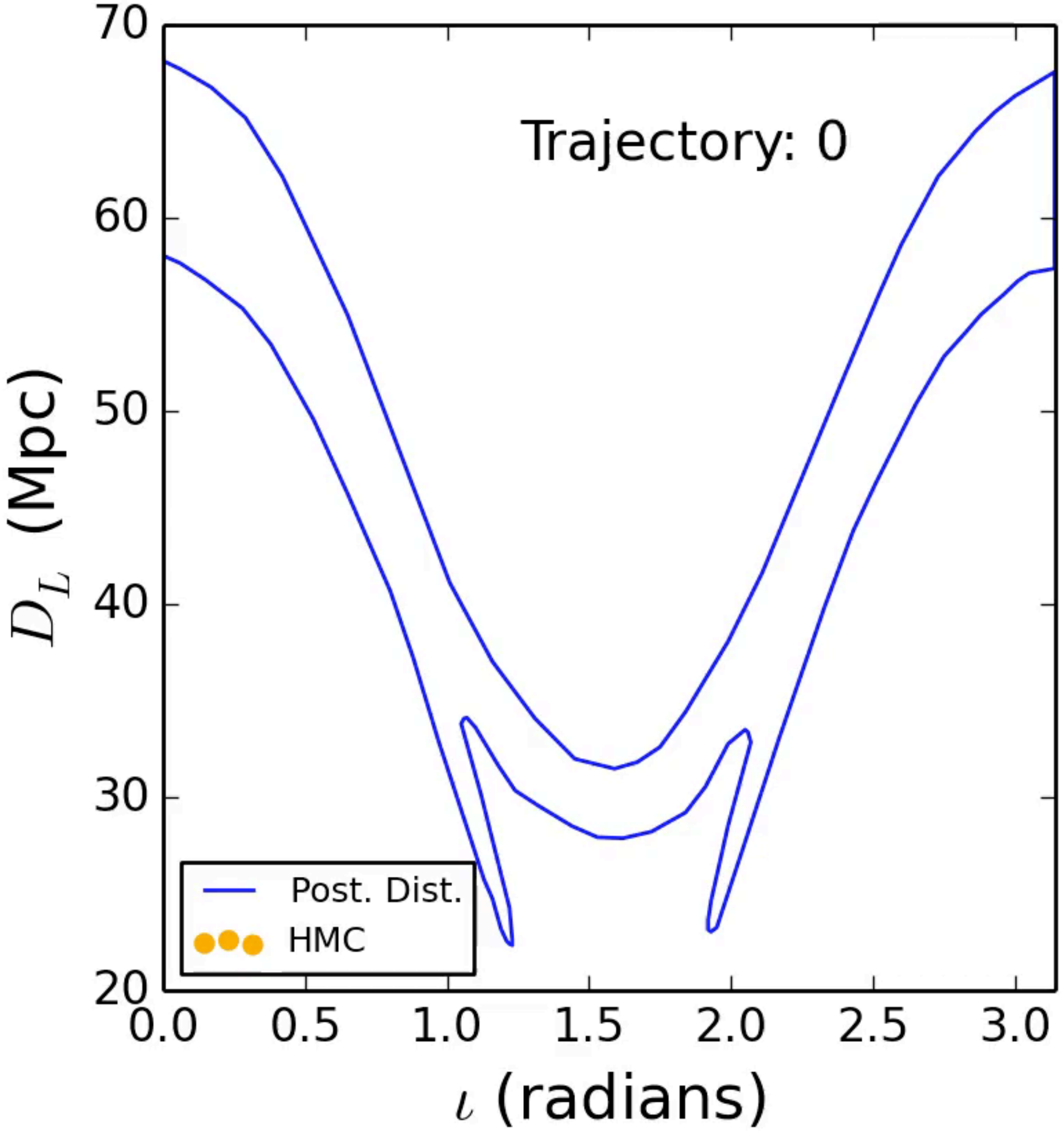
- *Define canonical momenta and Kinetic energy:*

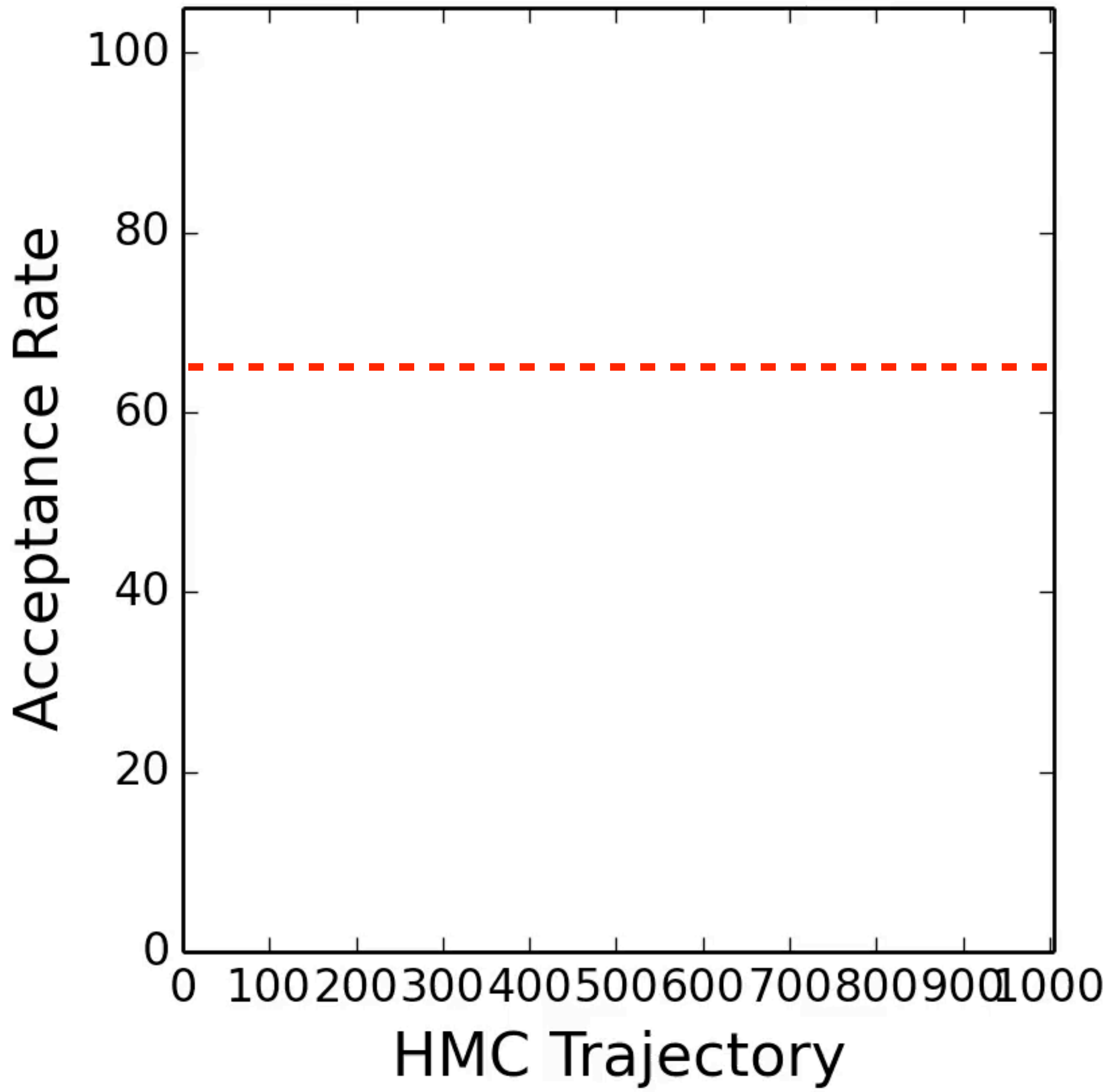
$$\mathcal{K}(p^\mu) = \frac{1}{2}M^{-1}_{\mu\nu}p^\mu p^\nu$$

- *and a Hamiltonian:* $\mathcal{H}(q^\mu, p^\mu) = \mathcal{U}(q^\mu) + \mathcal{K}(p^\mu)$



Hamiltonian Monte Carlo



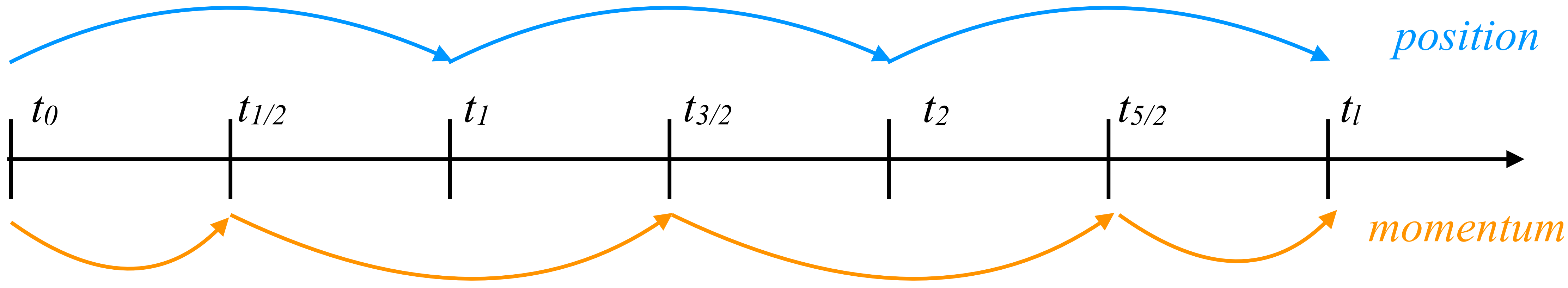


Motivation for using DNNs

Solving Hamilton's equations

- *GWs are analogous to 1D sound waves*
- *Use Wiener or matched filtering to search for weak signals buried in instrumental noise*
- *The method requires the generation of a waveform model or template that is used to cross-correlate the data*
- *While the analysis is conducted in the frequency domain, the templates can be generated in the time or frequency domains*
- *The templates are either semi-analytic (frequency) or require solving coupled differential equations (time)*
- *Each template has a digital representation, i.e.* $h = h(\Delta t)$
- *The analysis is limited by the Nyquist sampling theorem that imposes* $\Delta t = 1/(2f_{Nyq}) = 1/(2f_{max})$

Solving Hamilton's equations

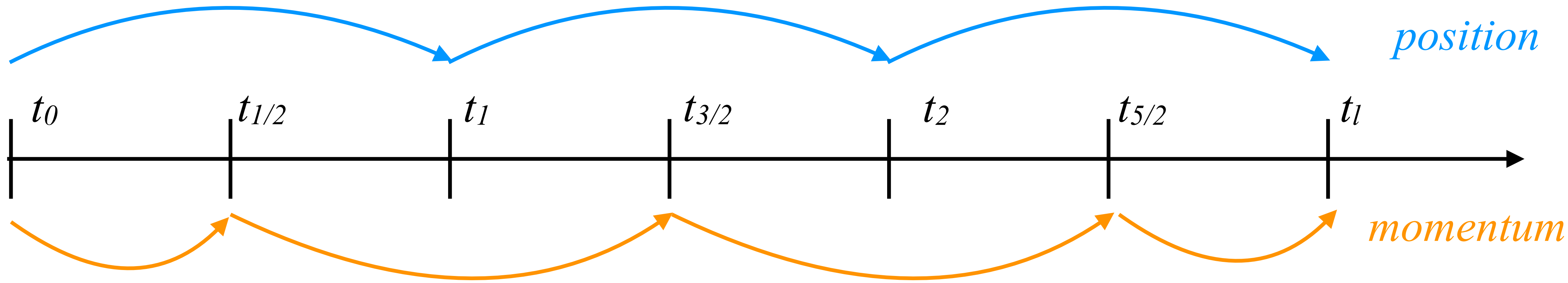


$$\tilde{p}^{\mu}(\tau + \epsilon^{\mu}/2) = \tilde{p}^{\mu}(\tau) + \frac{\epsilon^{\mu}}{2}\frac{\partial U}{\partial q^{\mu}}\bigg|_{q^{\mu}(\tau)}$$

$$q^{\mu}(\tau + \epsilon^{\mu}) = q^{\mu}(\tau) + \epsilon^{\mu}\tilde{p}^{\mu}(\tau + \epsilon^{\mu}/2)$$

$$\tilde{p}^{\mu}(\tau + \epsilon^{\mu}) = \tilde{p}^{\mu}(\tau + \epsilon^{\mu}/2) + \frac{\epsilon^{\mu}}{2}\frac{\partial U}{\partial q^{\mu}}\bigg|_{q^{\mu}(\tau+\epsilon^{\mu})}$$

Solving Hamilton's equations



$$\tilde{p}^{\mu}(\tau + \epsilon^{\mu}/2) = \tilde{p}^{\mu}(\tau) + \frac{\epsilon^{\mu}}{2} \boxed{\left.\frac{\partial U}{\partial q^{\mu}}\right|_{q^{\mu}(\tau)}}$$

$$q^{\mu}(\tau + \epsilon^{\mu}) = q^{\mu}(\tau) + \epsilon^{\mu}\tilde{p}^{\mu}(\tau + \epsilon^{\mu}/2)$$

$$\tilde{p}^{\mu}(\tau + \epsilon^{\mu}) = \tilde{p}^{\mu}(\tau + \epsilon^{\mu}/2) + \frac{\epsilon^{\mu}}{2} \boxed{\left.\frac{\partial U}{\partial q^{\mu}}\right|_{q^{\mu}(\tau+\epsilon^{\mu})}}$$

Solving Hamilton's equations

- *Define the log-likelihood ratio as* $\ln \mathscr{L}_R = \langle s \,|\, h \rangle - \frac{1}{2} \langle h \,|\, h \rangle$
- *where the noise-weighted inner product is* $< h|s >= 2 \int_{f_{low}}^{f_{high}} \frac{df}{S_n(f)} \tilde{h}(f) \tilde{s}^*(f) + c.c.$
- *The integral has no closed-form solution, so needs to be solved numerically.*
- *Can do this using brute force integration, or use acceleration methods (e.g. multibanding)*
- *We can use the linearity of integration to break up the integral into smaller frequency bands*

$$\int_{f_0}^{f_{max}} \ldots df = \int_{f_0}^{f_1} \ldots df + \int_{f_1}^{f_2} \ldots df + \ldots + \int_{f_{n-2}}^{f_{n-1}} \ldots df + \int_{f_{n-1}}^{f_{max}} \ldots df$$

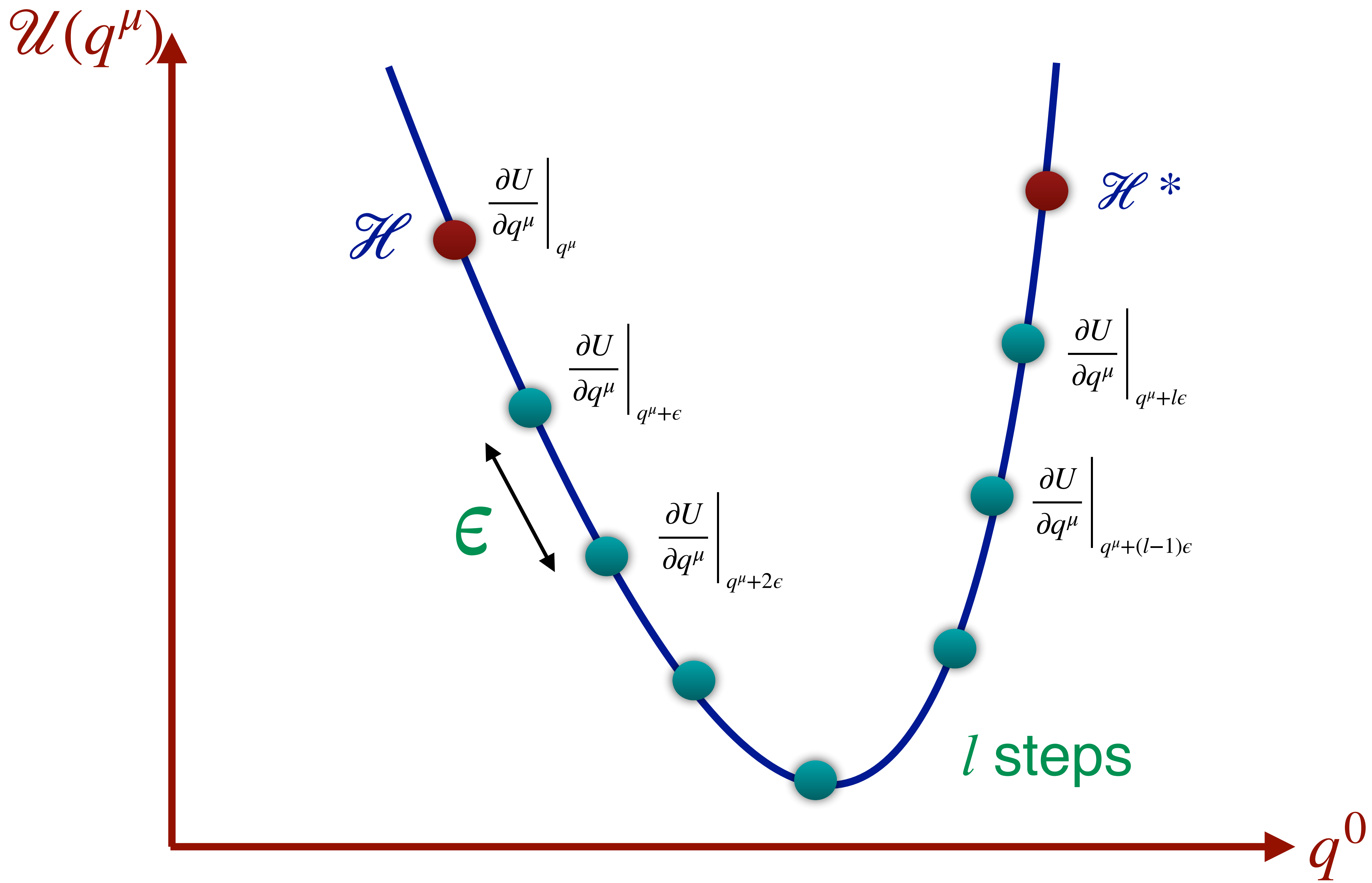
where each band has its own Nyquist frequency, meaning less evaluations.

- *This leads to an acceleration factor of around* $\sim (10^2)$

Solving Hamilton's equations

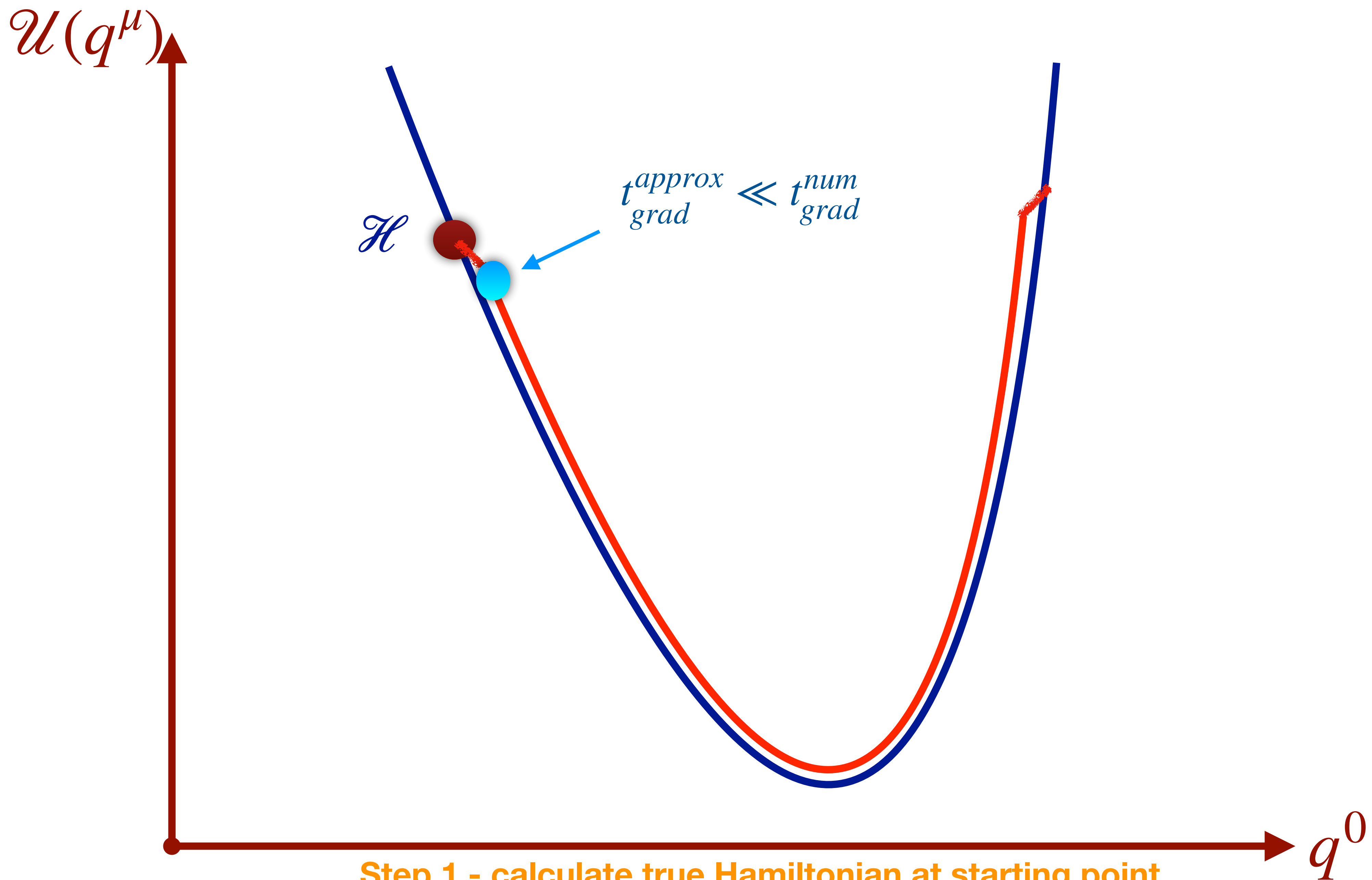
- *Some numbers:*
  - *12D parameter space (marginalizing over phase at coalescence)*
  - *use IMRPhenomD_NRTidal on 128secs of GW170817 data*
  - *each 12D gradient requires 13 waveform generations*
  - *each 12D gradient takes* ~ 2*secs to generate*
  - ~ 7*mins for a* $l = 200$ *step trajectory (2600 waveforms)*
  - $\sim$ ***1.26 years*** *for a 100,000 trajectory run (260 million waveforms)*
  - *clearly too slow, so what can we do?*
- *Possible solutions?*
  - *JAX autodiff: only works for waveforms with analytic solutions, doesn't work for EOB waveforms, multiband likelihoods, relative binning likelihoods…*

Solving the gradient bottleneck



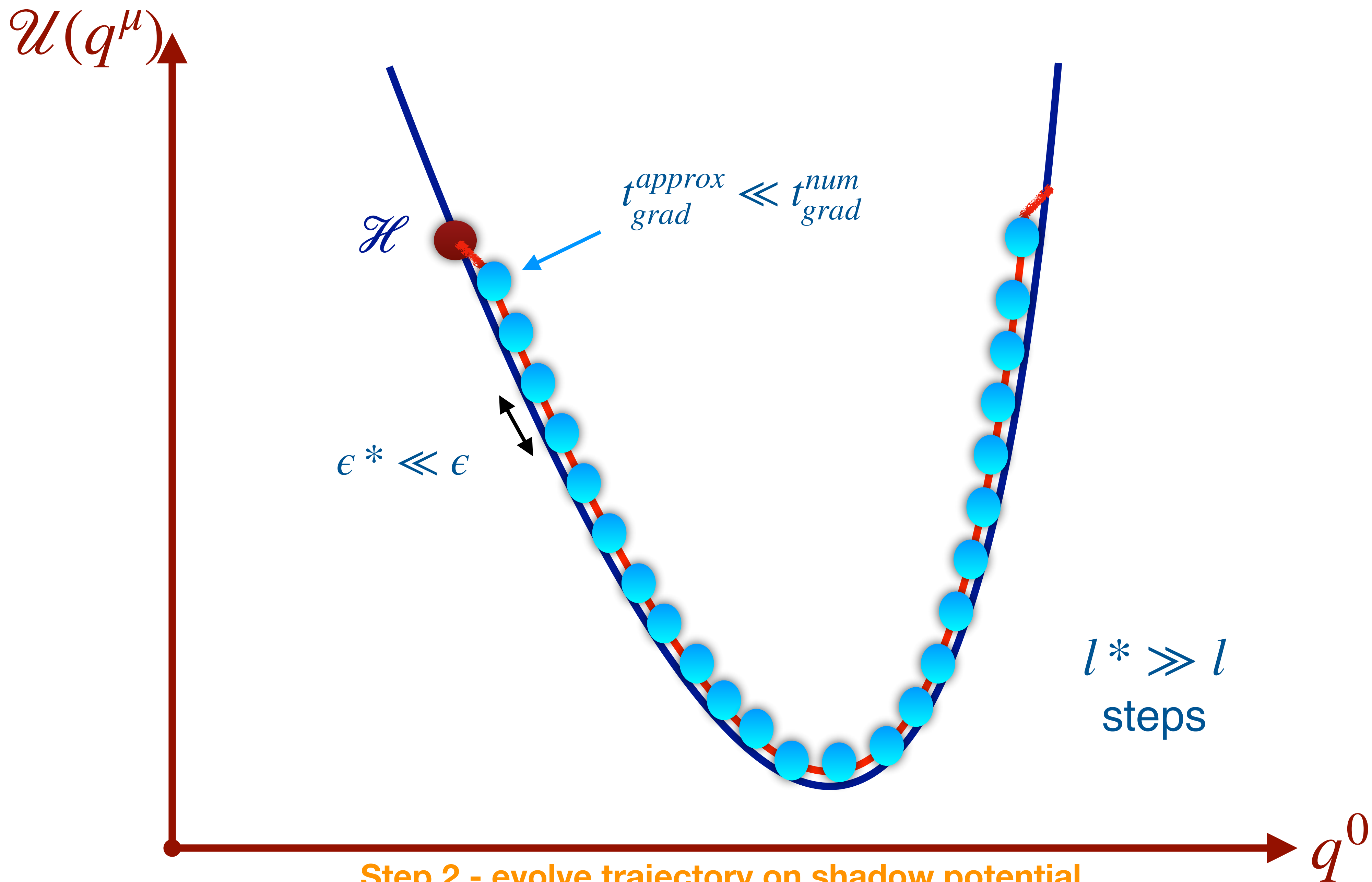
Moving in the shadow potential

Idea: create a shadow potential



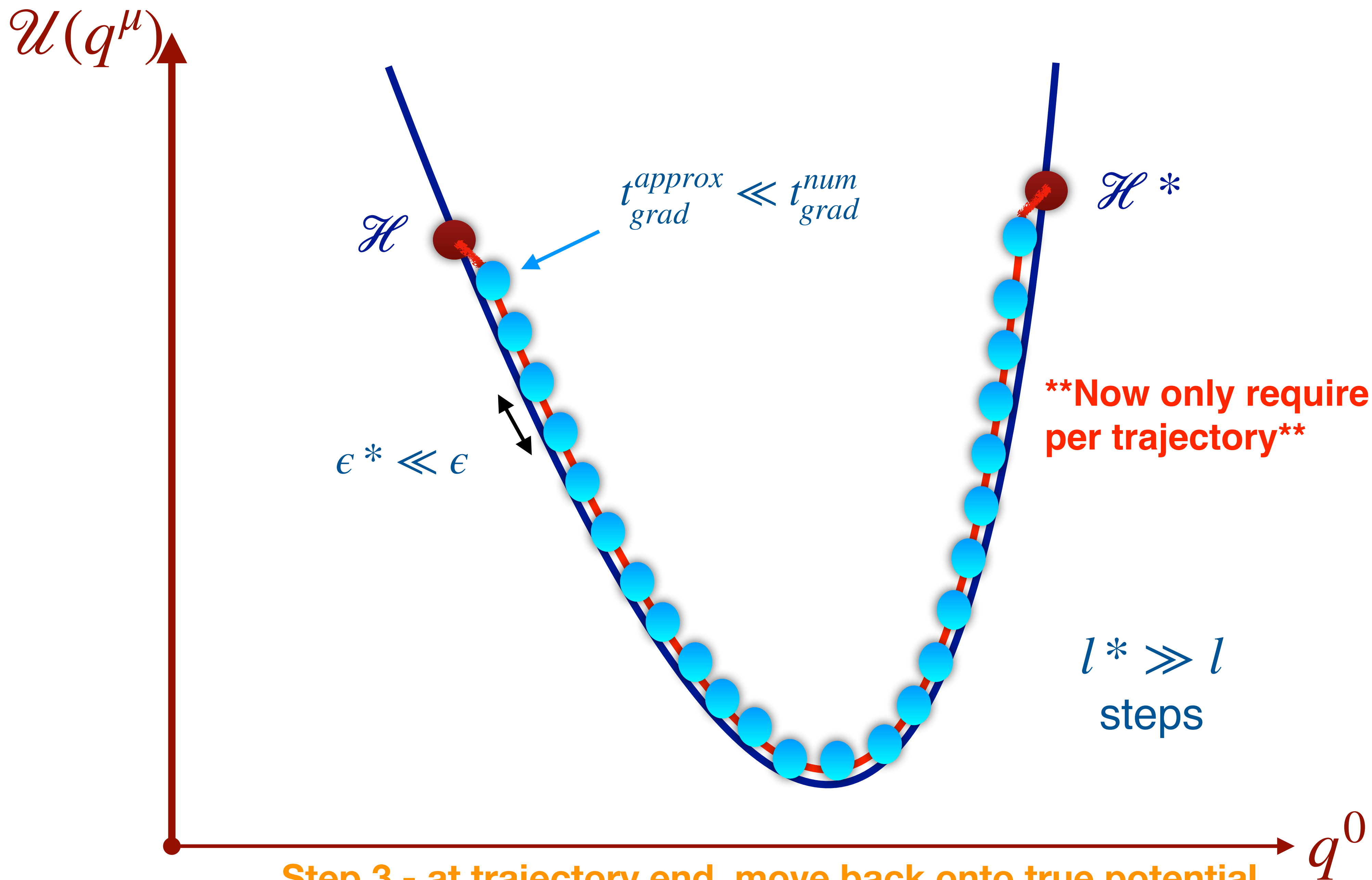
Step 1 - calculate true Hamiltonian at starting point
- then move to shadow potential

Moving in the shadow potential



Step 2 - evolve trajectory on shadow potential
- use approximate gradients and smaller step-size

Moving in the shadow potential

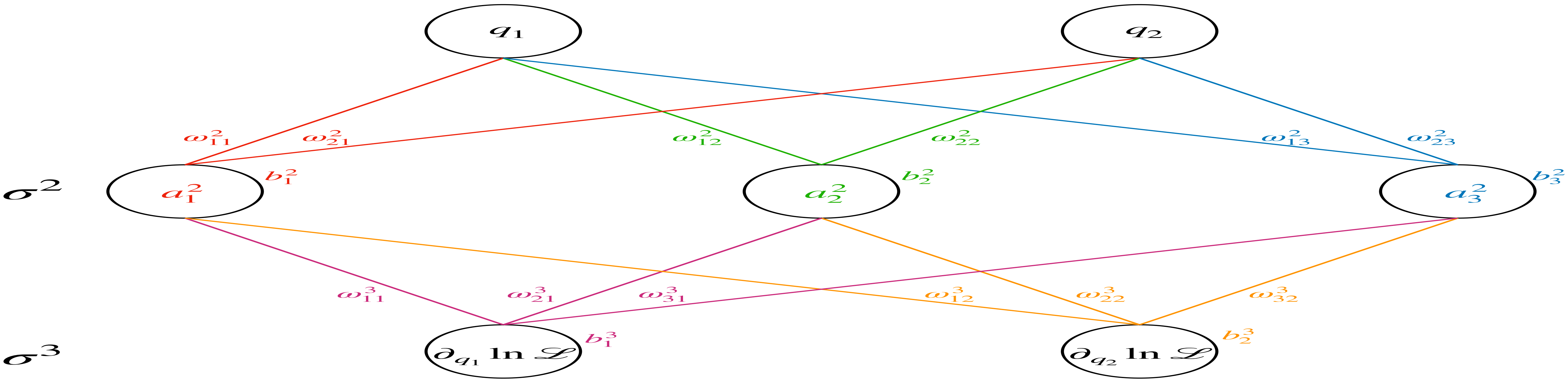


Step 3 - at trajectory end, move back onto true potential
- calculate Hamiltonian using waveform

Plan of Action

- *Divide the algorithm into 3 phases:*
    - *Phase I : evolve trajectories based on numerical gradient approximations, record all positional and gradient data. This data is never used for inference.*
    - *Phase II : use the Phase I data to build the shadow potential and DNN gradient models*
    - *Phase III: evolve trajectories based on the DNN gradient approximations to conduct a faster inference.*

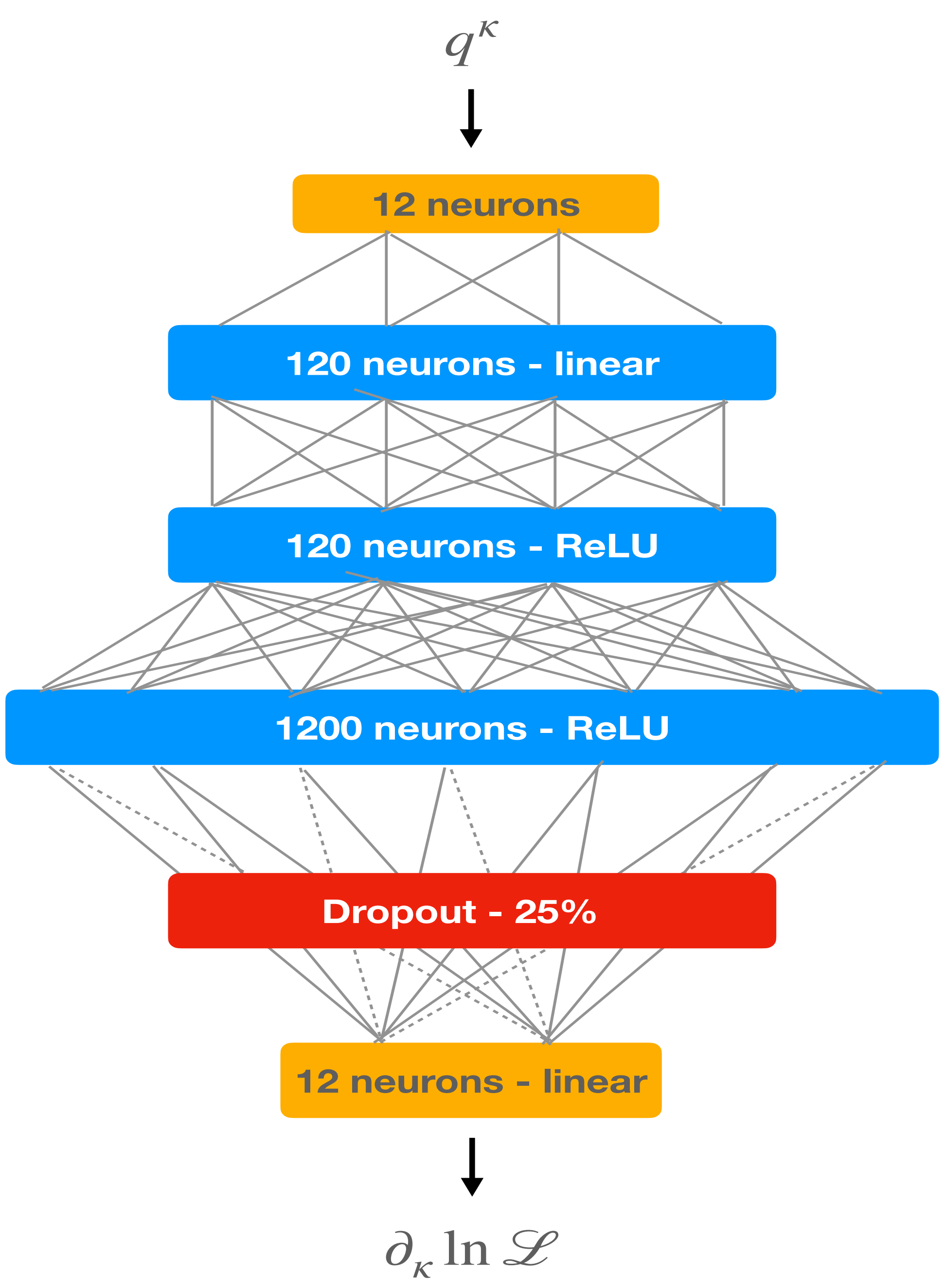
Constructing the DNN



DNN Architecture

- *To construct the architecture, we used* `TensorFlow/Keras`
- *Used* `Sequential()` *model to stack layers*
- *Stochastic gradient descent using* `Adam` *optmizer. Learning rate set to default value of* 10^{-3}
- *To build the architecture, used* `RandomSearch` *algorithm*
  - *3 hidden layers + 1 dropout layer*
  - *choice of (D, 5D, 10D) neurons for small layers*
  - *choice of (50D, 100D, 200D) neurons for large layers*
- `RandomSearch` *returned (10D, 10D, 100D)*
- *Upon investigation, we found (10D, 100D, 10D) gave similar performance, but was 1.4 times faster*

DNN Architecture



Tuning the network training

- *During the development, moved from Intel-based to ARM-based Apple silicon*
- *Required moving from* `TensorFlow` *to* `PyTorch`
- *FYI, needed to force 32bit modelling in PyTorch*
- *Normalised the data using scikit-learn's* `StandardScalar`
- *Used a mean-squared error loss function to measure the tuning, i.e.*

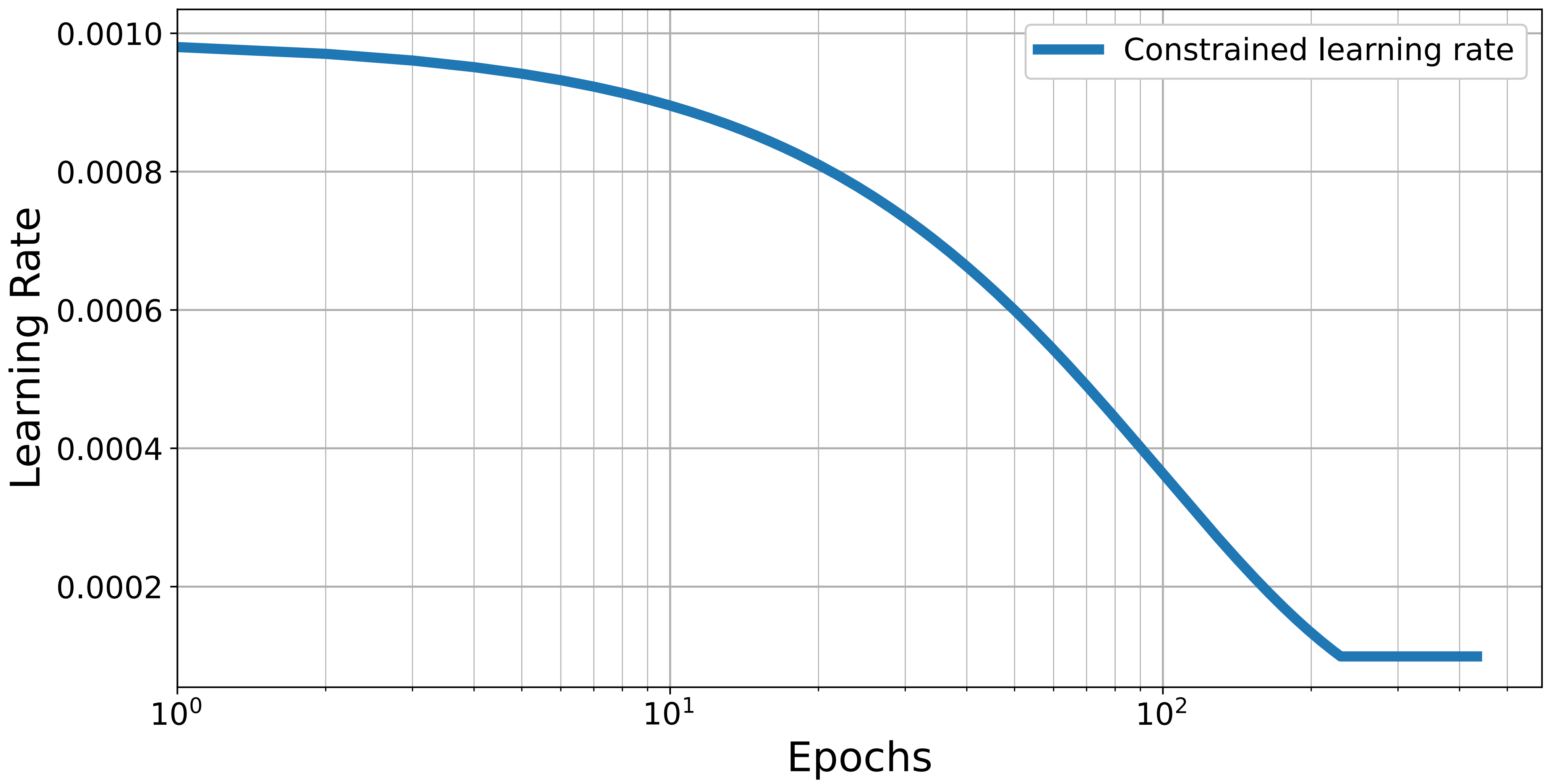
$$MSE = \frac{1}{BD}\sum_{i=0}^{B-1}\sum_{\kappa=0}^{D-1}\left[(\partial_\kappa \ln \mathcal{L})^i_{num} - (\partial_\kappa \ln \mathcal{L})^i_{DNN}\right]^2$$

- *Data split: 70% training, 20% validation, 10% test*

Tuning the network training

- *Tuning the learning rate:*
  - *Stochastic descent method using the Adam optimiser*
  - *Initially, used default learning rate* $R_L = 10^{-3}$
  - *Switched to the PyTorch* `ExponetialLR` *function*
    - $R_L^{e+1} = \gamma R_L^e$
  - *with* $R_L^0 = 10^{-3}, \gamma = 0.99$ *and* $R_L^B = 10^{-4}$

Tuning the network training



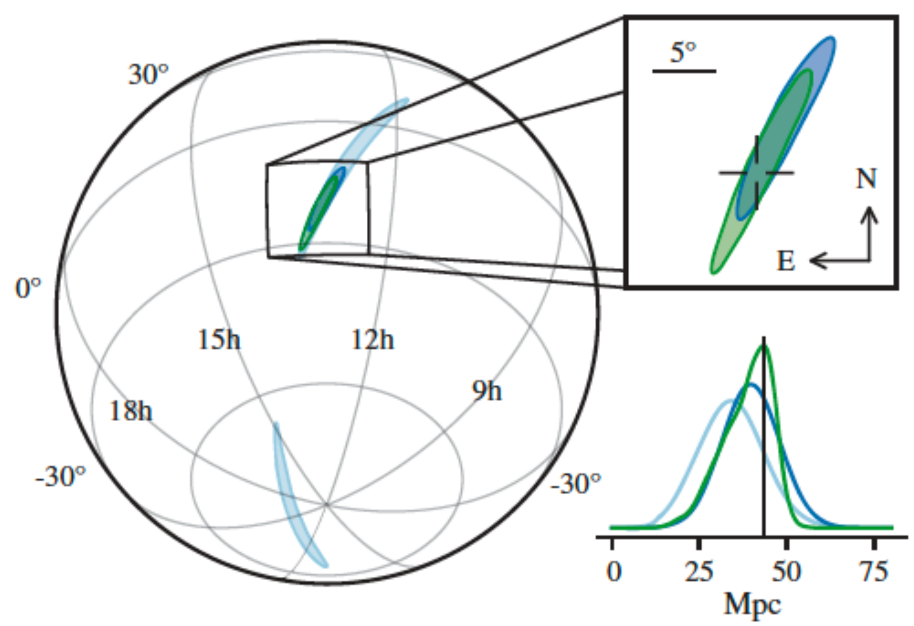
Tuning the network training

- *Tuning the Batch size:*
    - *Tried different batch sizes between* $B \in [2^4, 2^{15}]$
    - $B \in [16, 128]$ *quickly converged to low error loss*
    - *In terms of accuracy,* $B = 128 \sim B = 32$*, but almost 2x faster*

Tuning the network training

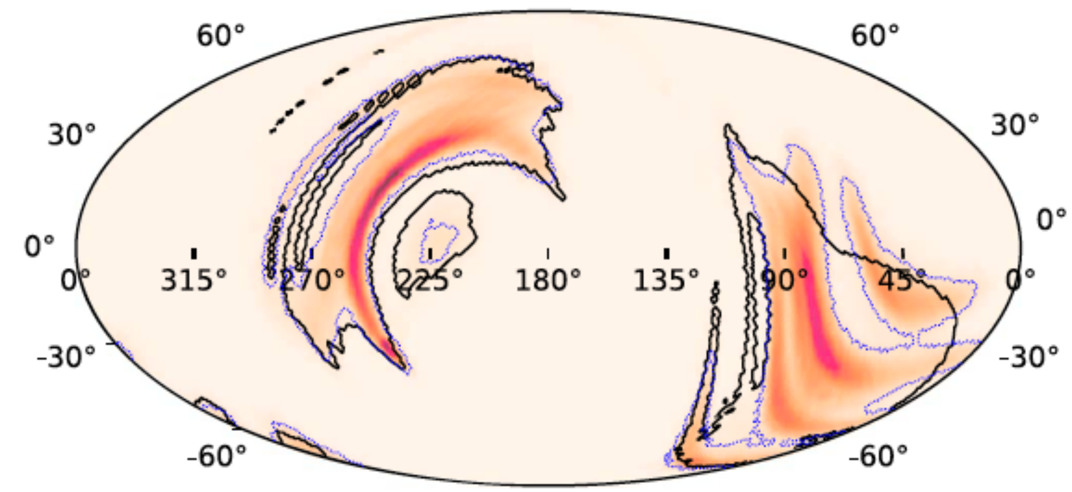
- *Tuning the number of training epochs:*

GW170817



3 detector event
SNR ~ 32

GW190425



single detector event
SNR ~ 12

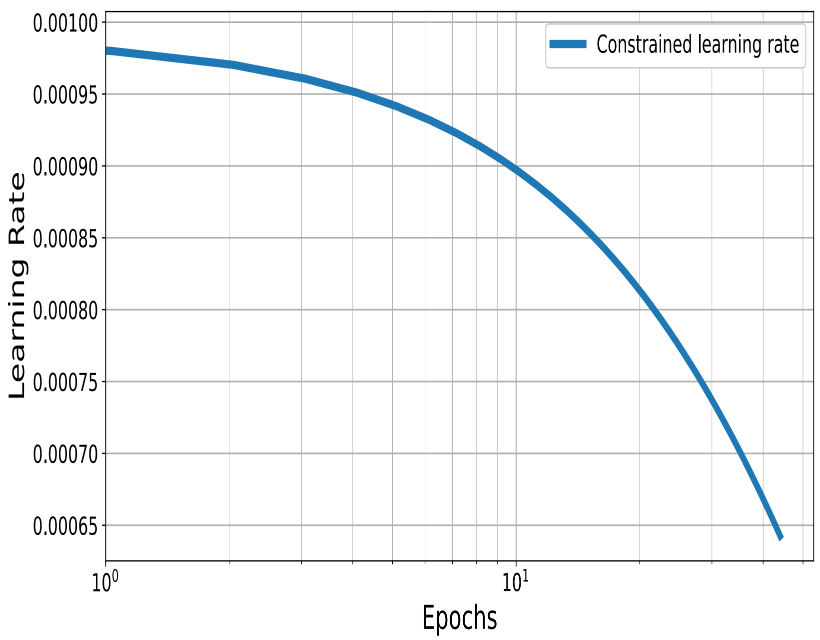
Tuning the network training

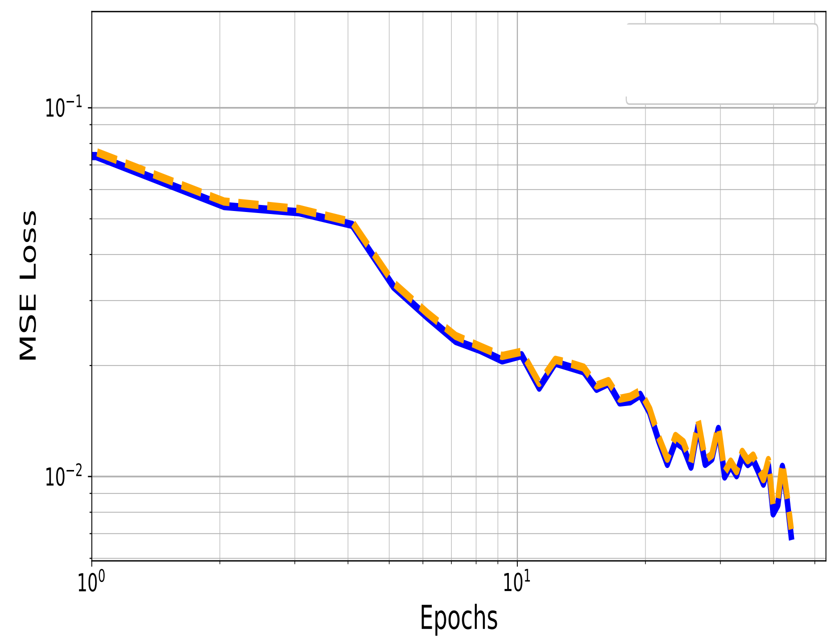
- *Tuning the number of training epochs:*
- *Coefficient of determination :* $R^2 = 1 - \frac{SS_{res}}{SS_{tot}} = 1 - \frac{\sum_{i=1}^{N}\left[\left(\partial_\mu \ln L\right)_i^{num} - \left(\partial_\mu \ln L\right)_i^{app}\right]^2}{\sum_{i=1}^{N}\left[\left(\partial_\mu \ln L\right)_i^{num} - \left\langle\left(\partial_\mu \ln L\right)^{num}\right\rangle\right]^2}$
  - *GW170817*
    - *Threshold :* $R^2 \geq 0.99$ *or 500 epochs*
    - *Threshold reached in 40-500 epochs (10-100 minutes)*
  - *GW190425*
    - *Couldn't reach* $R^2 > 0.95$ *in under 20,000 epochs*
    - *Threshold :* $R^2 \geq 0.9$ *or 1,000 epochs*
    - *Threshold reached in 400-800 epochs (80-200 minutes)*

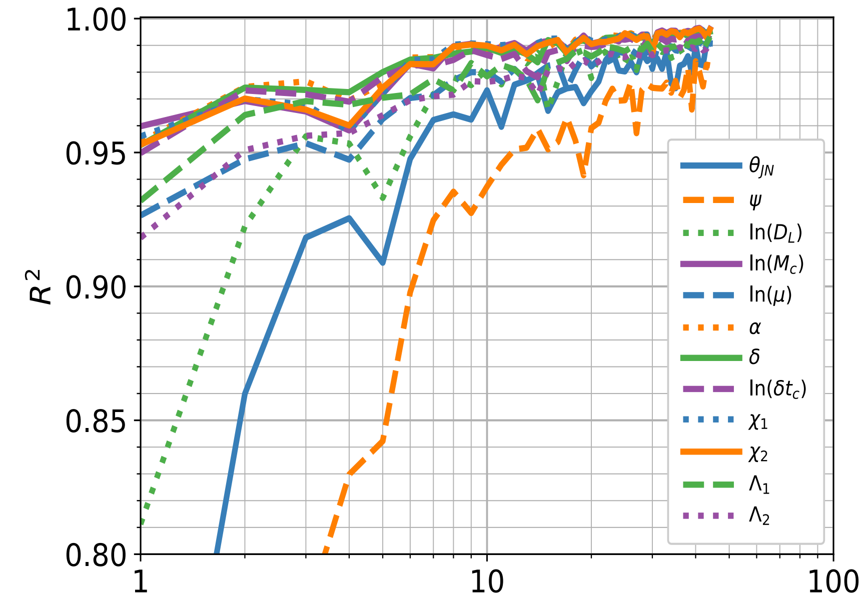
Tuning the network training

	Learning Rate	Training vs. Validation	R^2 evolution

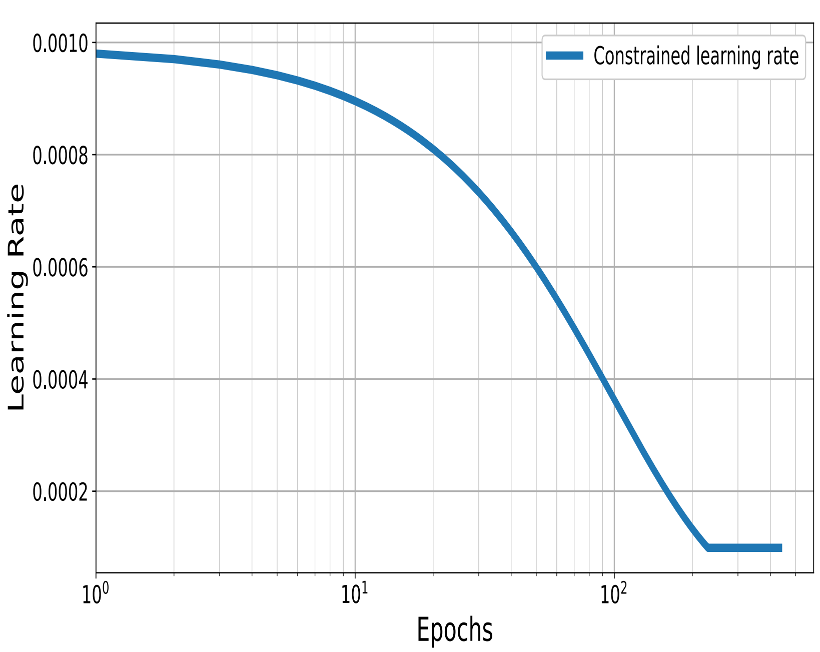
GW170817

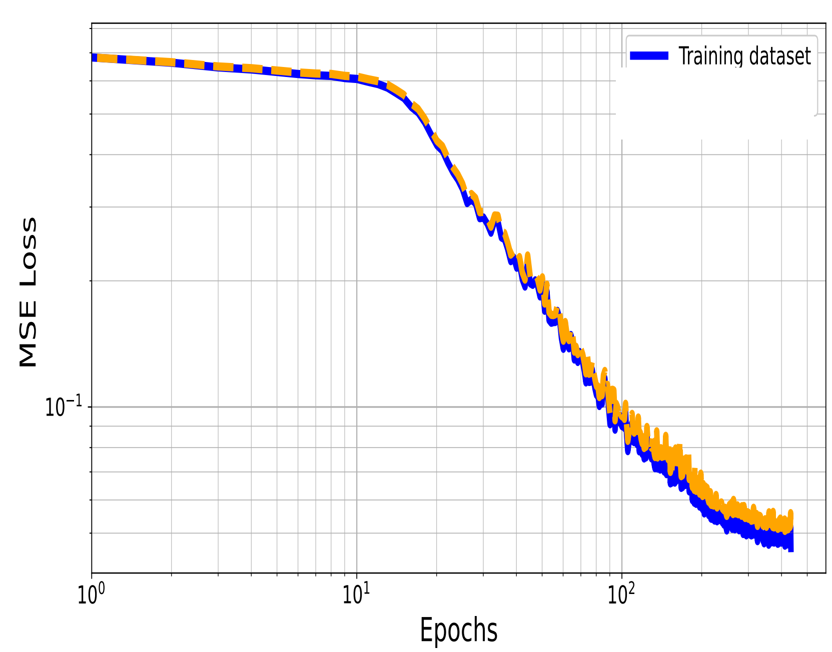


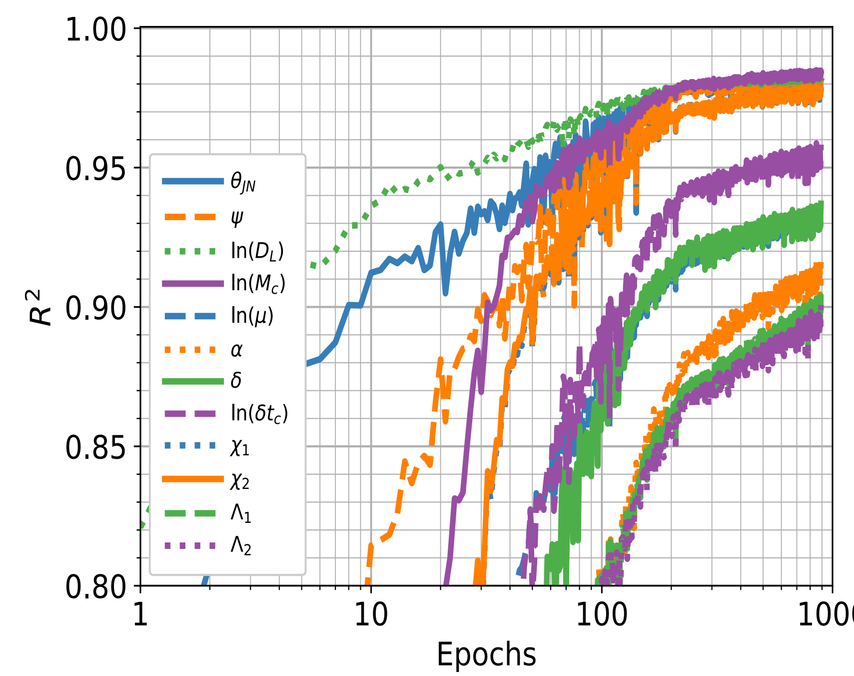




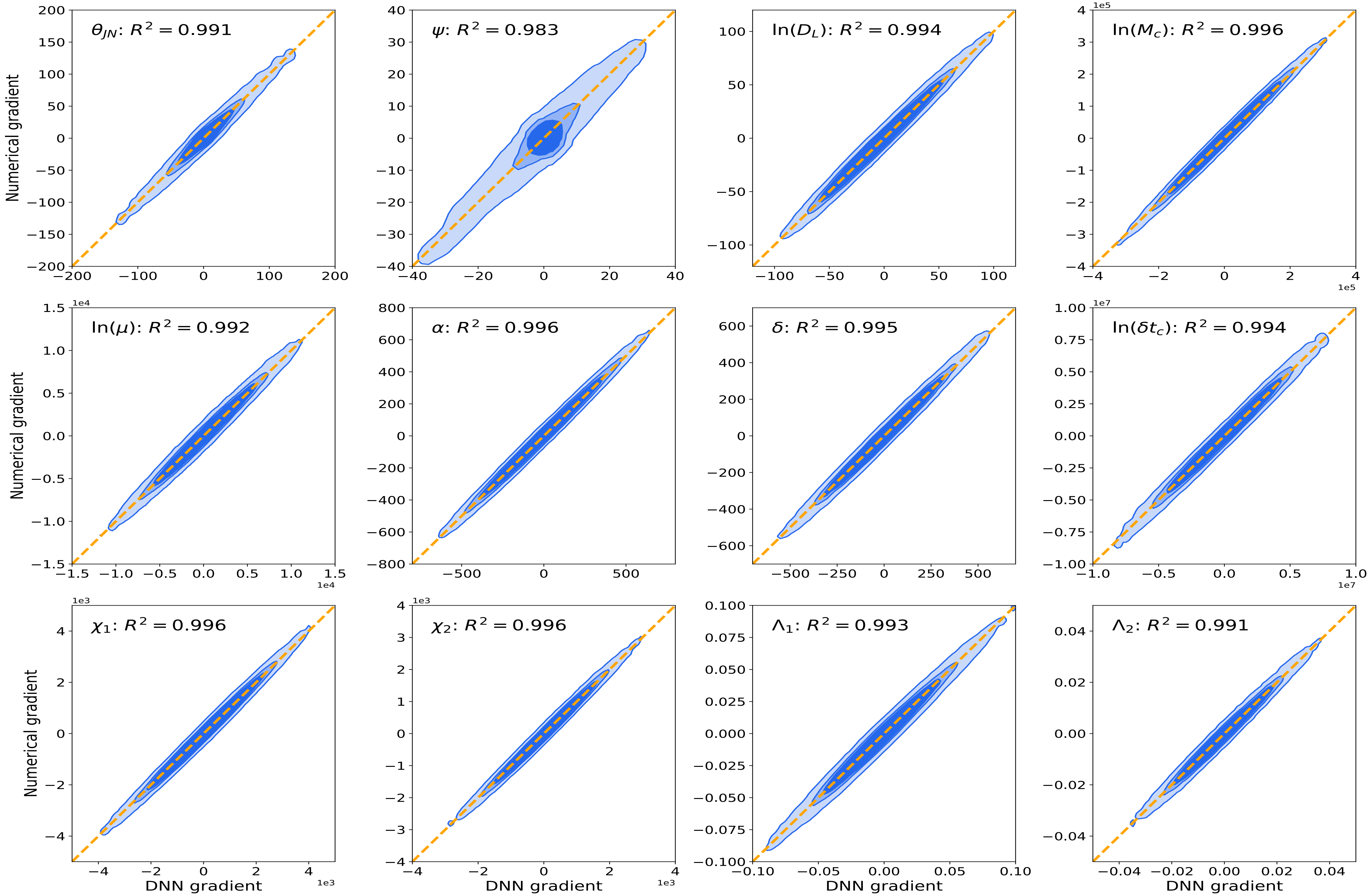
GW190425



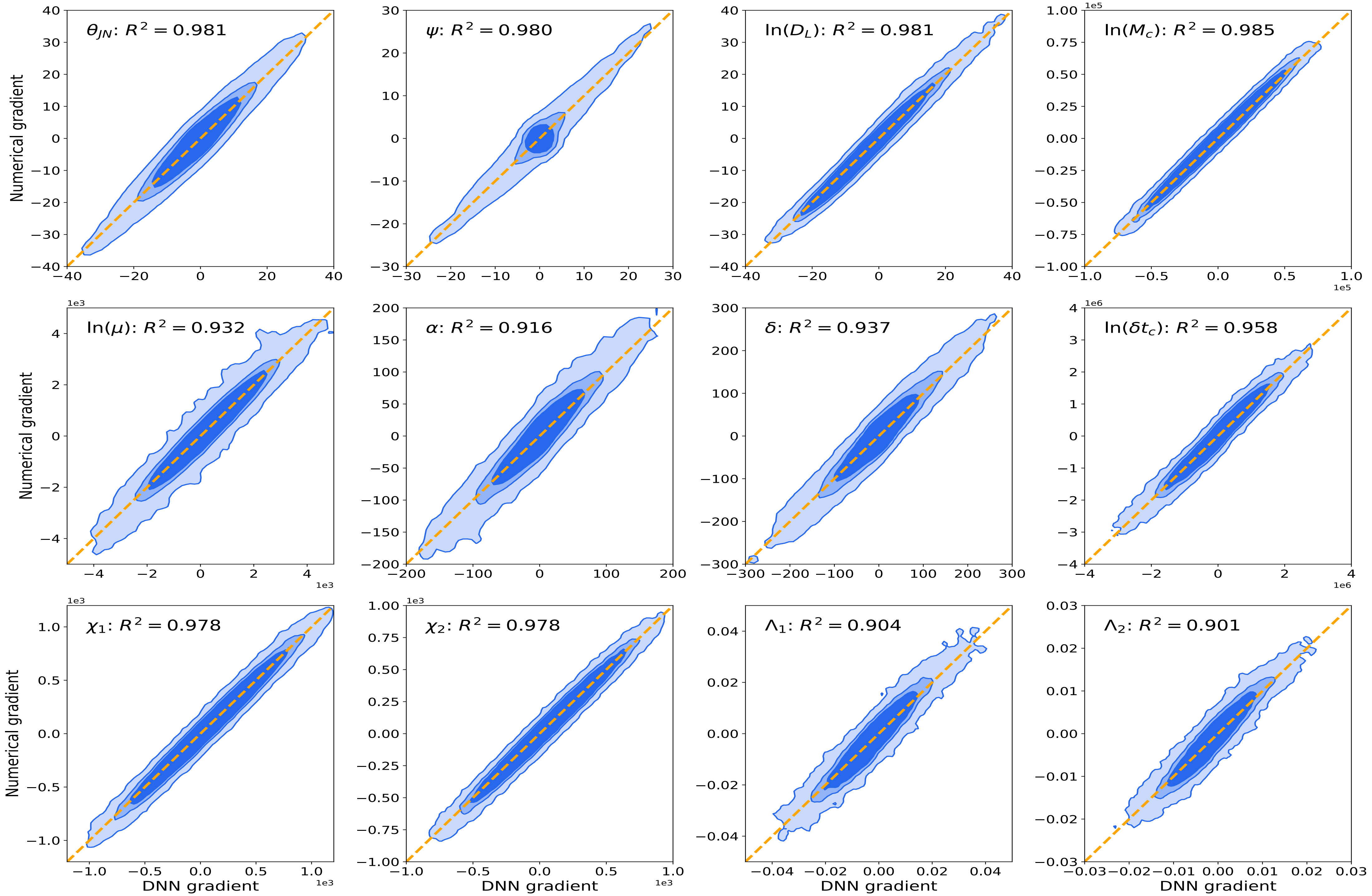




GW170817: DNN vs. test data

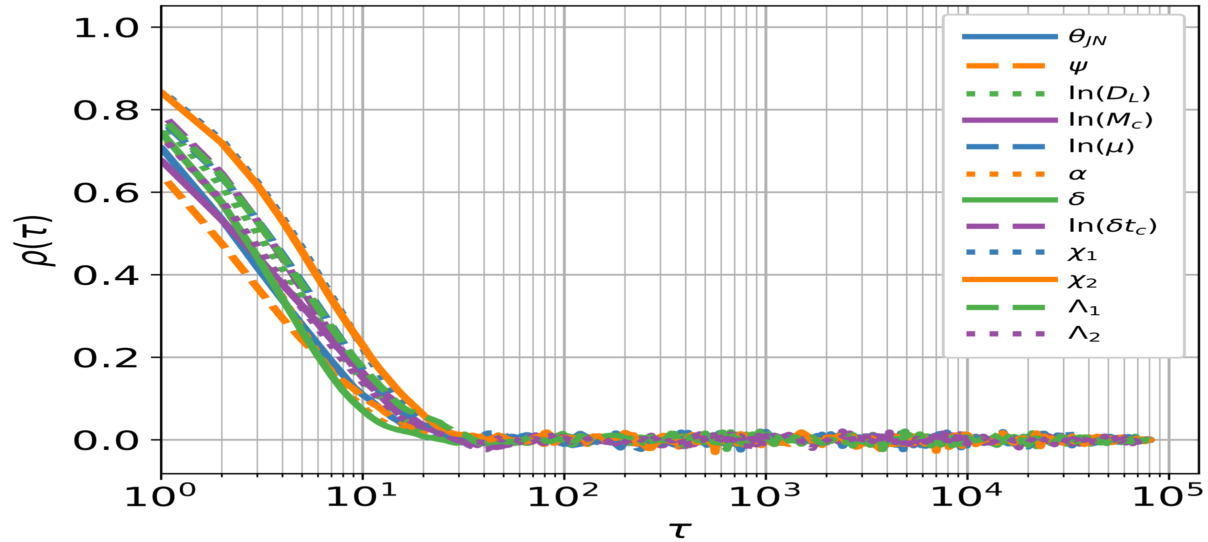


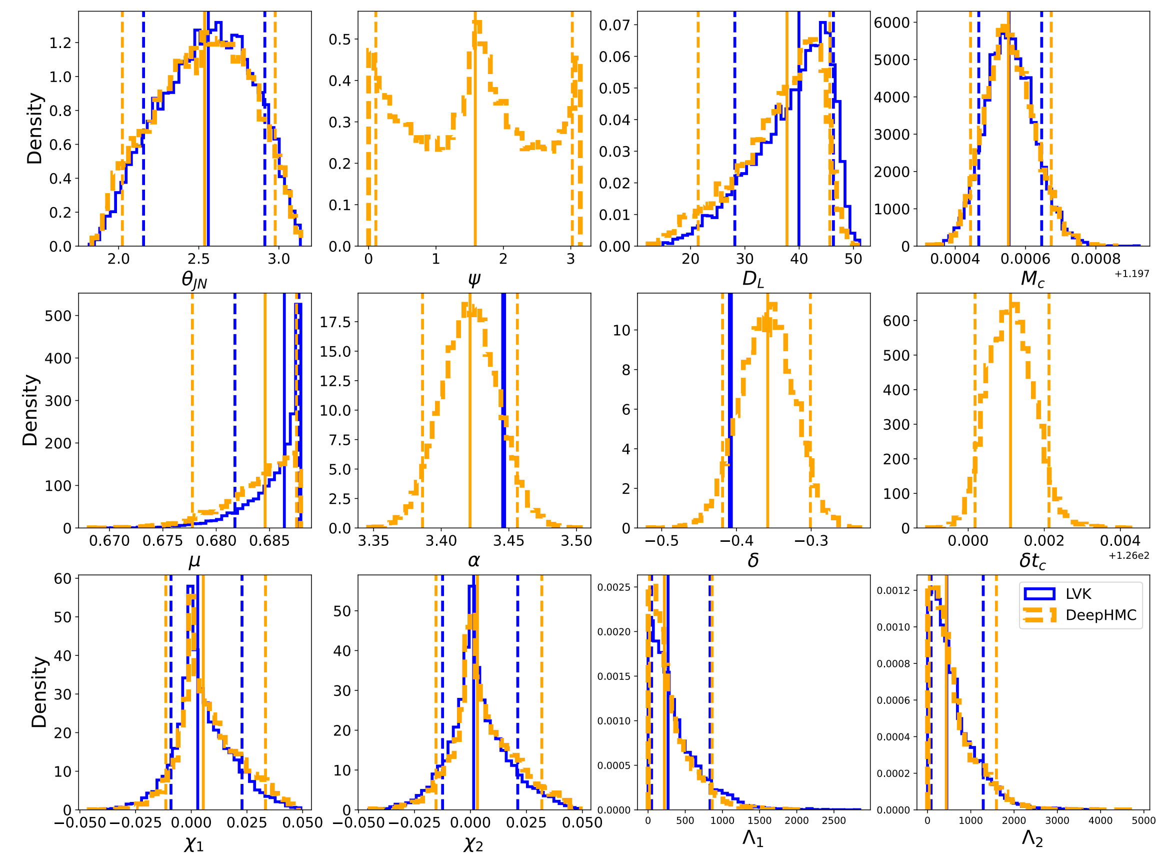
GW190425: DNN vs. test data



Application to GW170817

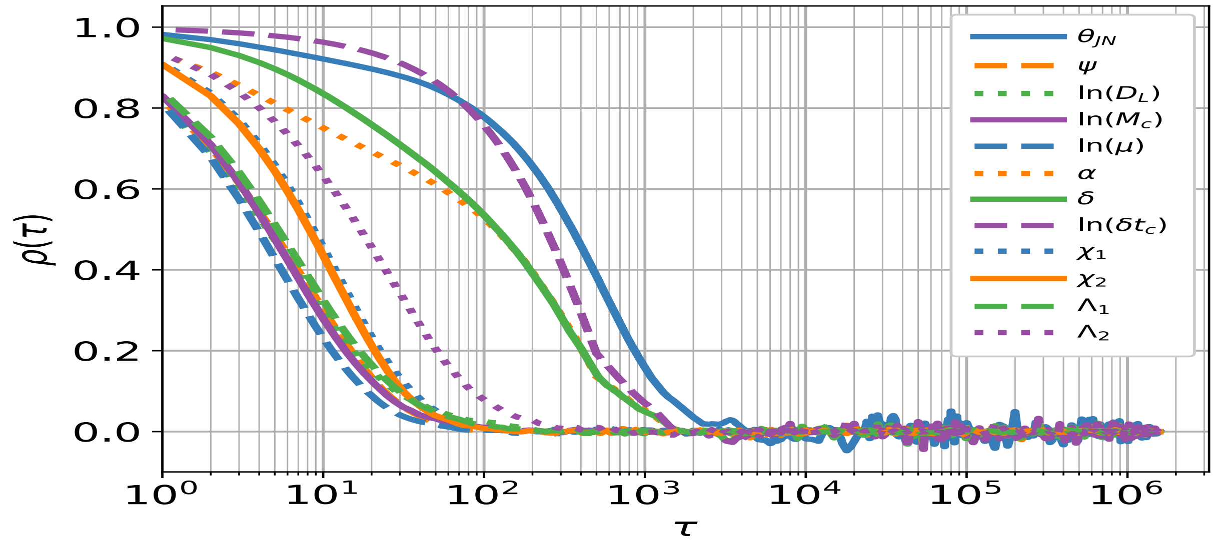
- *90-120 minutes to obtain 5000 SISs*
- *>80% acceptance rate*
- $\rho(\tau) = 0$ *at less than 100 lags*
- *IAT = 10*
- *1 SIS/second*

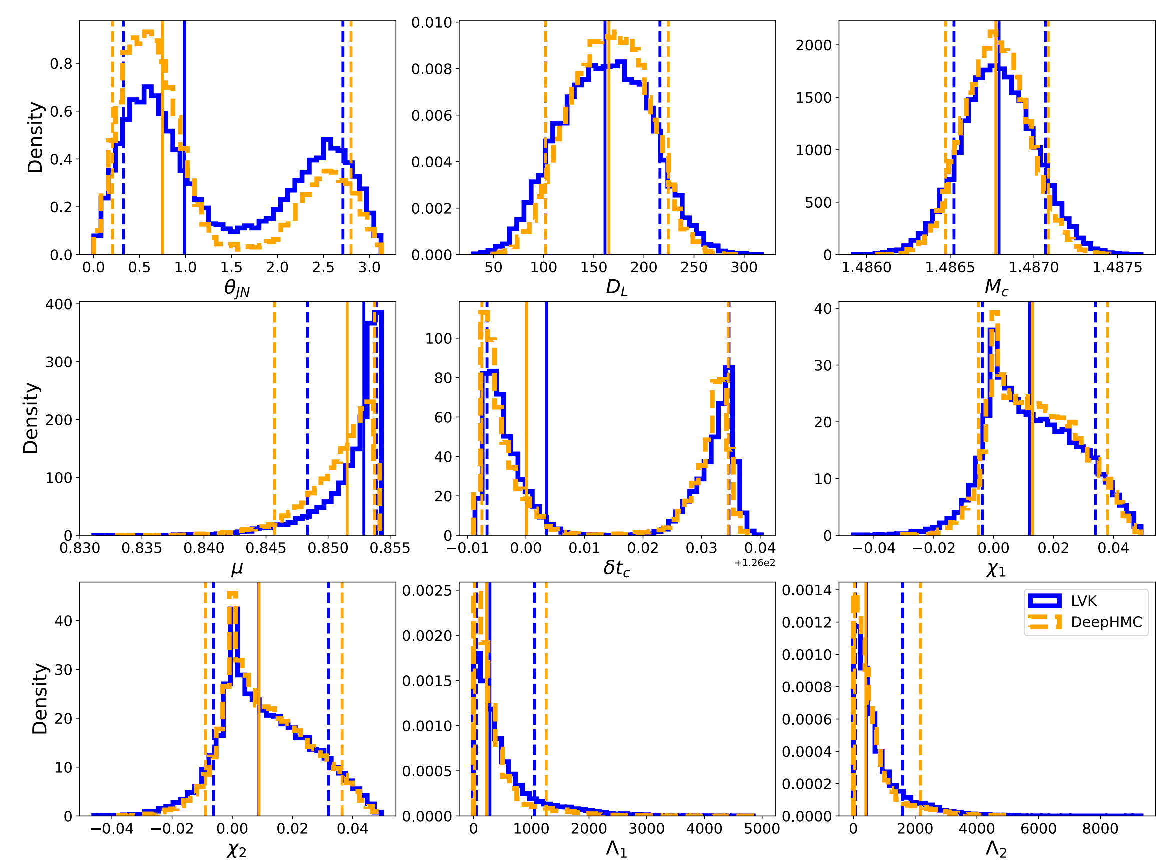




Application to GW190425

- *55-60 hours to obtain 5000 SISs*
- *>80% acceptance rate*
- $\rho(\tau) = 0$ *at less than 10,000 lags*
- *IAT = 200-300*
- *1 SIS every 26 seconds*





Summary

	GW170817	GW190425
Phase 1: data gathering (660,000 data points)	20 minutes	25 minutes
Phase 2: constructing the DNN gradients	10-100 minutes	80-200 minutes
Phase 3: GW inference	90-120 minutes	55-60 hours
Total	2-4 hours	57-64 hours (~2.5 days)

Conclusion

- *DNN based HMC for GW parameter estimation*
- *Accelerates gradient calculation of target posterior by using a DNN model*
- *DNN gradients are 30x faster than our state-of-the-art acceleration methods and 7000x faster than a brute force likelihood integration*
- *Excellent agreement with LVK public results*
- *Can be extended for extra dimensions / source types / problems*