Exploring the feasibility of neural likelihood estimator for fast parameter estimation.

Luca Negri

Utrecht University

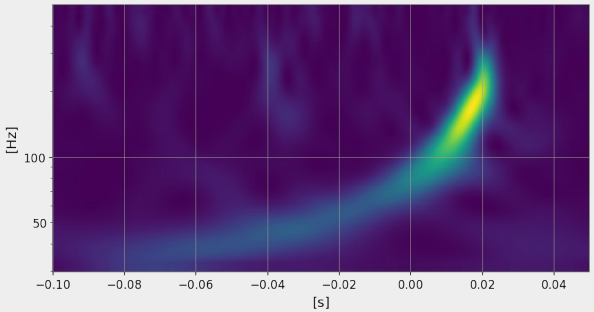
Nikhef

Belgian-Dutch gravitational waves meeting
Nijmegen, 27 October 2025

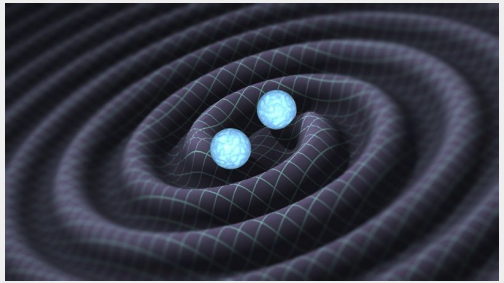
Parameter estimation

Parameter estimation answers a very simple question:

How can we link this …



… to this?



Credit : Caltech

Answers lies in the **Bayes theorem:**

Likelihood Prior

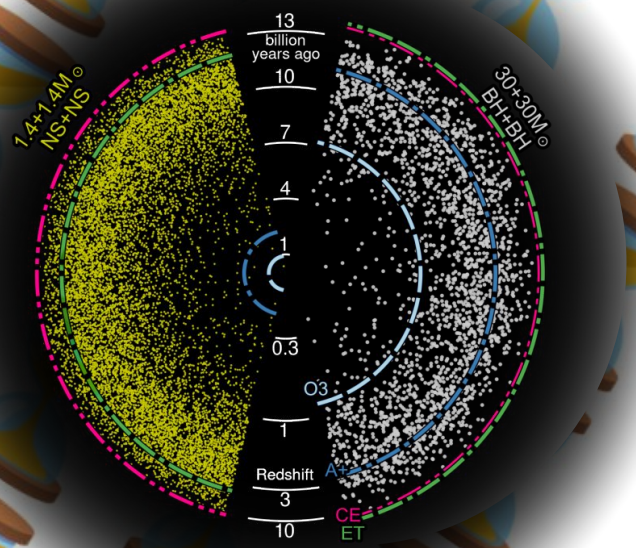
Posterior: what we want!

$$p(\vec{\theta}|y,\mathcal{M}) = \frac{p(y|\vec{\theta},\mathcal{M})\,p(\vec{\theta}|\mathcal{M})}{p(y|\mathcal{M})},$$

Evidence: Can be used for model comparison

But to fully represent the high dimensional space (9-17 dim)
You need to sample the system $\boldsymbol{O(10^6)}$, $\boldsymbol{O(10^7)}$ **times**

Parameter Estimation in the future



Credit : Evan Hall, Salvatore Vitale/ MIT

Next gen of interferometers will come with a new set of challenges for parameter estimation:

- ~1000 times more signals
- Signals will be louder
- Signals will be longer
- Overlapping signals

We are not ready to analyze this data!

Same signal often analyzed multiple times with

- different noise models
- different waveforms
- different physics

Cannot have one model to fit them all!

Neural Likelihood Estimators: the most flexible solution?

Many ML based solutions proposed to speed up computations and reduce costs , each with its own strengths and weaknesses:

- Neural posterior estimators
- Neural ratio estimators
- Neural proposal estimators
- Neural likelihood estimators:
  - Fix the data, and the likelihood function depends only on the parameters (9-17 dimensions)
  - Approximate the likelihood function $\mathbb{R}^{15} \rightarrow \mathbb{R}^{1}$ with a neural network

Takes ms to s to evaluate

$$p(\vec{\theta}|y,\mathcal{M}) = \frac{p(y|\vec{\theta},\mathcal{M})p(\vec{\theta}|\mathcal{M})}{p(y|\mathcal{M})},$$

Takes less than a μs even on modest hardware

$$p(\vec{\theta}|y,\mathcal{M}) = \frac{p(\vec{\theta}|\mathcal{M})}{p(y|\mathcal{M})},$$

PROs

✅ **Speedup** of single likelihood evaluation of $>10^5$

✅ **No need for special hardware** (CPUs only)

✅ **Plug and play:** avoids expensive pre-training

✅ **FLEXibility**! Any sampler, waveform or prior works.

BUT … :

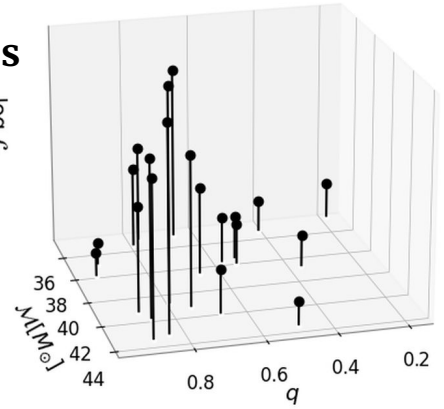
🤔 New NN trained on the fly for every signal

🤔 Training adds time to full inference

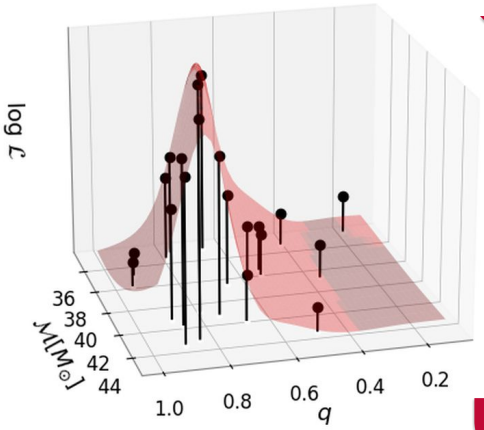
🤔 **Can NNs even be accurate enough?**

The FLEX cycle

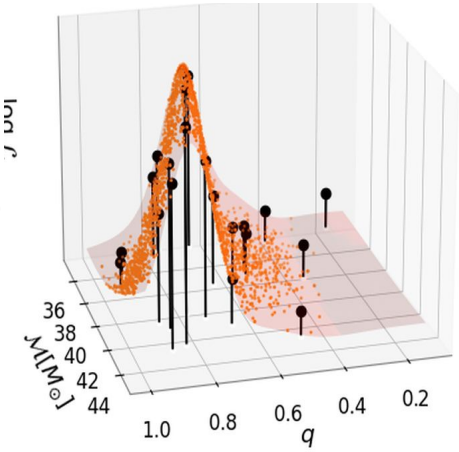
1: Generate training samples



4: Check results and retrain



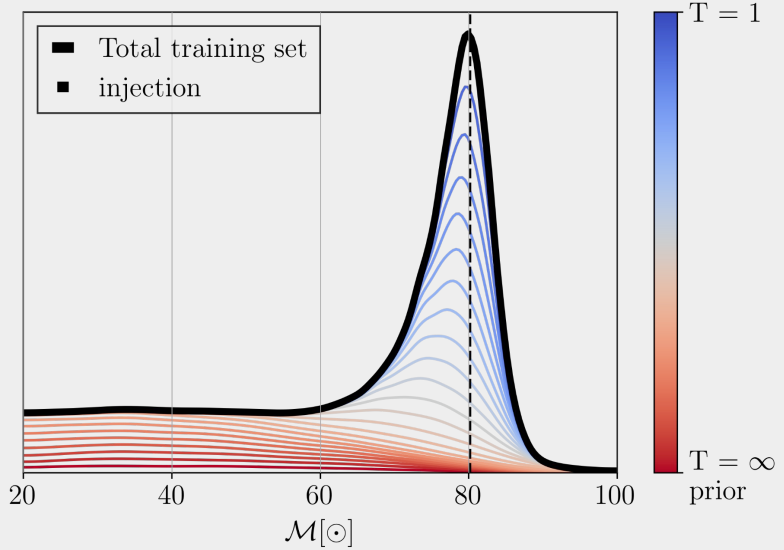
2: Train Neural Likelihood

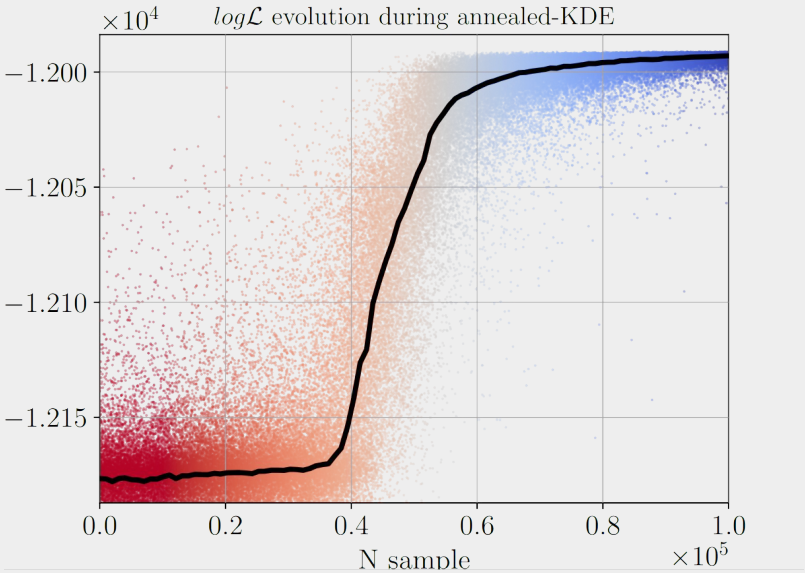


3: Run sampler on the Neural likelihood

Phase 1 : Generate initial training data

Samples obtained through an annealed series of tempered KDEs, which interpolate smoothly between prior and posterior, similar to Sequential Monte Carlo (SMC)





$$\theta_i \sim p(\theta|y,\mathcal{M})^{1/T}$$

For the algorithm to be effective
$N_{train} << N_{MCMC}$ ($10^{6\text{-}7}$)

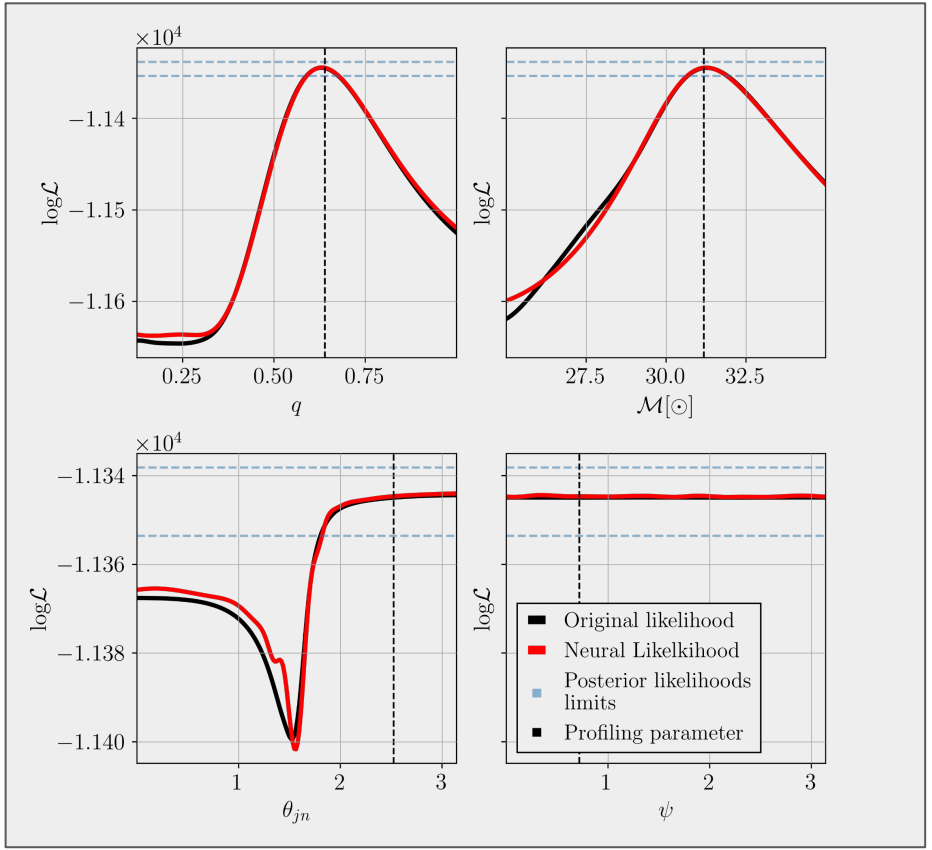
Phase 2 : Training the network

Keep network architecture as simple as possible:

- Small resnet, fully connected
- ~15k parameters
- Fast to train, fast to analyze

$$\text{Loss} = \frac{1}{N_t}\sum_{i=0}^{N_t}(e^{\log \mathcal{L}(\theta_i)} - e^{\text{NN}(\theta_i)})^2 w_i$$

Weights boost importance of samples in low density regions of parameter space

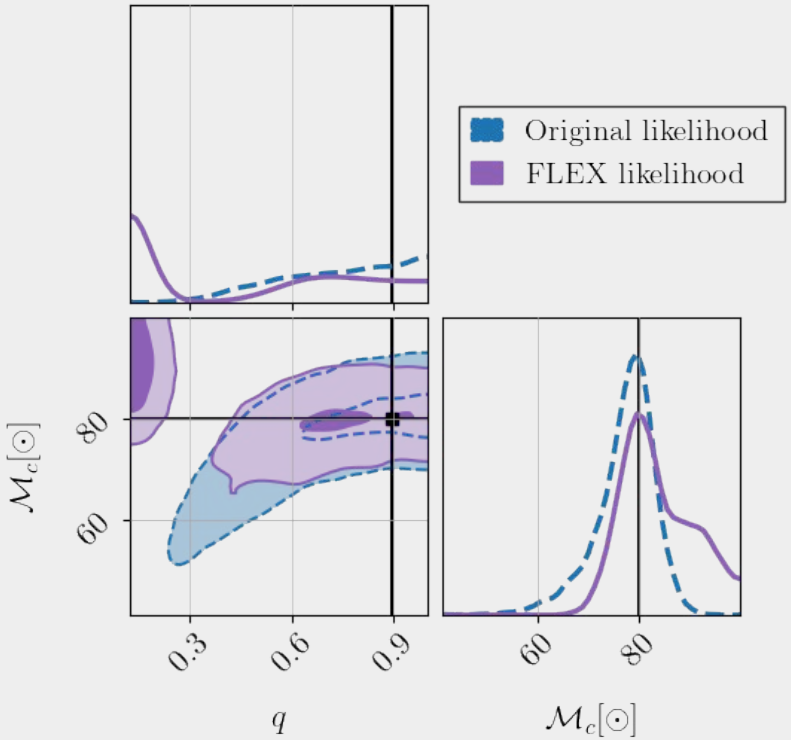


Phase 3 : Running MCMC

Posterior obtained through highly vectorizable MCMC sampler with parallel tempering (`eryn` [1])

- Very modest hardware (CPU)
- The PE run on the FLEX likelihood takes ~ 1 minute on even 1 CPU

But … what if the results are off?



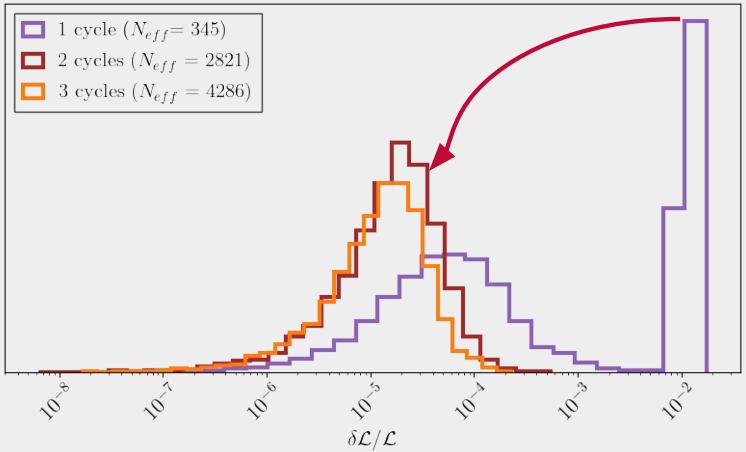
[1] N. Karnesis *et al.*: 2023:

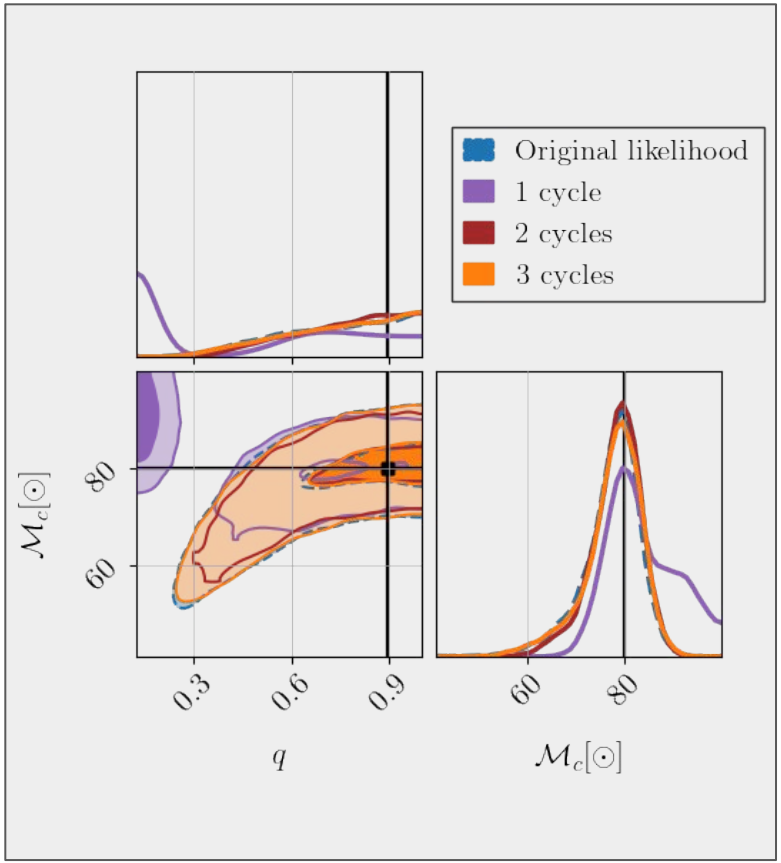
Phase 4 : Check and, if necessary, retrain

Calculating Effective Sample Size (ESS):

$$w_i = \mathcal{L}(\theta_i)/\mathrm{NN}(\theta_i), ESS = \frac{(\sum_i w_i)^2}{\sum_i w_i^2}$$

If ESS/N_{post}< Threshold (50%) restart cycle from phase 1, with added training samples from the (temporary) posterior



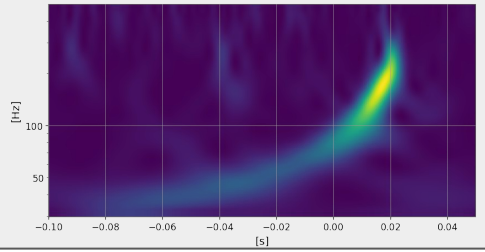


Results: setup

Which signal type will we tackle?

Start with the “easiest” problem in GWs:

- Heavy mass, short duration, BBHs [20-100 chirp mass]
- SNR<40
- 2-3 detector networks, O4-like sensitivities
- Align spin (no precession) IMRPhenomD
- Distance & phase analytical marginalization
- Sampling over 9 parameters : $[\mathcal{M}, q, \chi_1, \chi_2, \theta_{jn}, \psi, \text{zenith}, \text{azimuth}, t_{det}]$



GW150914-like signals

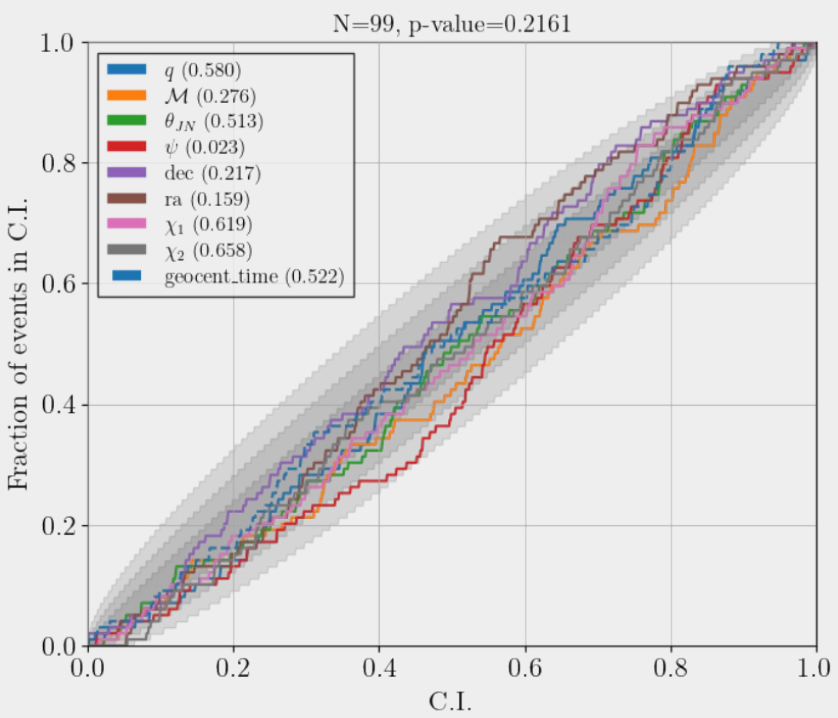
FLEX Hyperparameters :

- Initial waveforms evaluations : 1e5
- NN Training epochs (per cycle): 700
- Retraining samples (per cycle): 2e4
- Retraining samples temperature: 4
- ESS /N_{tot} threshold : 50%
- N_{tot} : 5000
- Max number of cycles : 6

Optimized to minimize likelihood evaluations (max 2e5)

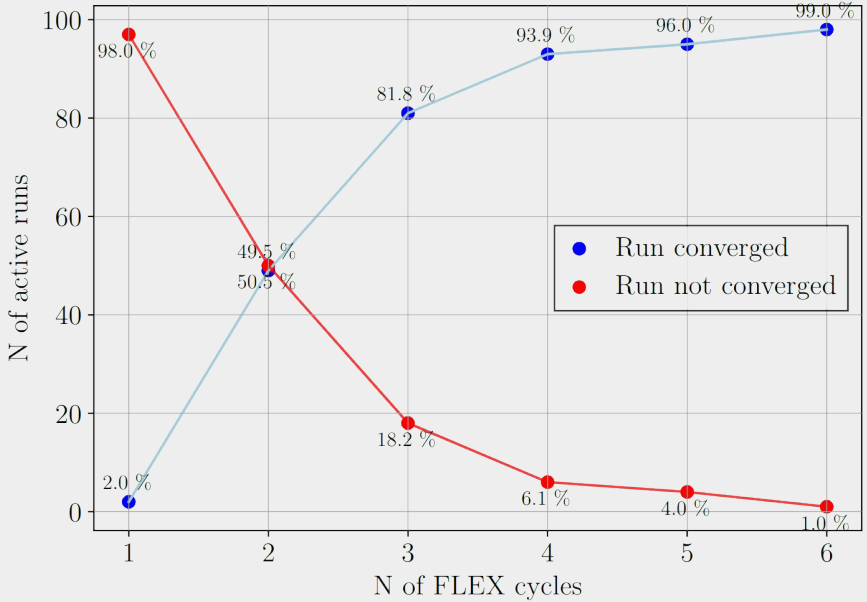
Results: Robustness tests

PP-plot shows promising results!

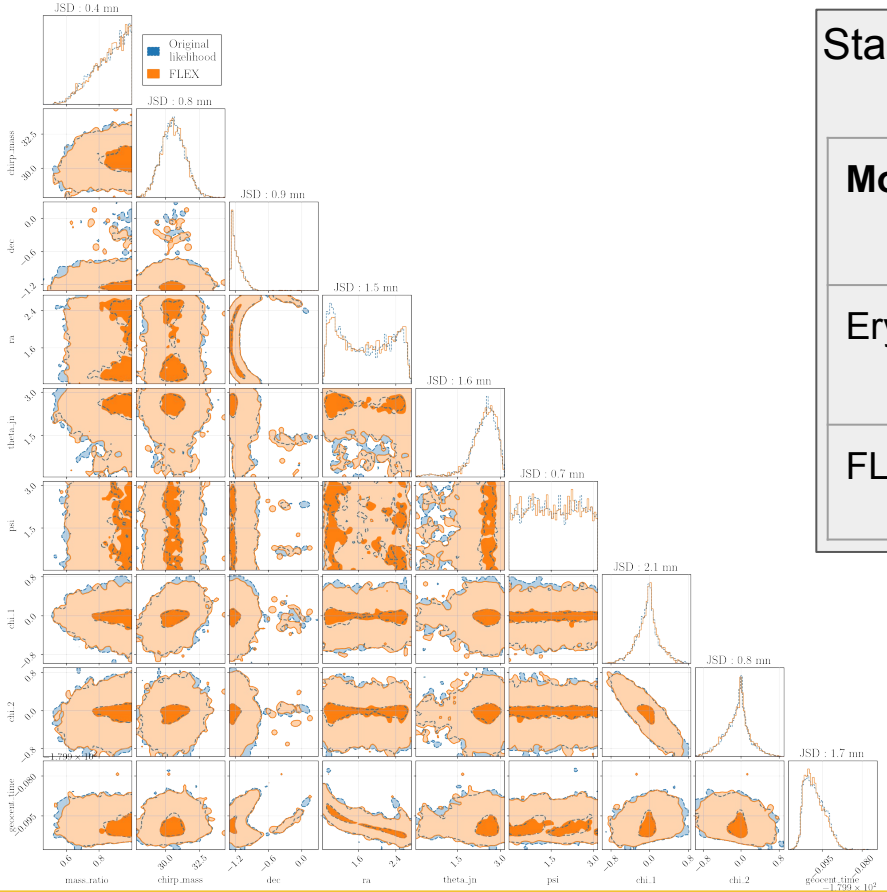


99 runs on simulated signals

- Only 1 did not reach the ESS threshold of 50%
- Half of the runs converged after 2 cycles



Results: 150914



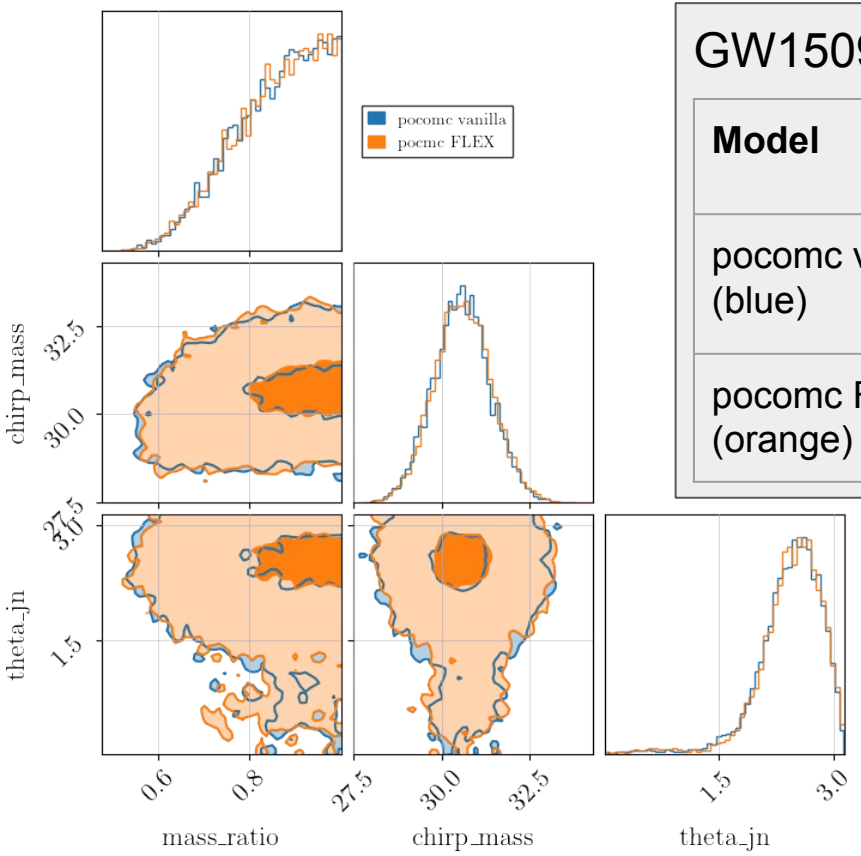
Standard test with real data, GW150914:

Model	**Time** (on same hardware)	**Number of Waveforms**
Eryn (blue)	~12 hours	~ 1.3e7
FLEX (orange)	~36 minutes	~ 1.8e5

60 times reduction

still much room for optimization!

Changing sampler (SMC)



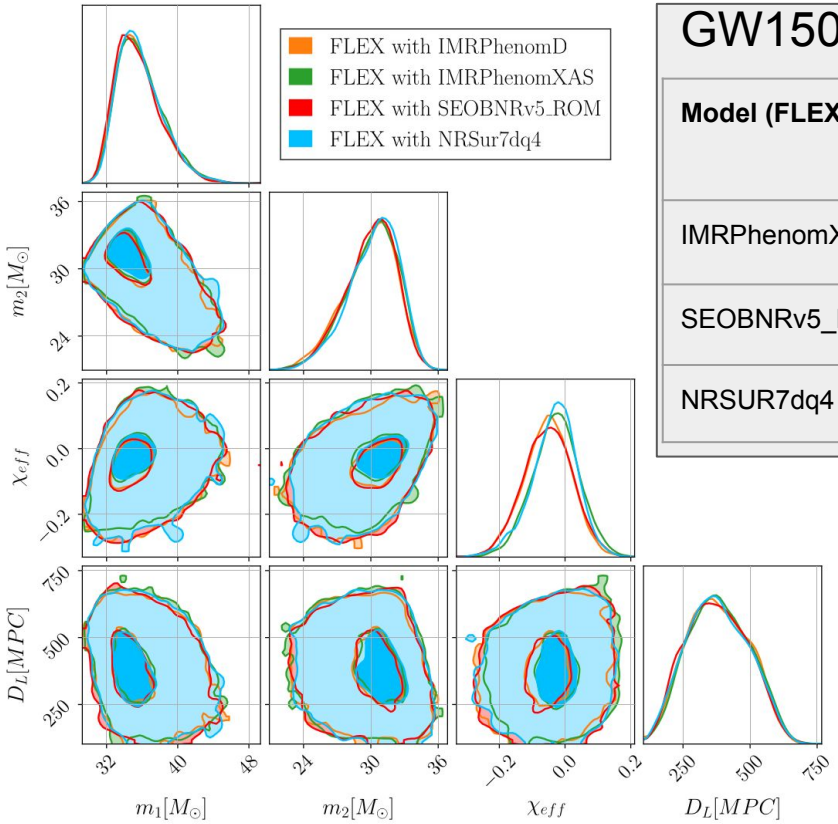
GW150914:

Model	**Log evidence**	**Time**
pocomc vanilla (blue)	286.09	50 mins x 16 cores
pocomc FLEX (orange)	285.06	3 mins x 1 core (after training)

The runs so far have been presented with MCMC, so we do not have access to the evidence.

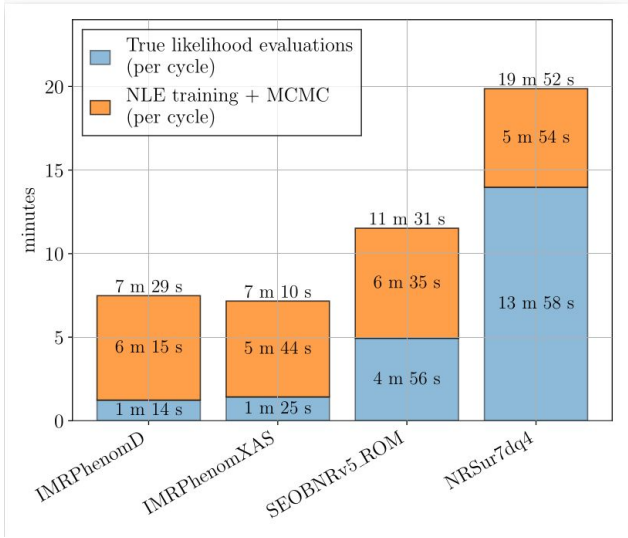
But the FLEX-NLE object, once trained can be used with other samplers afterwards, like an SMC one

Results: consistency and timing between waveforms



GW150914:

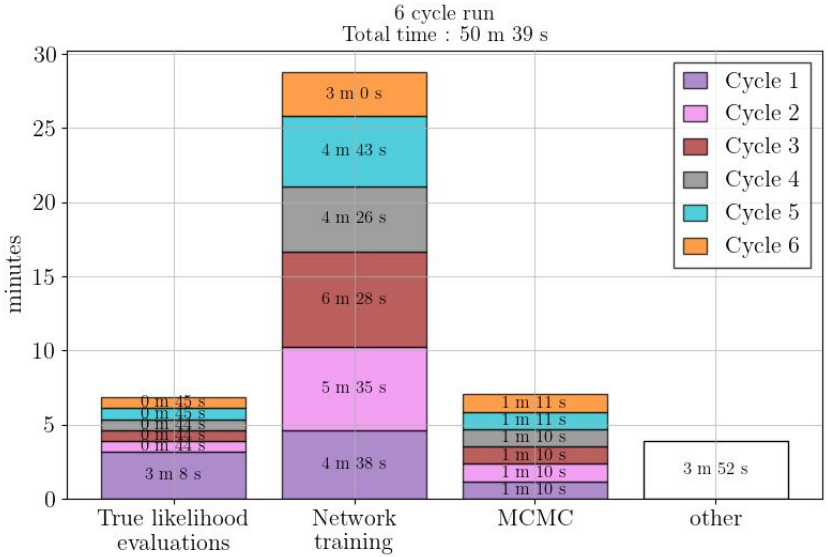
Model (FLEX)	**Log evidence**
IMRPhenomXAS	285.10
SEOBNRv5_ROM	285.05
NRSUR7dq4	284.67



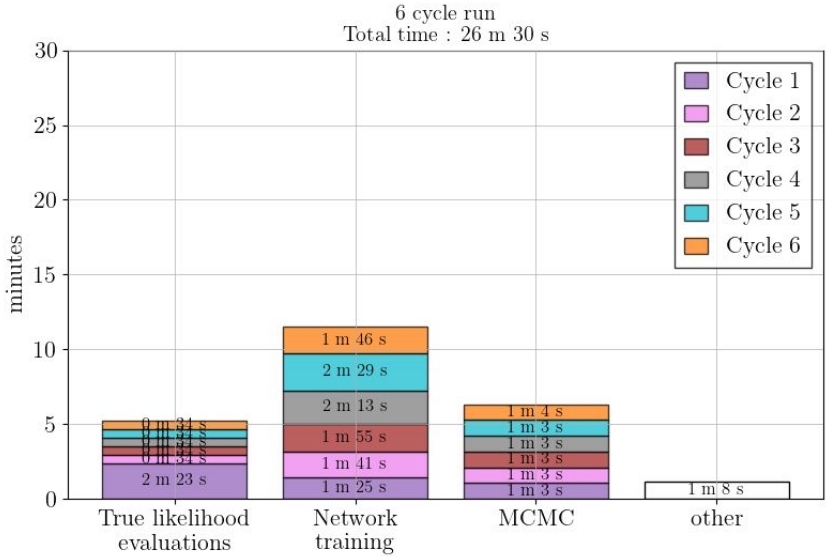
Consistency between waveforms shows that the algorithm is robust

Results : But can we go faster?

Run on CPU:



Run on GPU:



Conclusions

What has been done :

- Neural Likelihood seem to be a viable and reliable option for fast parameter estimation
- Get compatible results with standard PE ~50 times less likelihood evaluation

To Dos 📝

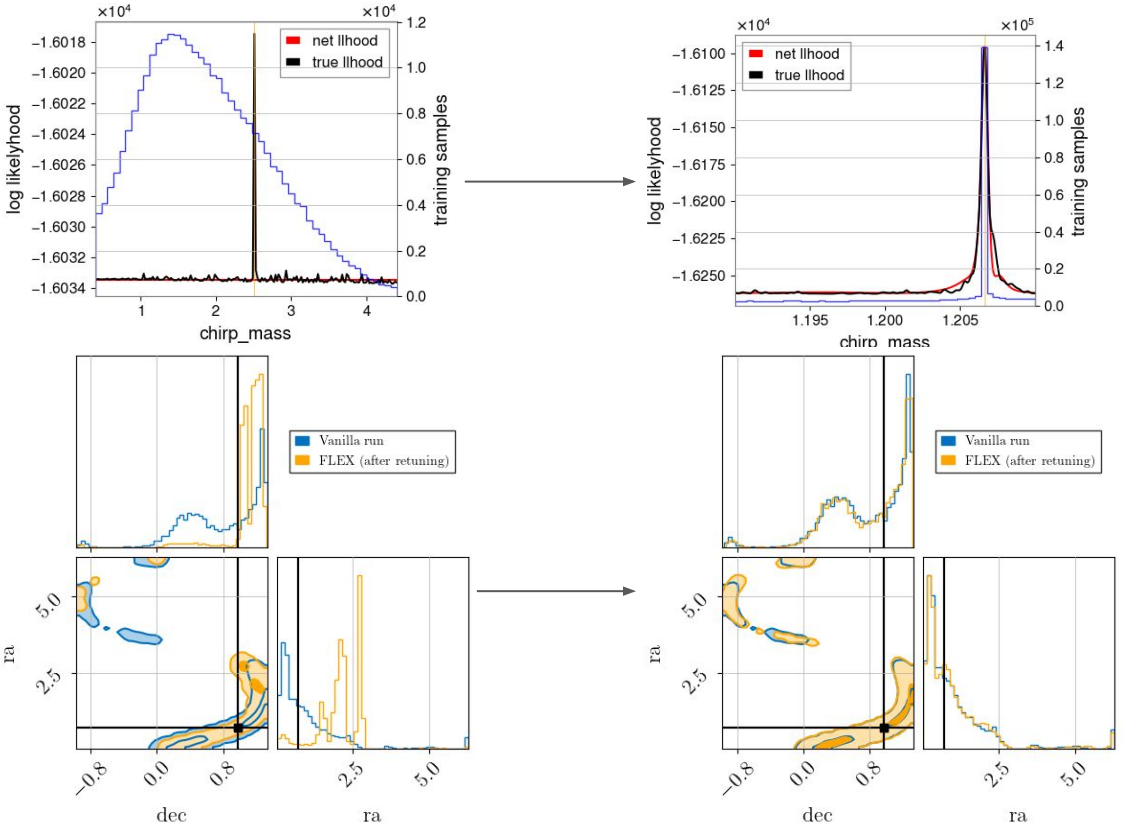
- Expand to different signals (BNSs,3G, precession)
- FLEX can be improved, Possible optimizations:
  - Final PE stage 🏍
  - Training scheme 🏍🏍
  - Initial samples 🏍🏍🏍🏍

Thank you!

Limitations of the algorithm

- 😕 Priors too broad (steep logL)
- 😕 High SNR (hard to find the logL peaks)
- 😕 Multimodalities (mode collapse, mode confusion)

- 🔧 Narrower priors
- 🔧 More training samples (costly …)
- 🔧 Smarter generation of initial samples
- 🔧 Better parametrizations



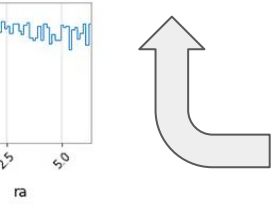
Phase 1 : Annealed KDE

Using weighted Kernel Density Estimation (KDE) allows us to get progressively less-awful approximations of the posterior

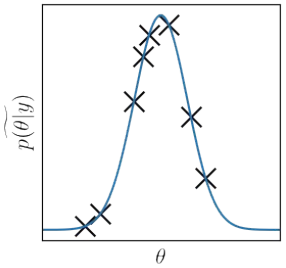
1. Assign weights to each sample with softmax

$$w_i = \frac{\exp(\mathcal{L}_i/T)}{\Sigma_{j=0}^{N}\exp(\mathcal{L}_i/T)}$$

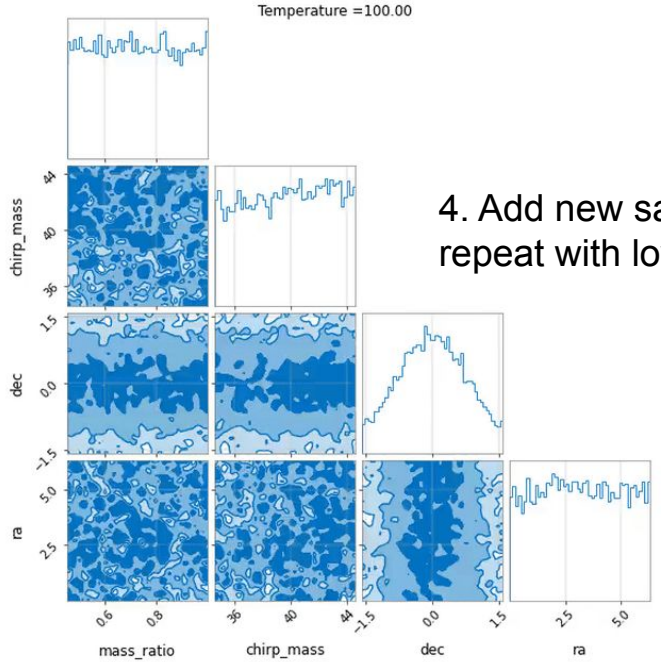
2. Train a KDE on $\{\theta_i\}$ with weights $\{w_i\}$



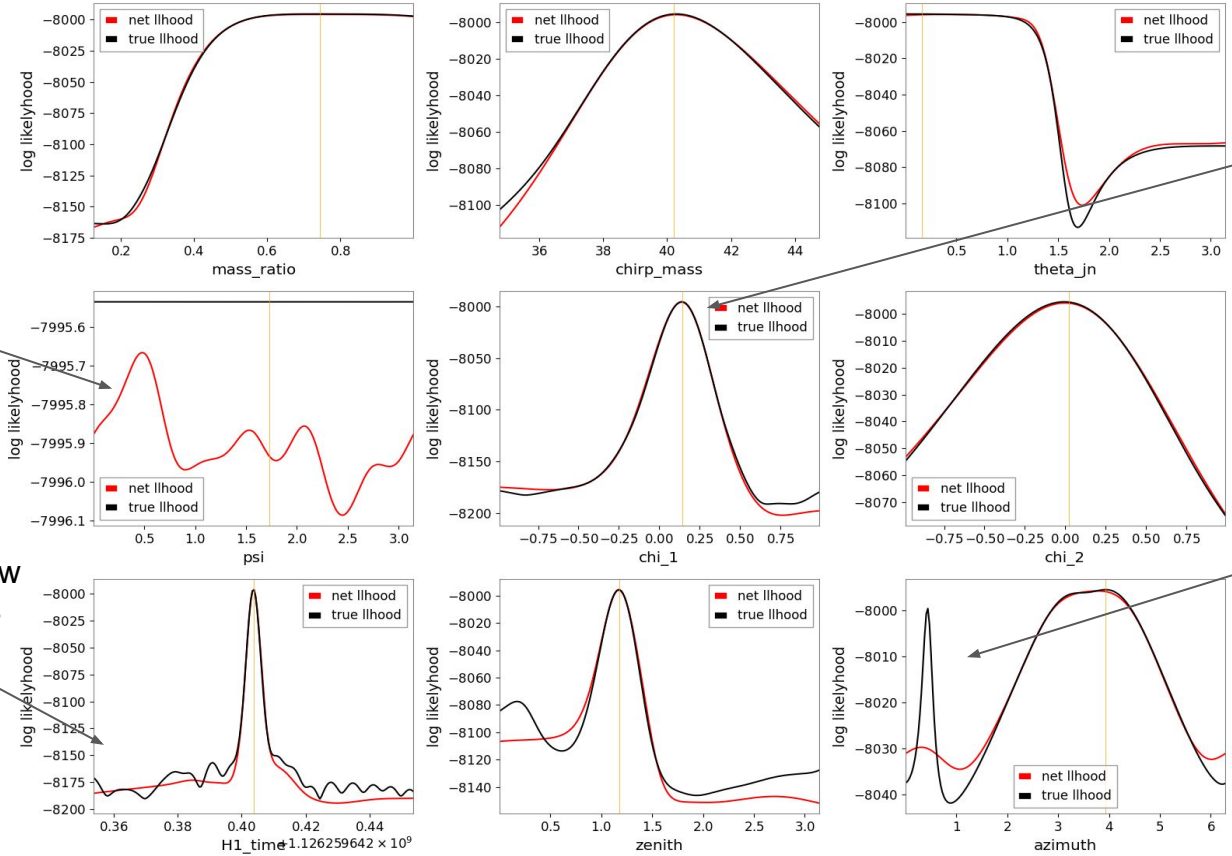
3. Take K new samples from the KDE and calculate their likelihood



4. Add new samples to $\{\theta_i\}$ and repeat with lower T



Backup - Check NN performance.



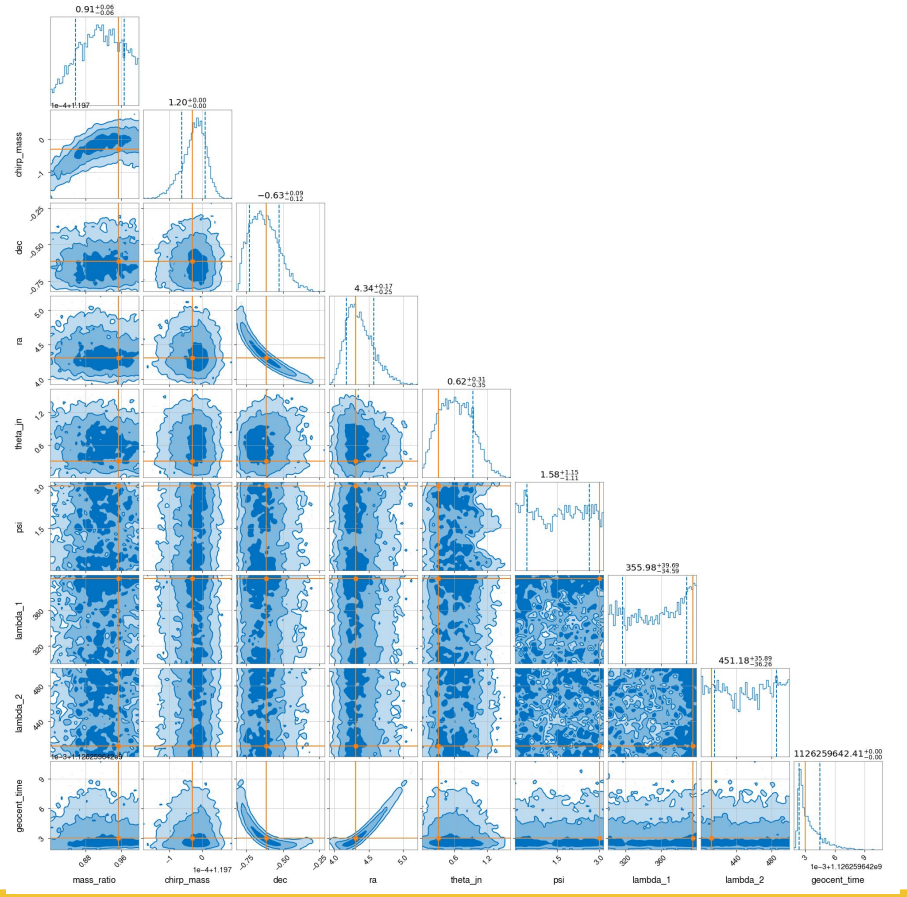
❌ What is even happening here

✅ **Very accurate in high likelihood regions!** (apparently this is enough to guarantee convergence!)

❌ Inaccurate in low LLhood regions

❌ Misses one peak

Backup - BNSs


mass_ratio
chirp_mass
dec
ra
theta_jn
psi
lambda_1
lambda_2
geocent_time

Differential evolution initial samples