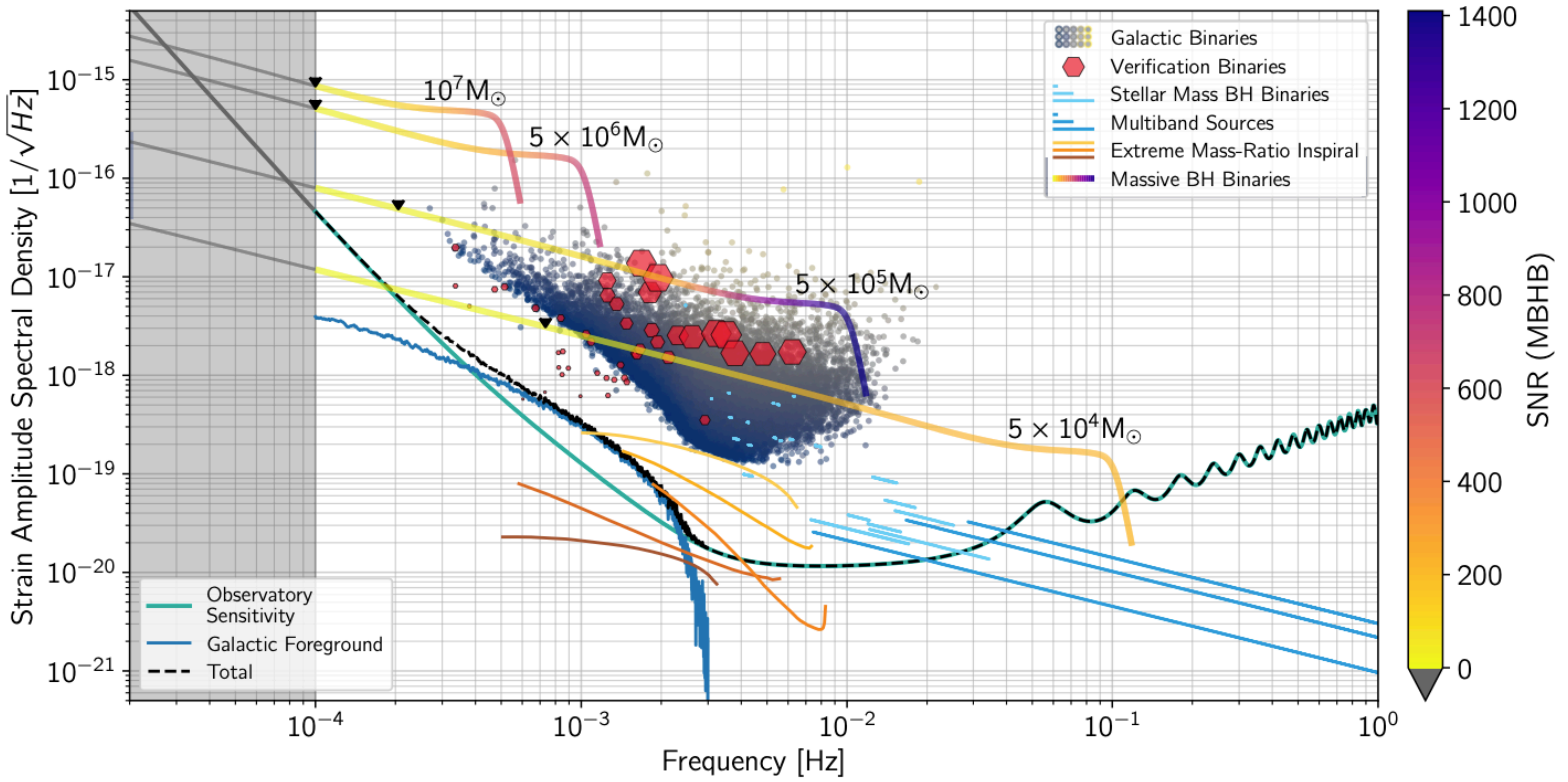
Sequential simulation-based inference for extreme mass ratio inspirals

with James Alvey, Lorenzo Speri, Christoph Weniger, Uddipta Bhardwaj, Davide Gerosa and Gianfranco Bertone

Sequential simulation-based inference for extreme mass ratio inspirals
arXiv:2505.16795

Pippa Cole, University of Milan-Bicocca

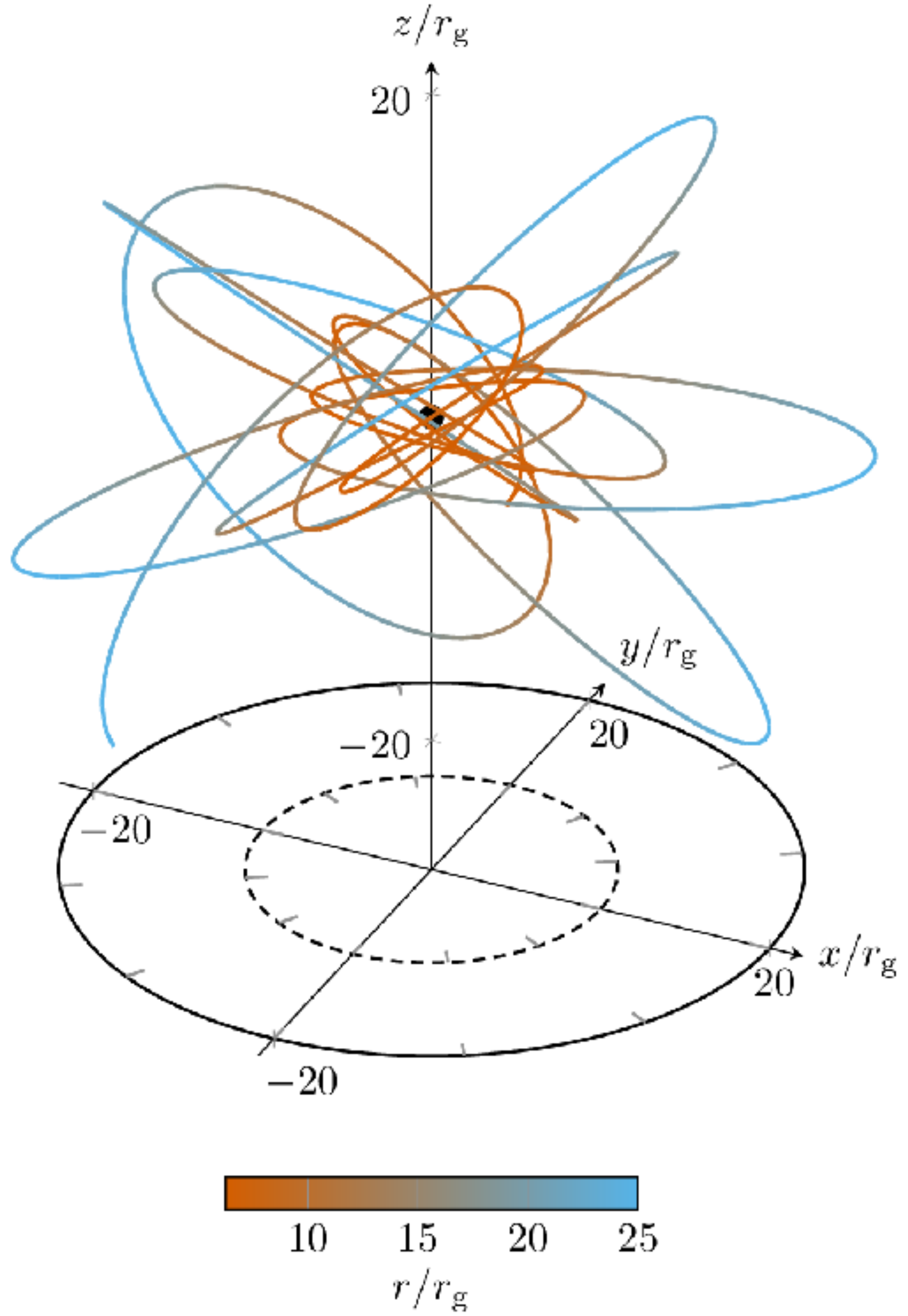
EMRIs in the milliHertz frequency band



LISA Red Book: arXiv:2402.07571

What's special about extreme mass ratio inspirals?

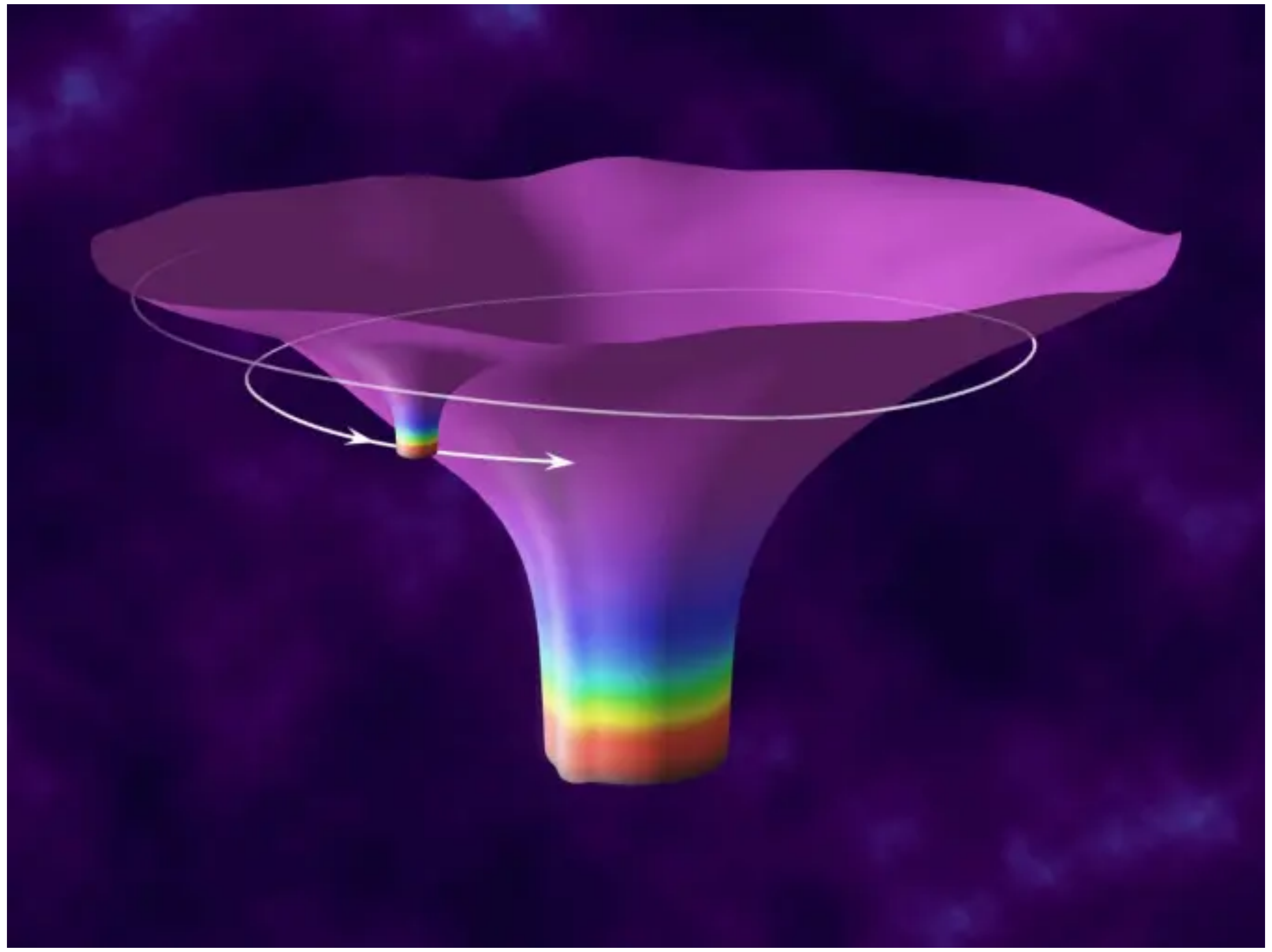
- A binary black hole system with mass ratio $q = m_2/m_1 \leq 10^{-4}$
- Rich dynamics due to eccentric orbits, spin precession and thousands of excited harmonics
- Long duration signals - could remain in band for years - opportunity to observe millions of cycles



Berry et al. 2019

Many astrophysics and fundamental physics opportunities

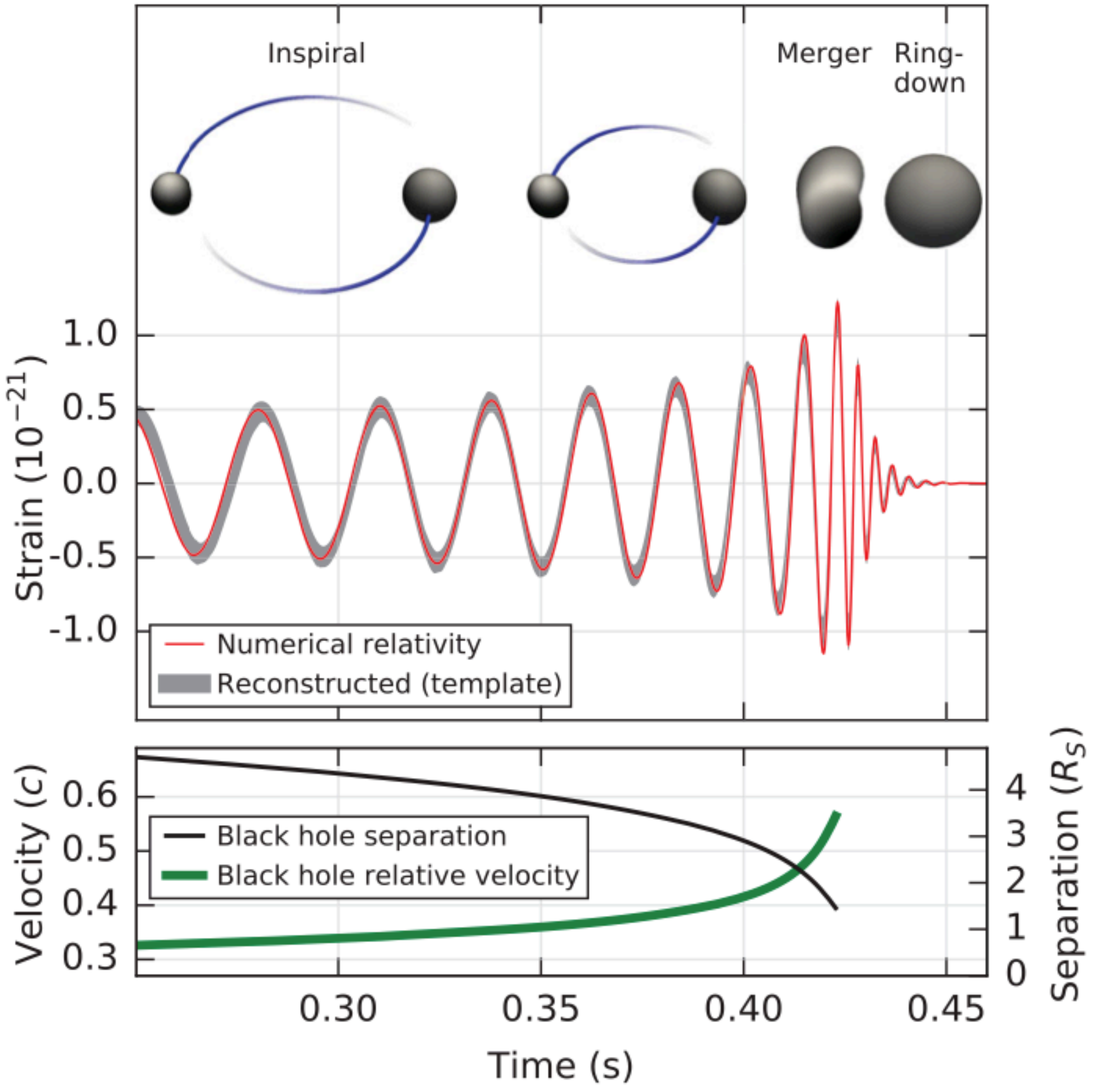
- Formation of intermediate mass black holes (Volonteri 2010)
- Environmental effects - dark matter, ultra-light bosons, accretion disks around primary object (Eda et al. 2013, Macedo et al. 2013, Kavanagh et al. 2020, Khalvati et al. 2024, Baumann et al. 2022, Barsanti et al. 2023, Zhang et al. 2023, Tomaselli et al. 2024, Cole et al. 2023, Speri et al. 2023, Duque et al. 2024, Coppaoni et al. 2025)
- Tests of General Relativity (Han & Chen 2019, Gupta et al. 2022, Speri et al. 2024, Kejriwal et al. 2024, A. Cardenas-Avendano & Sopuerta 2024)



Credit: NASA

Extreme mass ratio inspirals have complicated waveforms that evolve significantly in time

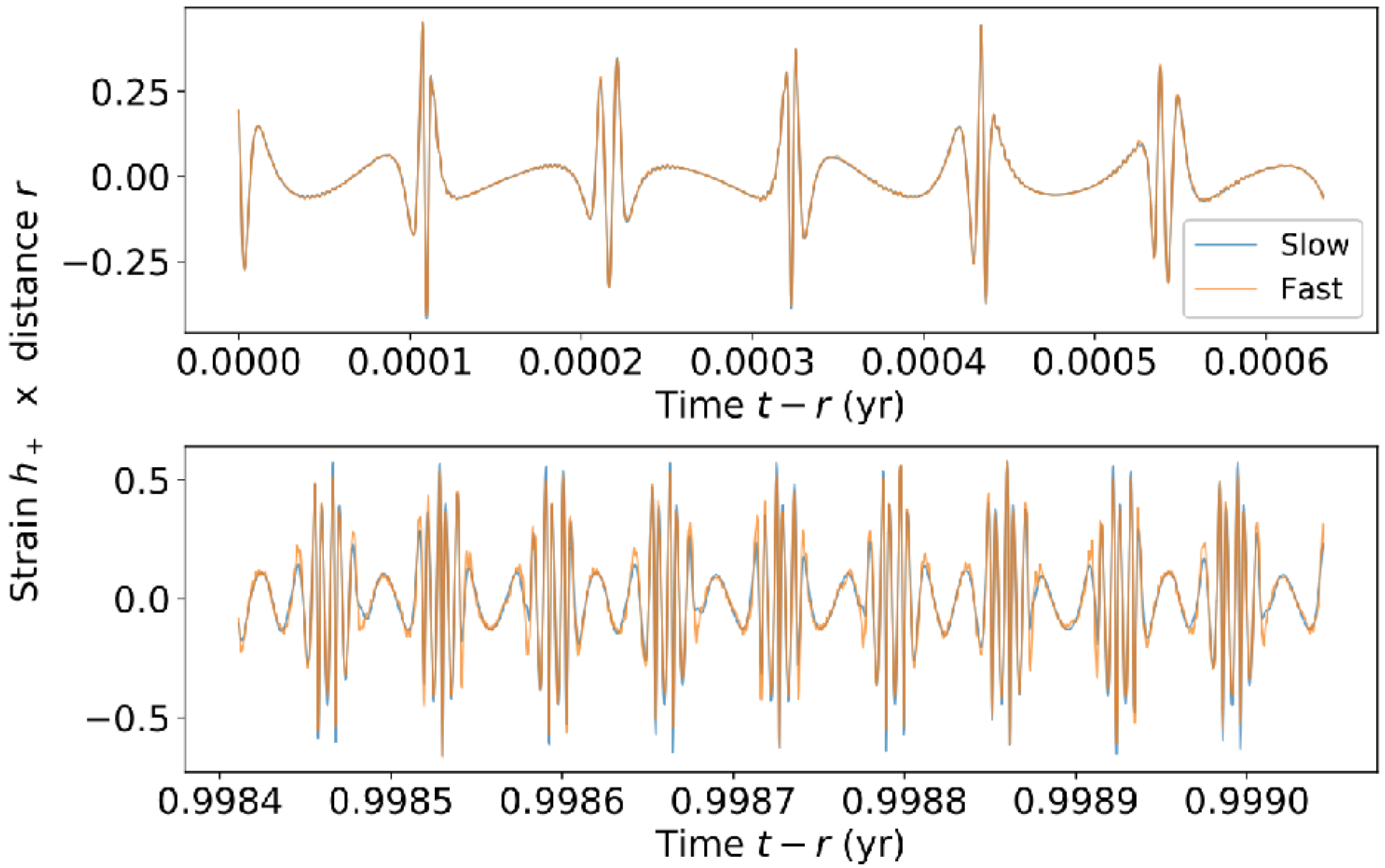
Equal stellar-mass



Abbott et al. arXiv:1602.03837

EMRIs

One year before 'plunge'



Just before 'plunge'

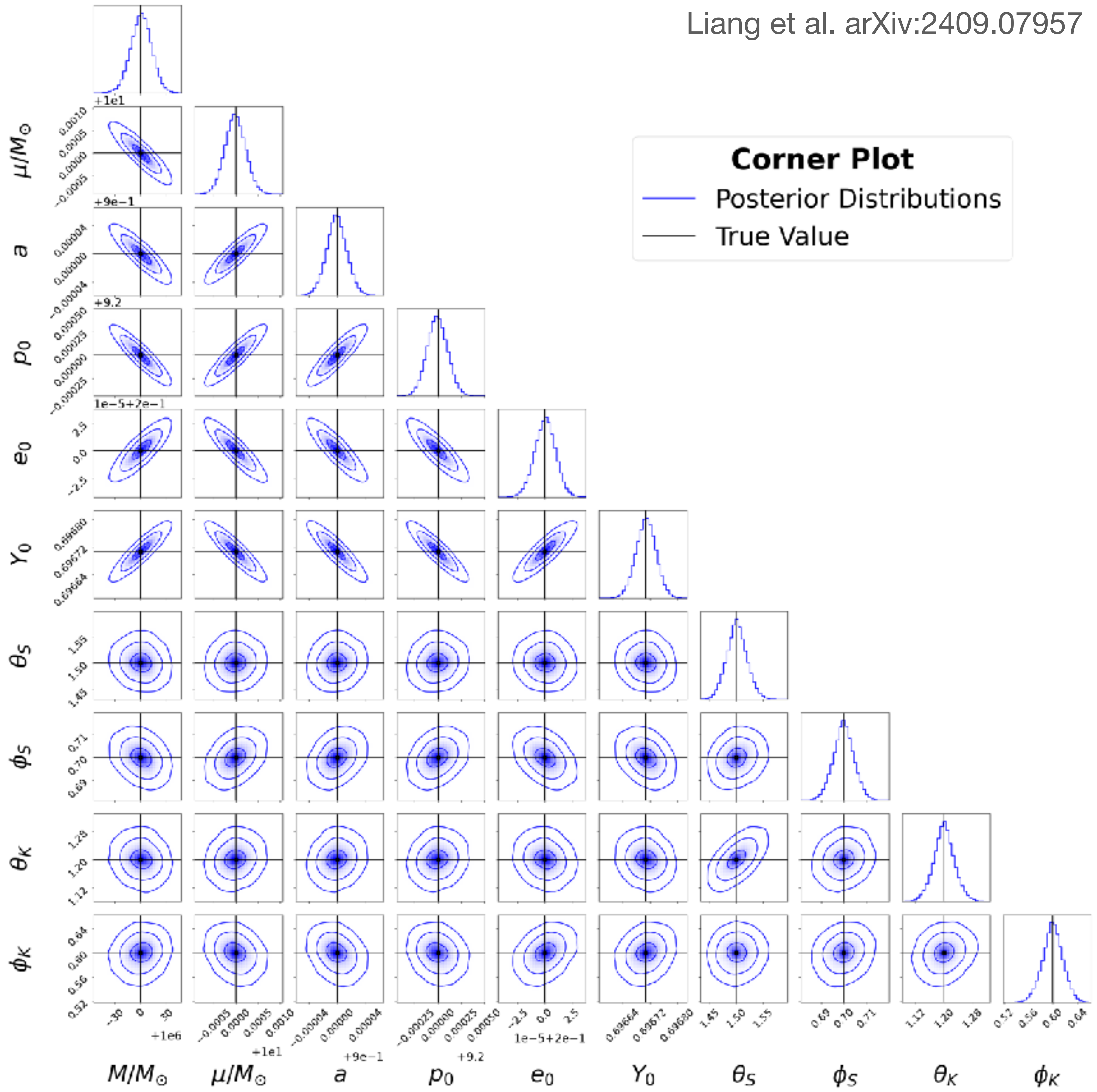
Chua et al. arXiv:2008.06071

Potential for extremely precise parameter measurements

Liang et al. arXiv:2409.07957

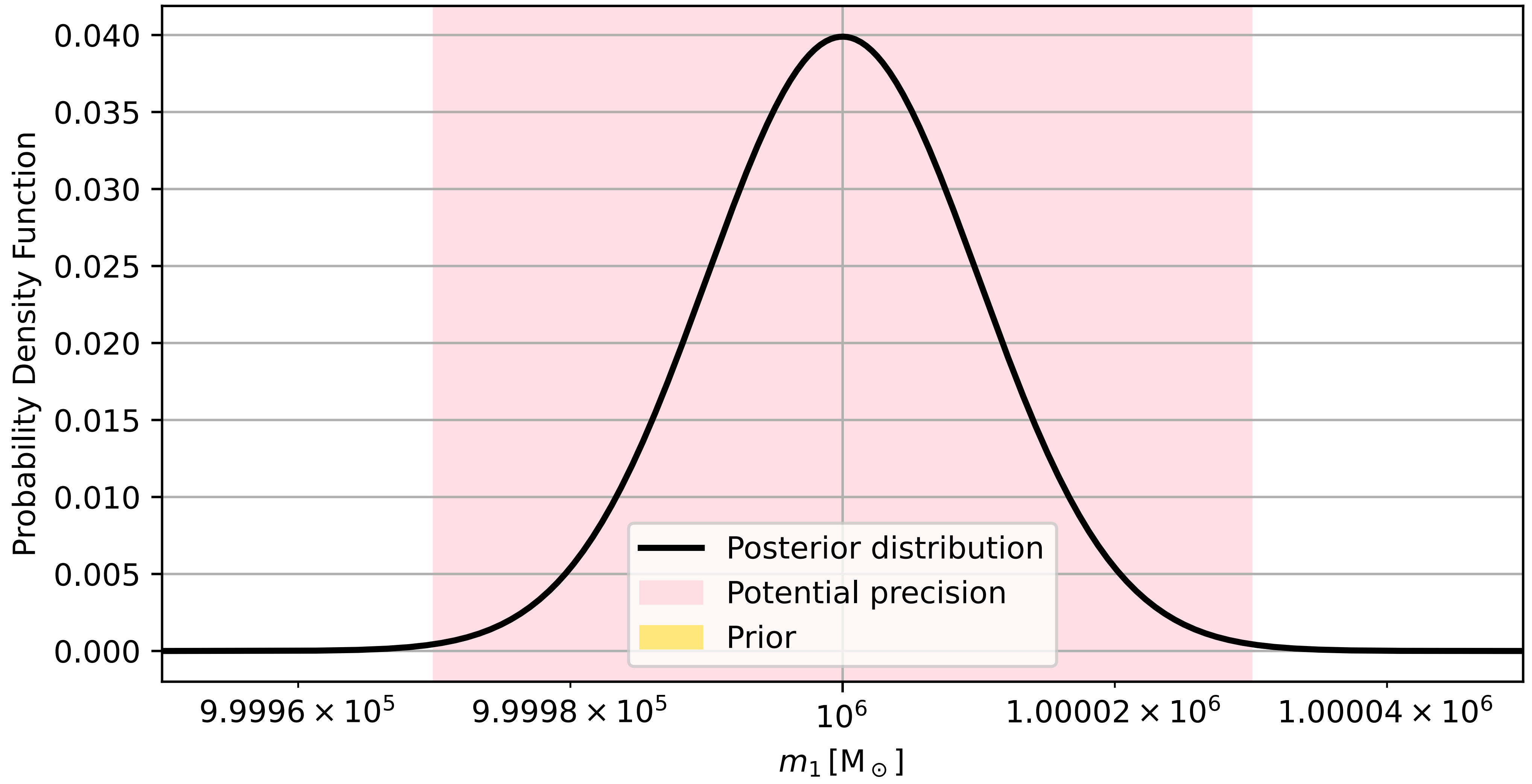
- Some parameters can theoretically be measured to a precision of e.g. 0.01% (eccentricity), 0.003% (primary mass), 0.005% (secondary mass)

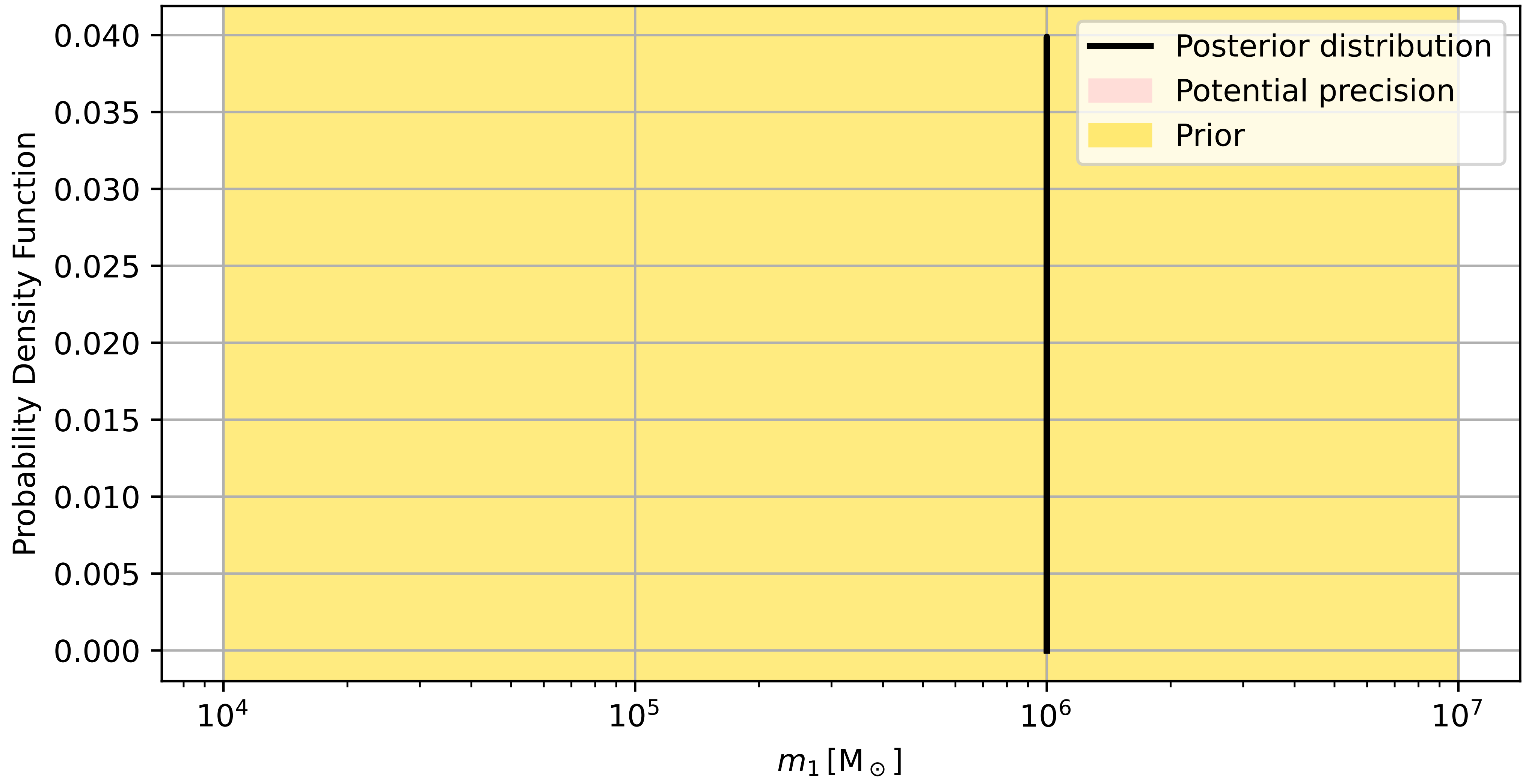
- Example of MCMC run initiated very close to the true injected values



Vast and multi-modal parameter space

- However the EMRI parameter space is vast, making search and parameter estimation strategies extremely difficult
- There are also many degeneracies between parameters, so it is highly multi-modal





How and why sequential simulation-based inference might help

1. Simulation-efficient way to narrow down a vast parameter space (important because EMRI simulations are also relatively expensive)

2. Future goal will be to cope with non-stationary, non-Gaussian noise (as well as many overlapping sources), where likelihood-based methods tend to become expensive or unfeasible

Sequential simulation-based inference

Simulate -> Train network -> Estimate posterior -> Truncate prior?

Yes

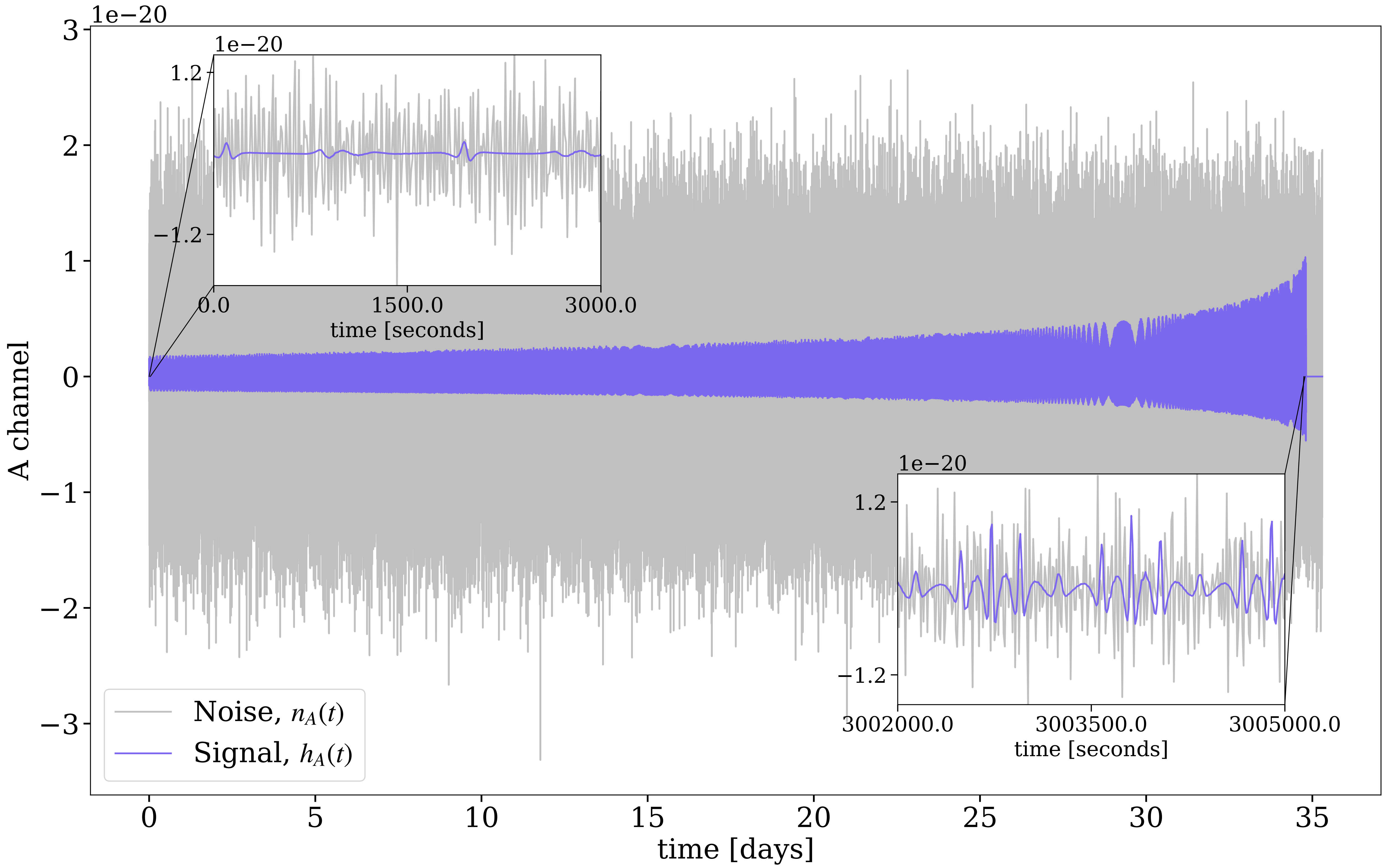
No

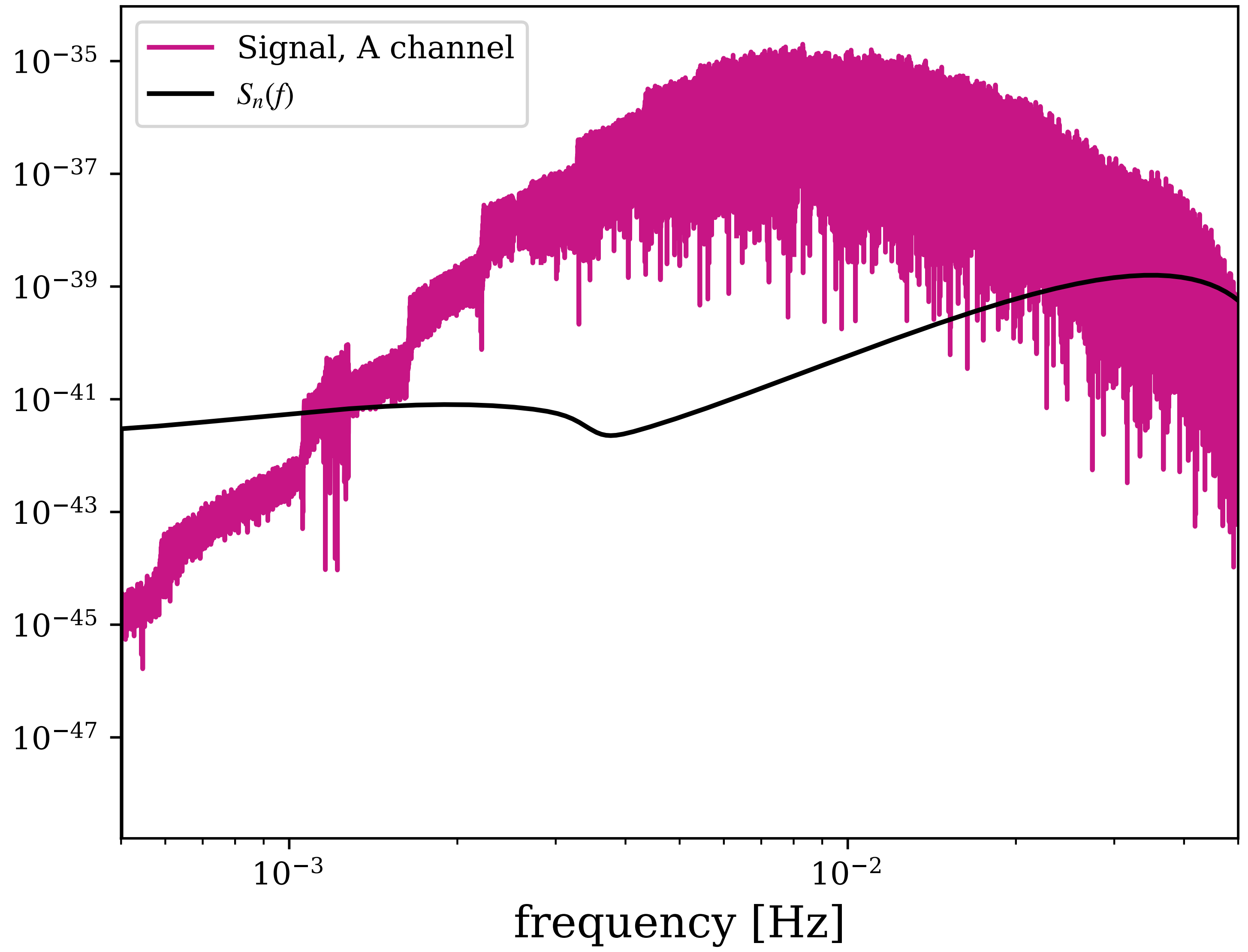
Convergence

Bhardwaj et al. 2023

Simulation set-up

- Signal simulated with Fast EMRI Waveforms (`FEW`) code
- Fed through `fastlisaresponse` which implements the LISA detector response in time domain and outputs the signal projected onto the ‘A’ and ‘E’ time delay interferometry channels
- Add noise sampled from the power spectral density as computed by `pycbc` (analytical including confusion noise)





$dt = 10\,\mathrm{s}, T_{obs} = 0.1\,\mathrm{yr}, m_1 = 5.385 \times 10^5\,\mathrm{M}_\odot, m_2 = 50.55\,\mathrm{M}_\odot, p_0 = 10.35\; e_0 = 0.2927,\ \cos\theta_K = 0.0384, \phi_K = 5.212, d_L = 232.8\,\mathrm{Mpc}, \Phi_\phi = 2.966, \Phi_r = 2.014,\ \cos\theta_S = 0.4107, \phi_S = 3.124$

PEREGRINE
Gravitational Wave Parameter Inference with Neural Ratio Estimation

Training set-up

PEREGRINE-style approach - Truncated Marginal Neural Ratio Estimation

- Train a neural network to recognise whether parameter vector θ_k and signal x are drawn jointly $p(x, \theta_k)$ or marginally $p(x)p(\theta_k)$ - binary classification

- Objective to minimise the binary-cross entropy loss function:

$$\mathscr{L}[\hat{\rho}_{k,\phi}] = -\int \left\{ p(x, \theta_k) \ln \sigma(\hat{\rho}_{k,\phi}(x, \theta_k)) + p(x)p(\theta_k) \ln \left[1 - \sigma(\hat{\rho}_{k,\phi}(x, \theta_k)) \right] \right\} dx \, d\theta_k \, .$$

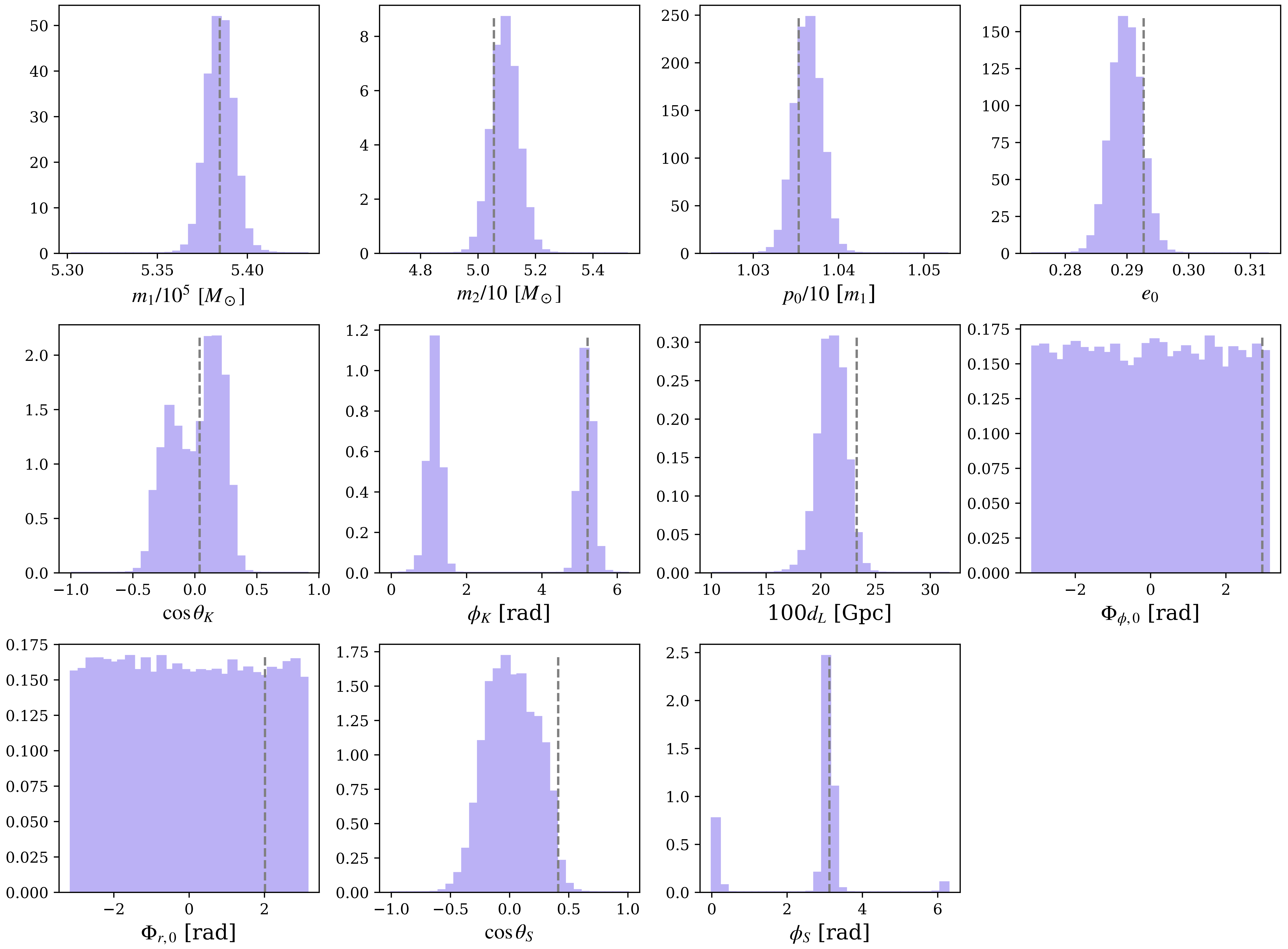
- Optimal classifier $\hat{\rho}_{k,\phi}(x, \theta_k)$ is the log of the likelihood-to-evidence ratio which can be re-weighted by prior samples to estimate the posterior

$$r_k(\boldsymbol{x} \mid \boldsymbol{\vartheta}_k) := \frac{\overset{\text{Likelihood}}{p(\boldsymbol{x} \mid \boldsymbol{\vartheta}_k)}}{\underset{\text{Evidence}}{p(\boldsymbol{x})}} = \frac{\overset{\text{Jointly drawn samples}}{p(\boldsymbol{x}, \boldsymbol{\vartheta}_k)}}{\underset{\text{Marginally drawn samples}}{p(\boldsymbol{x})p(\boldsymbol{\vartheta}_k)}} = \frac{\overset{\text{Posterior}}{p(\boldsymbol{\vartheta}_k \mid \boldsymbol{x})}}{\underset{\text{Prior}}{p(\boldsymbol{\vartheta}_k)}}$$

Bhardwaj et al. 2023
Miller et al. 2021

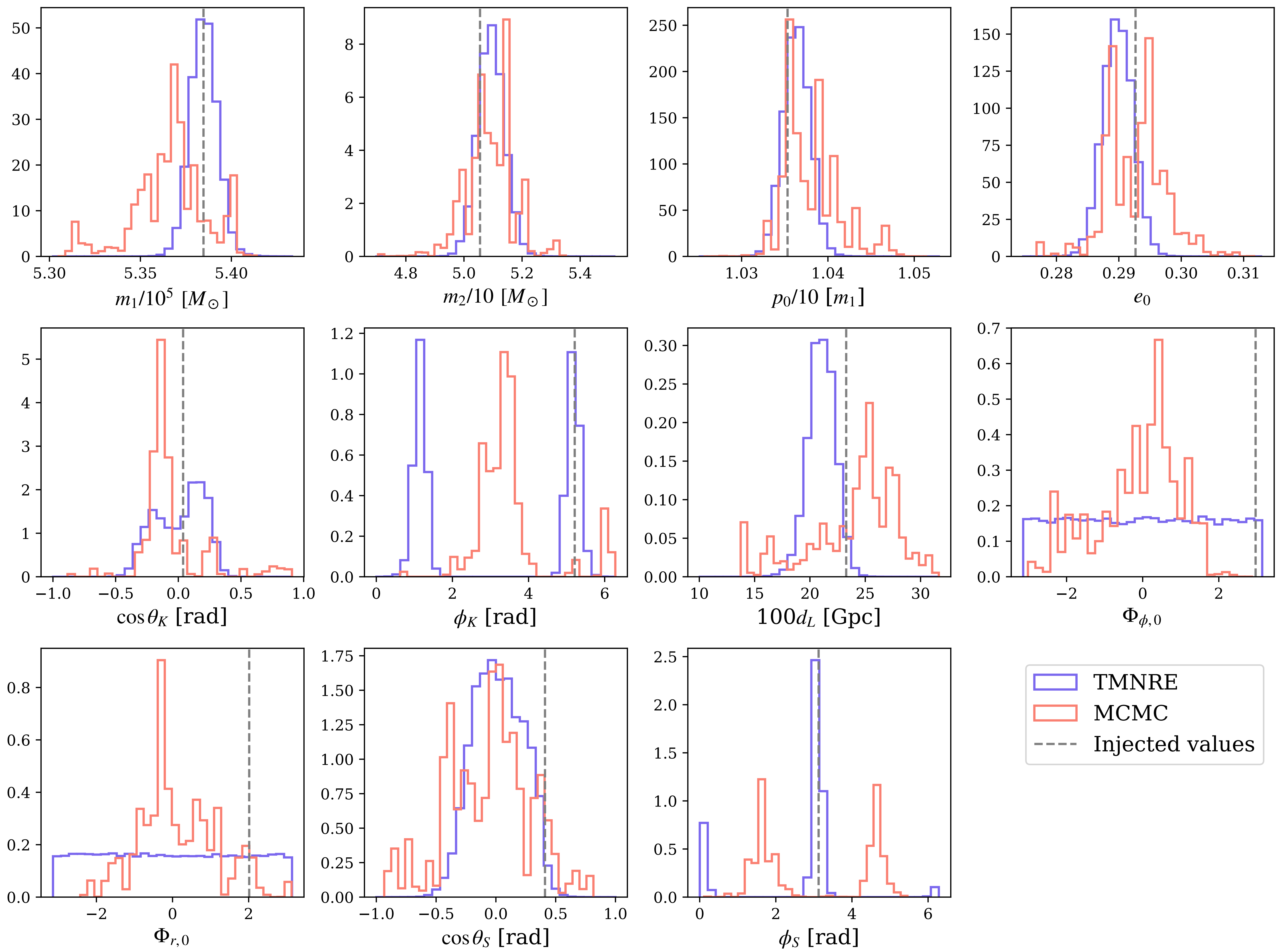
Posteriors/proposal distributions

- 6 sequential rounds, 150K sims each round, approx 12 hours each round
- Injected values all within 2σ credible intervals
- The relative half-widths (2σ credible intervals) are 0.3% for m_1, 2% for m_2, 0.3% for p_0 and 2% for e_0.



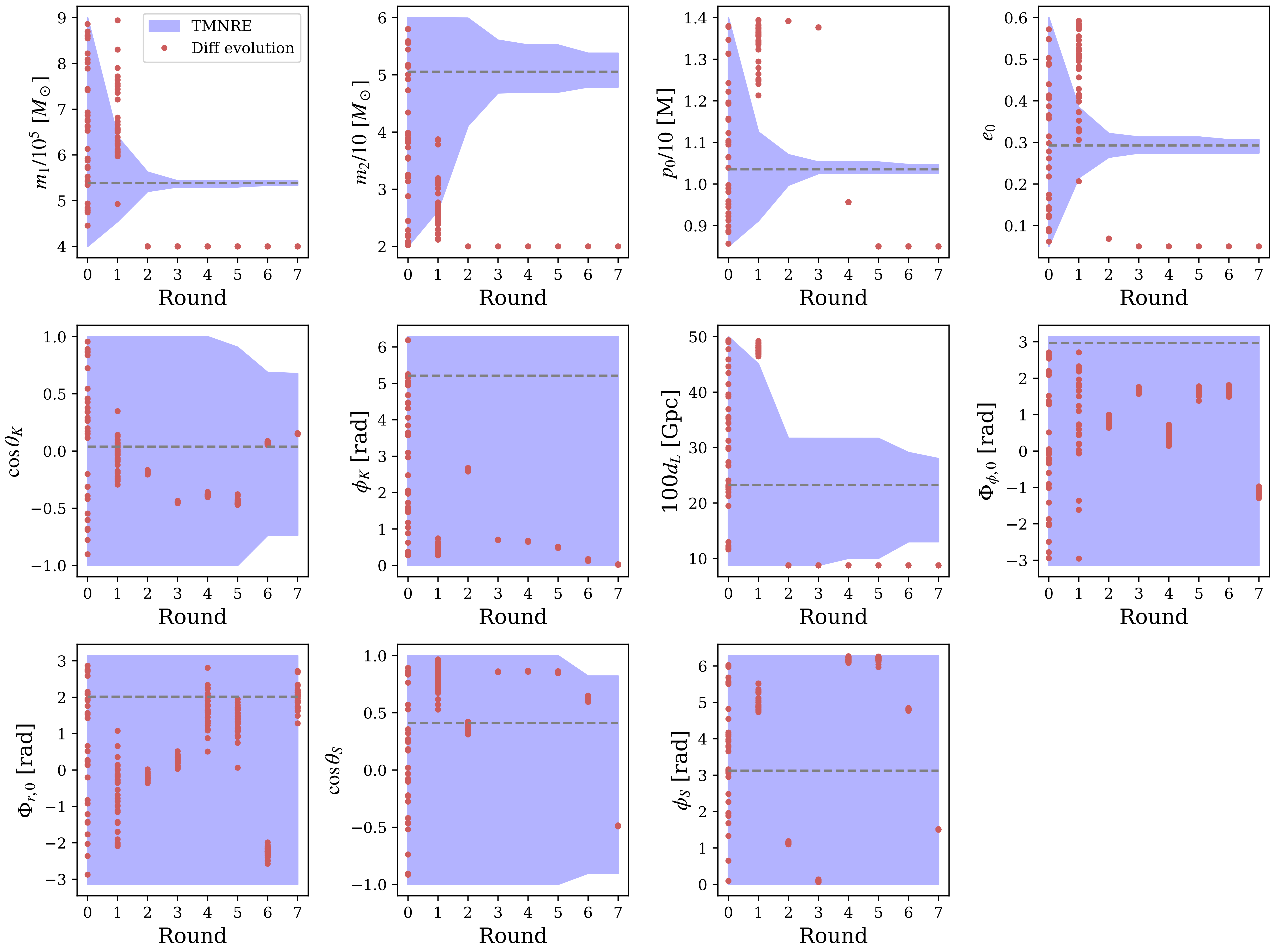
Comparison with MCMC

- MCMC prior chosen to be the 6th round prior identified with TMNRE
- 32 walkers and 4687 steps ~ waveform evaluations approximately 150K
- 2 days wall-clock time
- Chains not converged
- **However** - TMNRE approach does not achieve expected measurement precision



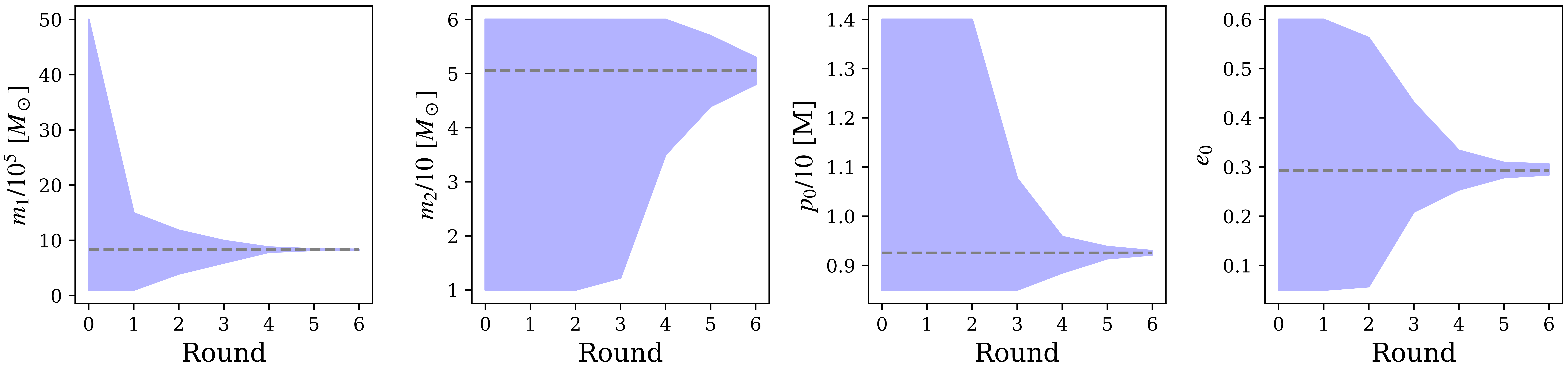
Effectively narrows down parameter space

- Parameter space for intrinsic parameters (top line) narrowed significantly via truncation between rounds
- Proposal distribution volume a million times smaller than prior volume
- Differential evolution (stochastic optimiser) performs poorly



Narrowing down parameter space

- Works particularly well for intrinsic parameters
- Here larger primary mass, and even wider priors



Conclusions

- EMRIs are difficult but rewarding systems to study
- TMNRE narrows down parameter space (by a factor of a million) very efficiently from wide priors and
- Estimates proposal distributions that are better converged than MCMC with same number of waveform evaluations and same priors
- Improvements required in order to achieve precise parameter measurements - especially important for distinguishing from environmental effects biases
- Eventually tackle non-stationary, non-Gaussian noise and overlapping sources
- How and when do we fold back in environmental effects to the data analysis pipeline?

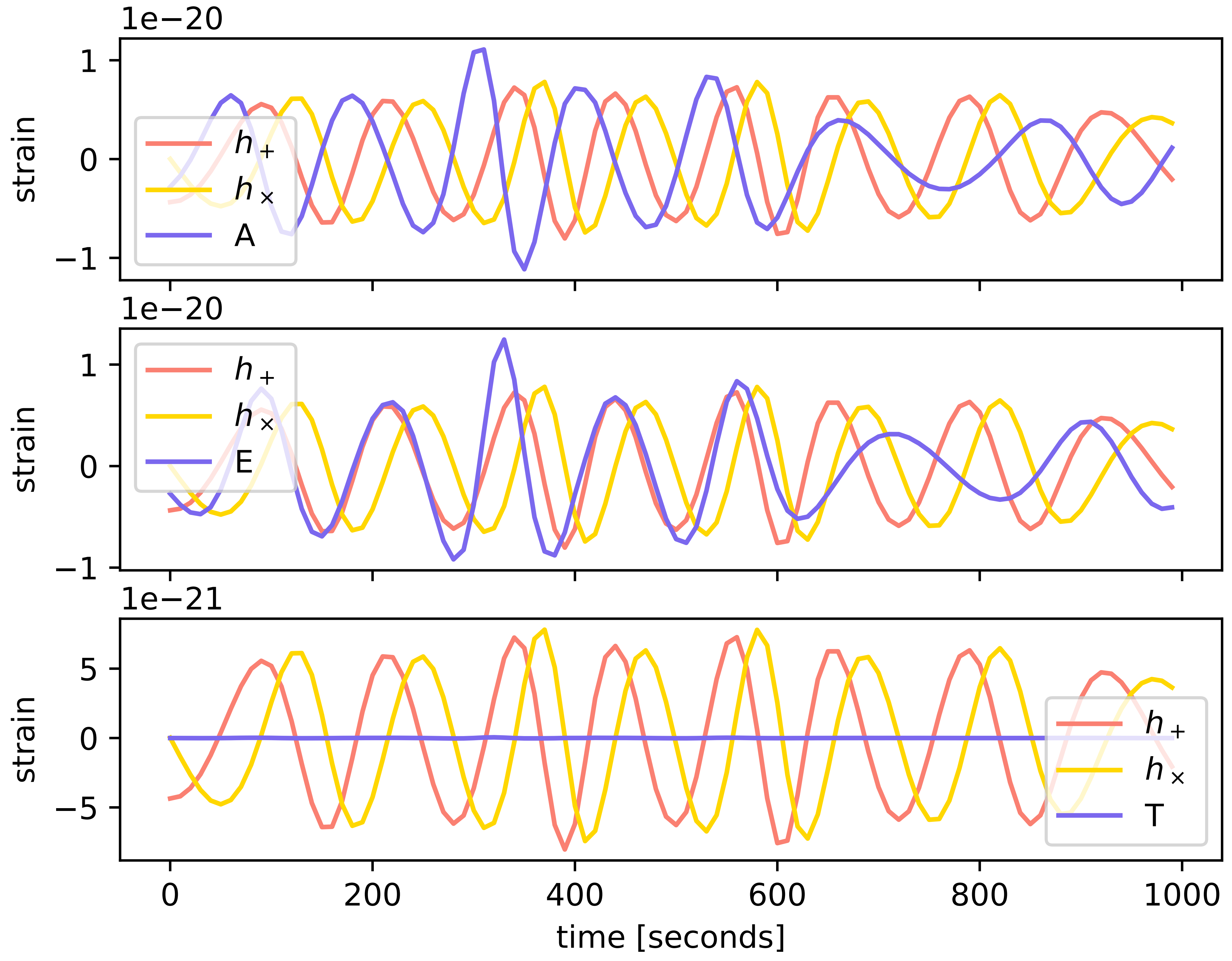
Simulation set-up

- Signal simulated with Fast EMRI Waveforms (`FEW`) code

- Phase and amplitude of each mode computed up to 1st order in gravitational self-force theory (expansion of the metric of the binary in powers of mass ratio).

- Modes summed over to produce adiabatic waveform $h(t) = h_+(t) - ih_\times(t)$ in time domain.

- Fed through `fastlisaresponse` which implements the LISA detector response in time domain and outputs the signal projected onto the 'A', 'E' and 'T' time delay interferometry channels



$dt = 10\,\mathrm{s}, T_{obs} = 0.1\,\mathrm{yr}, m_1 = 5.385 \times 10^5\,\mathrm{M}_\odot, m_2 = 50.55\,\mathrm{M}_\odot, p_0 = 10.35\; e_0 = 0.2927,\ \cos\theta_K = 0.0384, \phi_K = 5.212, d_L = 232.8\,\mathrm{Mpc}, \Phi_\phi = 2.966, \Phi_r = 2.014,\ \cos\theta_S = 0.4107, \phi_S = 3.124$

Training set-up

PEREGRINE-style approach

- 150K simulations per round
- Batch size = 128
- Initial learning rate = 10^{-4}
- Training:validation - 90:10
- Early stopping criterion: 7 epochs
- Utilise noise shuffling
- Bounding threshold = 10^{-5}
- Unet -> Linear compression -> Logratio estimator: input 16 features, 11 parameters, dropout = 0.1

PEREGRINE

Gravitational Wave Parameter Inference with Neural Ratio Estimation

Bhardwaj et al. 2023
Miller et al. 2021

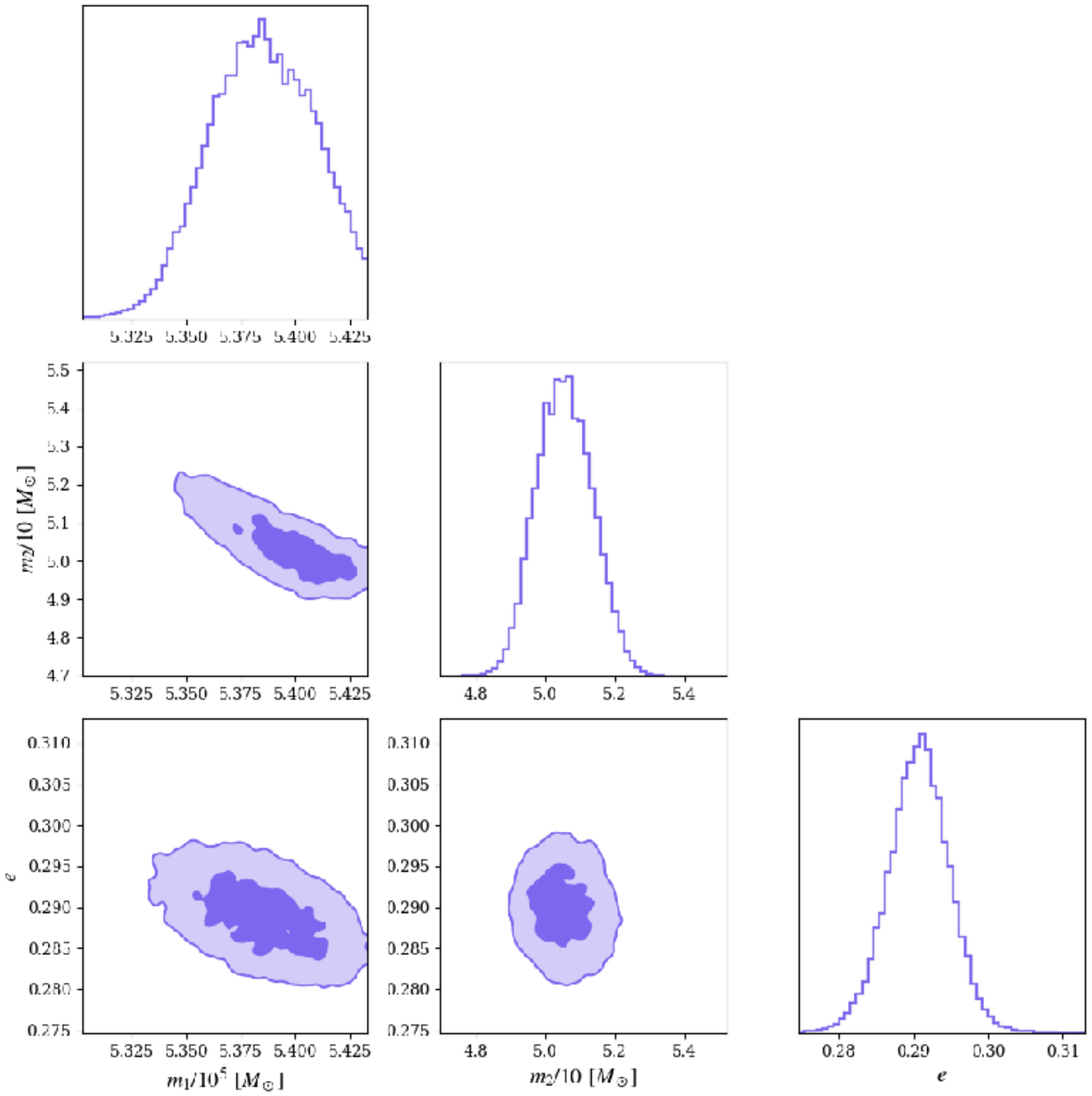
Train objective to minimise the binary-cross entropy loss function

$$\mathscr{L}[\hat{\rho}_{k,\phi}] = -\int \left\{ p(x,\theta_k) \ln \sigma(\hat{\rho}_{k,\phi}(x,\theta_k)) + p(x)p(\theta_k) \ln \left[1 - \sigma(\hat{\rho}_{k,\phi}(x,\theta_k)) \right] \right\} dx\, d\theta_k \,.$$

$$r_k(\boldsymbol{x} \mid \boldsymbol{\vartheta}_k) := \frac{\overset{\text{Likelihood}}{p(\boldsymbol{x} \mid \boldsymbol{\vartheta}_k)}}{\underset{\text{Evidence}}{p(\boldsymbol{x})}} = \frac{\overset{\text{Jointly drawn samples}}{p(\boldsymbol{x}, \boldsymbol{\vartheta}_k)}}{\underset{\text{Marginally drawn samples}}{p(\boldsymbol{x})p(\boldsymbol{\vartheta}_k)}} = \frac{\overset{\text{Posterior}}{p(\boldsymbol{\vartheta}_k \mid \boldsymbol{x})}}{\underset{\text{Prior}}{p(\boldsymbol{\vartheta}_k)}}$$

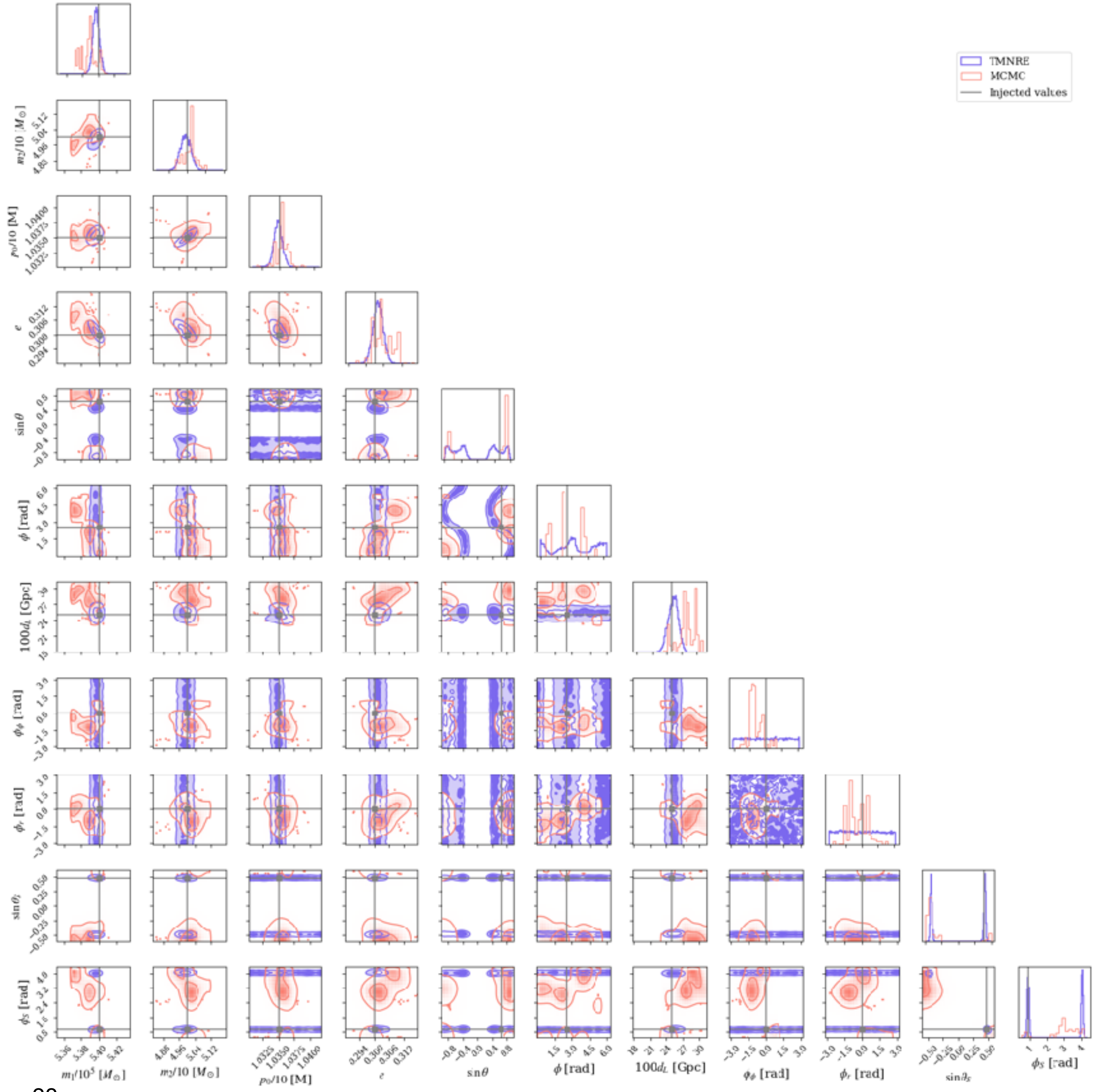
2d issues with TMNRE

- Intrinsic parameters consistent…



2d issues with TMNRE

- Some other 2d marginal distributions inconsistent with 1d distributions



Unet

```python
self.unet_f = Unet(
    n_in_channels=2,
    n_out_channels=1,
    sizes=(16, 32, 64, 128, 256),
    down_sampling=(8,8,4,2),
)
```

```python
class Unet(nn.Module):
    def __init__(
        self,
  n_in_channels,
        n_out_channels,
        sizes=(16, 32, 64, 128, 256),
        down_sampling=(2, 2, 2, 2),
    ):
       super(Unet, self).__init__()
        self.inc = DoubleConv(n_in_channels, sizes[0])
        self.down1 = Down(sizes[0], sizes[1], down_sampling[0])
        self.down2 = Down(sizes[1], sizes[2], down_sampling[1])
        self.down3 = Down(sizes[2], sizes[3], down_sampling[2])
        self.down4 = Down(sizes[3], sizes[4], down_sampling[3])
        self.up1 = Up(sizes[4], sizes[3])
        self.up2 = Up(sizes[3], sizes[2])
        self.up3 = Up(sizes[2], sizes[1])
        self.up4 = Up(sizes[1], sizes[0])
        self.outc = OutConv(sizes[0], n_out_channels)

    def forward(self, x):
        x = x.float() # required for resp
        x1 = self.inc(x)
        x2 = self.down1(x1)
        x3 = self.down2(x2)
        x4 = self.down3(x3)
        x5 = self.down4(x4)
        x = self.up1(x5, x4)
        x = self.up2(x, x3)
        x = self.up3(x, x2)
        x = self.up4(x, x1)
        f = self.outc(x)
        return f
```

Linear compression

```
class LinearCompression(nn.Module):
    def __init__(self):
        super(LinearCompression, self).__init__()
        self.sequential = nn.Sequential(
            nn.LazyLinear(1024),
            nn.BatchNorm1d(1024),
            nn.ReLU(),
            nn.LazyLinear(256),
            nn.BatchNorm1d(256),
            nn.ReLU(),
            nn.LazyLinear(176),
            nn.BatchNorm1d(176),
        )

    def forward(self, x):
        return self.sequential(x)
```

Logratios estimator

```
self.logratios_1d = sl.LogRatioEstimator_1dim(
    num_features=16, num_params=int(self.num_params), dropout=0.1, varnames="z_total"
)
```