Annual Progression Review

Cecilia Maria Fabbri

1 Steptember 2025

CONTENTS

LITERATURE REVIEW

Gravitational waves (GWs) are perturbations of spacetime generated by mass distributions with non-null second derivative of the quadrupole mass-moment. Binary systems of stellar objects are exemplary GW emitters and the more compact the object, the bigger the wave produced, making the system more suitable for detection.

The LIGO-Virgo-KAGRA (LVK) collaboration uses a system of interferometers to detect gravitational waves. The latest complete catalogue published by the LVK collaboration is the third gravitational-wave transient catalogue (GWTC-3) and contains about a hundred signals generated by the coalescence of binaries of neutron stars (NSs) and stellar-mass black holes (BHs).

When LVK detect an astrophysical signal, they analyze the data to infer the properties of the event, such as the masses of the objects in the binary, their spins, the distance from the observer and its sky localization. The statistical framework used for individual event analysis is Bayesian inference. With the number of events growing, it becomes crucial to do population analyses, as they allow to set further constraints on the astrophysics underlying stellar objects. To do so, one uses Hierarchical Bayesian statistics.

The models used have multi-dimensional parameters, resulting in probability distributions computationally prohebitive. Traditional methods rely on stochastic sampling methods, such as Markov Chain Monte Carlo (MCMC) or Nested sampling, to infer the probability distribution. However powerful, these methods are expensive, as probability distributions have to be evaluated iteratively million of times for each individual event analysis. Each evaluation requires to generate a waveform on the fly, which heavily contributes to the overall costs. Machine learning approaches have the advantage of moving most of the costs in the training stage, dramatically speeding up inference time, which is crucial for multimessanger astronomy. Several attempts have

been made in this direction, from architectures that model the noise more realistically than traditional methods, to architectures that employ complex physical models without an analytical expression, which is unfeasible with stochastic samplers.

0.1 BAYESIAN STATISTICS

Let us refer to the vector of binary parameters as θ. It consists of intrinsic parameters, such as the masses and the spins, and extrinsic parameters, such as the redshift and the binary orientation. The goal of the analysis is to infer the posterior distribution $p(\theta|d)$, which associates to each point in the binary-parameter space the probability of that binary to generate the observed data d. Observed data is the sum of the astrophysical signal, which is assumed to be deterministic and a function of binary-parameters, and a noise realization in the detector, which is assumed to be stochastic. To infer the posterior one needs a prior $\pi(\theta)$ and a Likelihood $\mathcal{L}(d|\theta)$. The prior is the probability distribution that an event with parameters θ takes place in the Universe. A possible approach to build is based on theory or previous experiments . Nevertheless, the preferred approach is often conservative, and uninformative priors are used, such as priors with constant probability in the binary-parameters dominium and zero outside. The Likelihood is the probability in the data space, given θ parameters. To evaluate the Likelihood one needs to generate waveforms for θ and a noise model. To build the noise model one often assumes that noise is stationary and Gaussian. Bayes theorem links the posterior to the prior and the Likelihood:

$$p(\theta|d) = \frac{\pi(\theta) \times \mathcal{L}(d|\theta)}{p(d)}. \tag{1}$$

The term $p(d)$ is called evidence, and it is the probability of observing the data d under the assumed model, which is often general relativity. The evidence is defined as

$$p(d) = \int \mathrm{d}\theta \pi(\theta) \times \mathcal{L}(d|\theta). \tag{2}$$

The evidence is a normalization factor and does not influence the posterior shape. For this reason one does not compute it during inference, except when doing model comparison.

0.2 HIERARCHICAL BAYESIAN STATISTICS

The goal of population analysis is to infer how binary parameters are distributed at the population level. To do so, one parametrizes the distribution of binary parameters with the population parameters λ, that will determine the shape of the distribution. Examples of hyperparameters are the slope of the binary black hole (BBH) mass distribution, the minimum and maximum mass of stellar-mass black holes. The prior on binary parameters is conditioned on the population parameters, and we refer to it as the population model $p_{\rm pop}(\theta|\lambda)$. For the analysis one needs the prior on population parameters, called hyperprior, $\pi(\lambda)$.

The choice of the population model is crucial, and several approaches have been proposed to construct it. The state of the art are parametric models, which are based on easy to evaluate parametric forms built based on a qualitative astrophysical motivation. Other approaches move towards highly flexible models, called semi-parametric or non-parametric models, where the hyperparameters are for instance spline perturbations. These models allow for data-driven fits but the results are difficult to interpret physically. Moving towards astrophysical approaches, other attempts include comparing data with the output of simulations that synthesize binary populations.

In population analysis one needs to take into account selection effects. Since the events have different probabilities to be detected, the observed population is biased. An event is considered detectable if data exceeds some threshold. The detection probability for an event is thus defined as

$$p_{\rm det}(\theta) = \int_{d>treshold} {\rm d}d\, p(d|\theta). \tag{3}$$

Let us refer to the dataset of observed events as d and to individual events as d_i, with $i = 1, ..., N_{\text{obs}}$, where N_{obs} number of observed events. The posterior for the astrophysical distribution, called hyperposterior, is:

$$p(\lambda|d) = \frac{\pi(\lambda)}{p(d)} \prod_{i=1}^{N_{\text{obs}}} \frac{\int \mathrm{d}\theta \mathcal{L}(d_i|\theta) p_{\text{pop}}(\theta|\lambda)}{\int \mathrm{d}\theta p_{\text{pop}}(\theta|\lambda) p_{\text{det}}(\theta)}, \tag{4}$$

with $\mathcal{L}(d_i|\theta)$ the likelihood of individual events in Eq. 1. The term $p(d)$ is the hyperevidence, and it is the probability of the observed dataset under the assumed population model, i.e.:

$$p(d) = \int \mathrm{d}\lambda' \pi(\lambda') \prod_{i=1}^{N_{\text{obs}}} \mathcal{L}(d_i|\lambda'), \tag{5}$$

with the likelihood respect to the hyperparameters $\mathcal{L}(d_i|\lambda')$ being linked to the individual-event likelihood through

$$\mathcal{L}(d_i|\lambda') = \int \mathrm{d}\theta \mathcal{L}(d_i|\theta) p_{\text{pop}}(\theta|\lambda). \tag{6}$$

The hyperevidence is often computed in population analysis to compare different population models and their goodness in describing the observed dataset.

Population analysis is divided in two steps. First, individual events are analyzed with Bayesian statistics. Then, one uses the individual-event posteriors and the assumed population model to infer the hyperposterior in Eq. 4.

0.3 SIMULATION BASED INFERENCE

Simulation-based inference (SBI) is a deep learning technique to approximate a probability distribution using simulated data. I am working with the DINGO code, which uses SBI to model a posterior. To simulate data, we need to draw samples from the prior and the likelihood. Contrarily to traditional methods, the likelihood is not evaluated and an analytical expression is not necessary. In principle, more complex models can be used, for instance dropping the stationary and gaussian noise assumption or using waveforms from simulations.

Normalizing flows are often used in SBI to approximate complex distributions. A normalizing flow is a mapping from a simple $\pi(u)$ to an approximation of the posterior $q_\phi(\theta|d)$, with ϕ parameters of the mapping. The mapping $f_\phi(u)$ must be invertible and with a simple Jacobian determinant, so that:

$$q_\phi(\theta|d) = \pi(f_\phi^{-1}(\theta)) \left|\det J_{f_\phi}^{-1}\right|. \tag{7}$$

To evaluate $q_\phi(\theta|d)$ one uses Eq. 7. Drawing samples from $q_\phi(\theta|d)$ is equivalent to drawing $u \sim \pi(u)$ and transforming $\theta = f_\phi(u)$. Often one chooses the base distribution $\pi(u)$ is a multivariate standard normal. In the case of DINGO code, the flow f_ϕ is a sequence of transforms $f_\phi^{(i)}$, so that

$$f_\phi^i(u) = \begin{cases} u_i & \text{if } i \leq d/2 \\ c_i(u_i; u_{1:d/2}, d) & \text{if } i > d/2 \end{cases}, \tag{8}$$

so that half of the latent variables u are unchanged while the other half is transformed. At each $f_\phi^{(i)}$ the indices of the u that transform and that do not are permuted. The transformation is the neural spline coupling transform c_i, whose parameters depend on the unchanged u_i and on data strain d.

During training the neural network learns the mapping by learning which ϕ minimize the loss function. Typically in these problems loss functions are built based on the Kullback-Leibler divergence, which is defined as:

$$D_{\mathrm{KL}}(p||q_\phi) = \int \mathrm{d}\theta p(\theta|d) \log \frac{p(\theta|d)}{q_\phi(\theta|d)}, \tag{9}$$

which vanishes as the approximant gets similar to the posterior. Since the approximant of the mapping should not depend on the individual data, the loss function will be the expectation value over $p(d)$ of $D_{\mathrm{KL}}(p||q_\phi)$:

$$L = \int \mathrm{d}d p(d) \int \mathrm{d}\theta p(\theta|d) \log \frac{p(\theta|d)}{q_\phi(\theta|d)}. \tag{10}$$

Using Bayes Theorem one gets:

$$L \simeq -\int \mathrm{d}\theta \pi(\theta) \int \mathrm{d}d p(d|\theta) \log q_\phi(\theta|d), \tag{11}$$

where the posterior at the nominator in the logarithm can be omitted since the minimization over ϕ does not depend on the true posterior. Simulating

events is equivalent to drawing N events from the prior, $\theta_i \sim \pi(\theta)$, and then data from the likelihood $d_i \sim p(d|\theta_i)$, with $i = 1, ..., N$. The loss used is then

$$L = -\frac{1}{N} \sum_i \log q_\phi(\theta|d), \tag{12}$$

where monte carlo (MC) sums have been used to approximate the integrals.

DINGO architecture consists of an embedding network that compresses data strains before passing them to the flow. The embedding is trained together with the flow. During training the network learns how to compress data, i.e. what are the most important features in the data.

0.4 GROWING PAINS

SCRIVI QUALCOSA SU SBI PER POP - similar process - simulated data and then you apply selection effects - problem: training can be very expensive/unfeasible - maybe can solve the groweing pains problem MAGARI SEZIONE CHIAMATA GROWING PAINS?

COMPLETED WORK

Transfer learning is a series of techniques used in machine learning to transfer knowledge from a neural network to another. These strategies allow to reduce training time, and, more importantly, could even improve the results respect to training the new run without any information from previous trainings.

My first projects regards testing transfer learning with DINGO for individual events analysis, following the multifidelity approach in Ref. **??**. The typical training dataset size to have well-trained NNs in DINGO is of the order of 10^6 simulations. Let us consider two waveform models. An high-fidelity model is an accurate but expensive model, with prohibitive costs that limit the simulation size under the typical order of magnitude (e.g. numerical relativity simulations). We also consider a low-fidelity model, less accurate and cheap, and in this case we easily simulate 10^6 data. Assuming the architecture of the flow in the two cases is the same, we fine tune the training for the high-fidelity model using the network parameters optimized during the low-fidelity training run. The NN for the high-fidelity model reaches higher performances when pretrained, partially inheriting knowledge acquired in the low-fidelity training run.

With this strategy one can analyze events with waveform models never used before, opening a window to analyses with, for instance, beyond general relativity (GR) models, or with precessing and eccentric waveforms. Another interesting application is population analyses, where one could use population-synthesis codes to simulate the dataset. This would allow to do inference directly on astrophysical processes, as the population parameters would be the input parameters of the codes, e.g. the supernova kick strength, common-envelope efficiency, metallicity, and winds strength.

0.5 DIAGNOSTICS TO CHECK THE PERFORMANCE OF THE NEURAL NETWORK

There are many diagnostic to check the quality of the posterior approximant.

The evolution of the loss function gives us information on how the training is proceeding. When training a NN, the dataset is split into a training set and validation set. The training set is used to update parameters of the NN to minimize the loss. The validation set is used to compute the loss when parameters are updated, thus checking how the NN is performing on unseen data. A sanity check consists in making sure that the validation and training loss to decrease together during training. When training is finished they should get to a plateau.

Importance sampling tests how close $q_\phi(\theta|d)$ and $p(\theta|d)$ are in multiple dimensions. During inference one draws samples θ_i from the approximant of the posterior $\theta_i \sim q_\phi(\theta|d)$. One can associate to each θ_i the a weight w_i:

$$w_i \propto \frac{\pi(\theta_i)p(d|\theta_i)}{q_\phi(\theta_i|d)}. \tag{13}$$

The variance of weights gives the sampling efficiency

$$\epsilon = \frac{(\sum_i w_i)^2}{n\sum_i w_i^2}. \tag{14}$$

Smaller values of ϵ correspond to poorer posterior approximations.

0.6 FROM NON-SPINNING TO ALIGNED-SPINS MODEL

To test the multifidelity procedure, we use a non-spinning waveform as low fidelity model, while the high-fidelity model has aligned spins. The approximant used to generate waveforms is IMRPhenomXPHM.

The size of the simulations for the low-fidelity model is 5×10^6, with a split of 95% training set and 5% validation set. For the high fidelity model, we generate multiple datasets, with size 5×10^6, 5×10^5, 5×10^4, and 8192. In the first three cases the split is the same as in the low-fidelity case, while in the latter case we split the dataset in two halves. For the case $\sim 10^6$ and $\sim 10^3$

simulations, we do an additional training where we pretrain the embedding only. Results for pretraining only the embedding are preliminary. We aim to explore this alternative as it would improve the high-fidelity flexibility, allowing us to use pretraining while changing the flow architecture.

Figure 1 shows the evolution during training time of the validation-set loss functions. In the two cases with higher number of simulations ($\sim 10^6$ and $\sim 10^5$), pretraining both the embedding and the flow does not greatly influence the values of the loss plateau. The biggest difference between this pretraining and training from scratch is the time it takes to plateau, resulting in lower computational costs when pretraining. With limited simulation budgets instead ($\sim 10^4$ and $\sim 10^3$), pretraining pushes the loss to lower values than traditional training. Pretraining the embedding with $\sim 10^6$ simulations allows us to get to slightly lower loss functions than training from scratch. When considering $\sim 10^3$ simulations, this strategy improves the results of traditional training but does not reach results as good as when pretraining both the embedding and the flow.

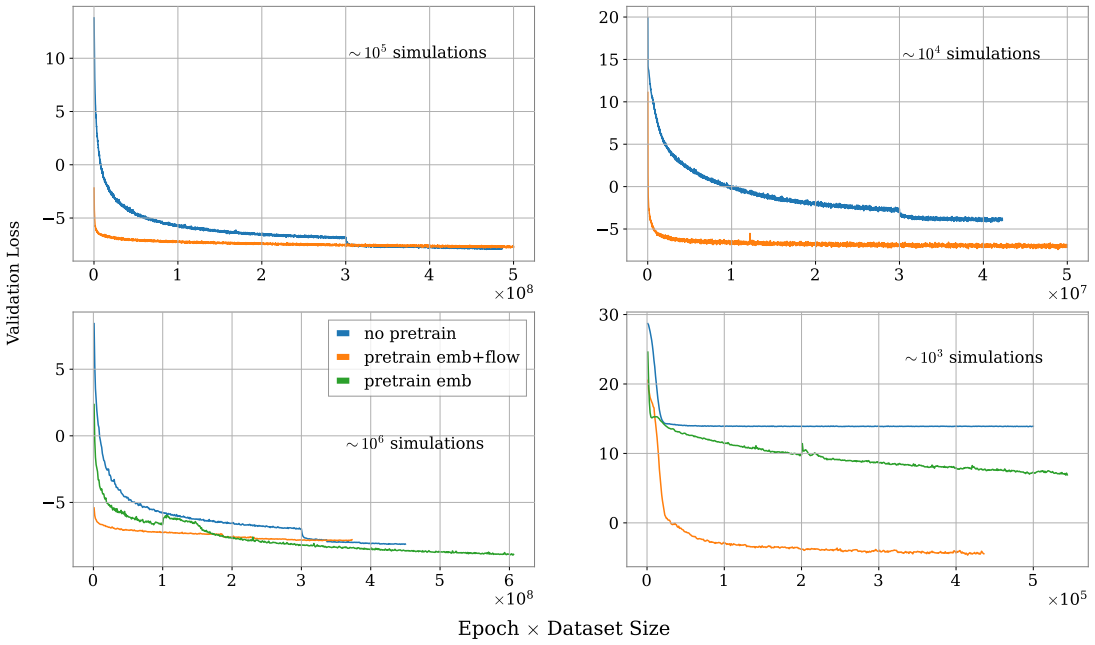

Figure 1: Evolution of the validation loss as a function of training time. The number of epochs is the number of times the full training set passes through the neural network. The top two panels show losses for the traditional training (blue) and pretraining both the embedding and the flow (orange) for $\sim 10^5$ (left) and $\sim 10^4$ (right) simulations. The two lower panels show the validation losses for $\sim 10^6$ (left) and $\sim 10^3$ (right). Additionally to the previous cases, results are shown for pretraining the embedding only (green)

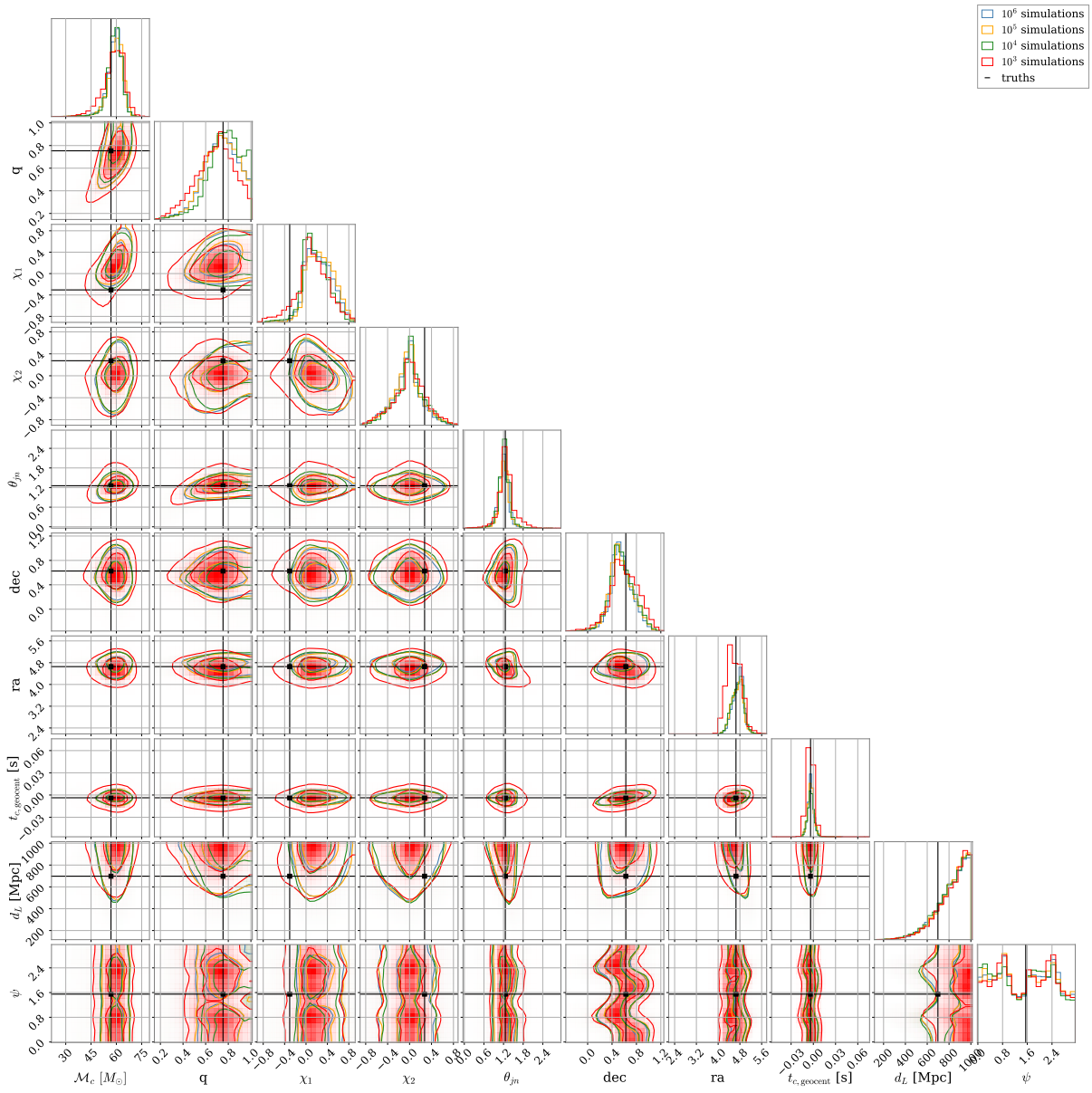
10^6 simulations
10^5 simulations
10^4 simulations
10^3 simulations
truths
$\mathcal{M}_c$ $[M_\odot]$
q
χ_1
χ_2
θ_{jn}
dec
ra
$t_{c,\mathrm{geocent}}$ [s]
d_L [Mpc]
ψ

Figure 2

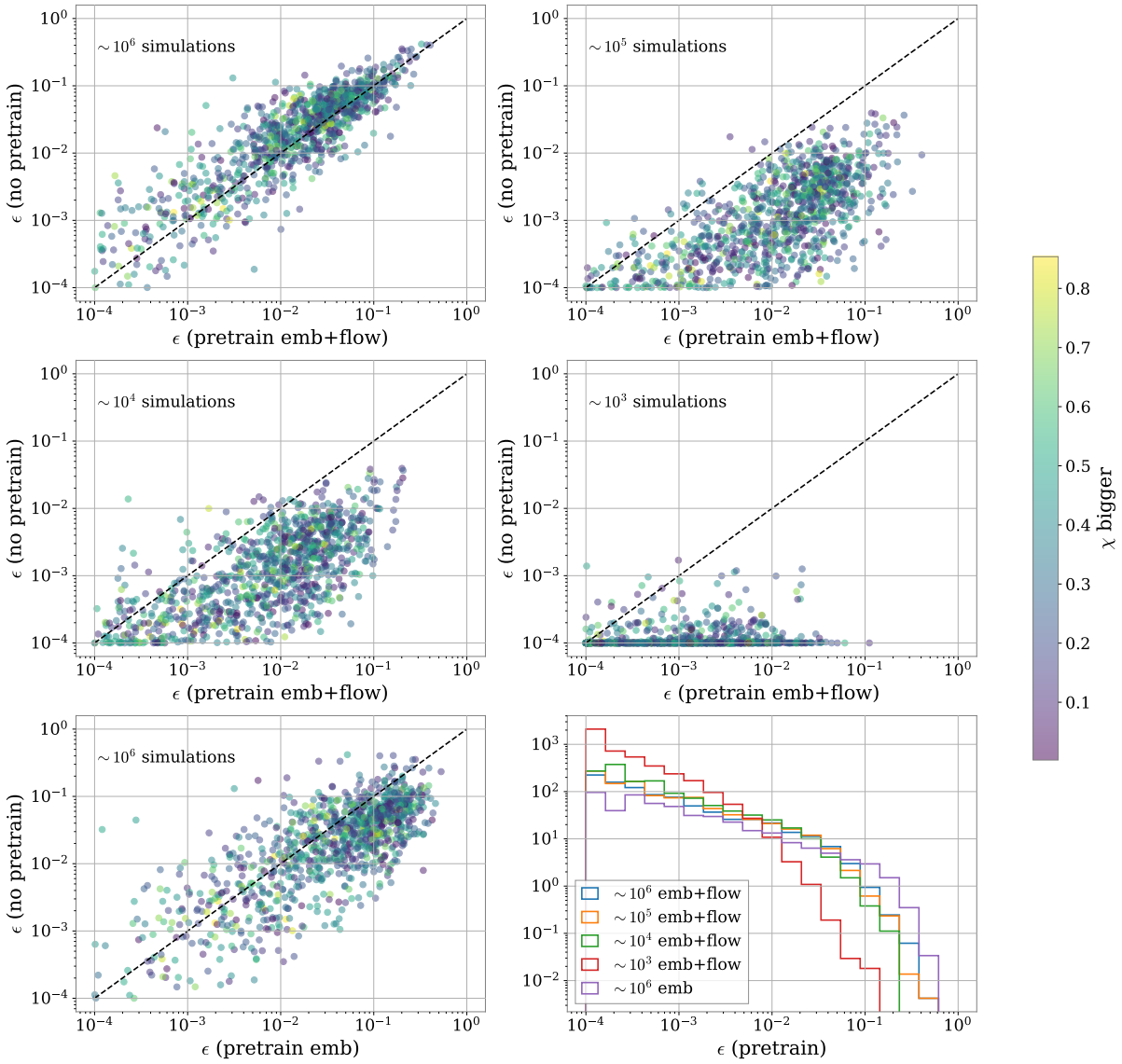
~ 10^6 simulations
~ 10^5 simulations
~ 10^4 simulations
~ 10^3 simulations
~ 10^6 simulations
ϵ (no pretrain)
ϵ (pretrain emb+flow)
ϵ (pretrain emb)
ϵ (pretrain)
χ bigger
~ 10^6 emb+flow
~ 10^5 emb+flow
~ 10^4 emb+flow
~ 10^3 emb+flow
~ 10^6 emb

Figure 3

WORK PLAN

a plan of work for the next 12 months - pop growing pains for sure - we talked about using astrophysical models as pop models. We need to understand better. Surely something more astro - another idea is to keep developing the current project (NR simulations, beyond GR models)

PERSONAL DEVELOPMENT PLAN

Copy of personal development plan summary and a statement of progress made towards those training goals