Normalizing Flows for Bayesian Statistical Inference Posterior Analysis

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE
in
ASTRONOMY AND DATA SCIENCE

Author :	Kutay Nazlı
Student ID :	3140881
Supervisor :	Elena Sellentin
Second corrector :	Javier Silva Lafaurie

Leiden, The Netherlands,

Typeset using LaTeX default style in AASTeX631

Normalizing Flows for Bayesian Statistical Inference Posterior Analysis*

Kutay Nazli[1]

[1]Leiden Observatory, Leiden University, Niels Bohrweg 2, 2333 CA Leiden, Netherlands

ABSTRACT

Context: Analyzing the shape distortions of galaxies on large scales due to weak lensing allows for statistical analysis of the shear field, which measures the overdensities of matter in the universe. Thus, weak lensing shear fields and their power spectra can be used to constrain the values of multiple cosmological parameters, such as Ω_m, σ_8 and H_0. These observables can be modeled independently of assumed cosmological models, allowing for a flexible complement to the current probes.

Aims: To model the posterior distributions of the low multipole ($\ell < 50$) E-modes of the power spectra of the weak lensing shear field obtained from the Almanac (Loureiro et al. 2023) Bayesian hierarchical model (BHM) using normalizing flows.

Methods: First, I present a mathematical framework adjoining normalizing flows and BHM. Within this framework, the analytically unknown posteriors of the low-ℓ are iteratively mapped to known probability distributions via variable transformations learned by neural networks. This enables the generation of new data, probability calculation and parameter inference.

Results: In this thesis, I present the mathematical framework and the initial code-base developed for solving simple inference problems using normalizing flows. I demonstrate the basic capabilities of the normalizing flow code, set up and execute an inference problem utilizing normalizing flows to ensure the accuracy of the mathematical formulation. I also present the initial progress of the next step in the project, which is generalizing the flow models to non-Gaussian functions.

Conclusions: The inference tests prove the accuracy of the assumptions and conclusions of the mathematical formulation. The normalizing flows code produces accurate results in Gaussian and simple non-Gaussian tests. However, more work needs to be done to reliably map to a larger selection of non-Gaussian functions.

1. INTRODUCTION

—In the absence of matter, photons travel along paths that minimize the distance they travel called geodesics. However, the presence of luminous or dark matter in the vicinity of their trajectory causes the geodesics to be bent due to the gravitational potential proportional to the mass affecting it. Thus, when observing light emitted by distant galaxies, it is possible to notice a systematic distortion of the shapes of these distant galaxies. Applying this general methodology to the entire night sky allows for the creation of weak lensing cosmic shear maps, which quantify how much and in which direction the shapes of the background galaxies would be distorted in a given line of sight. Since this phenomenon is well understood and observed, it is possible to accurately link the amount and direction of the distortion to the distribution and amount of mass along the path of photon travel. As a result, and for the first time in Castro et al. (2005) it becomes possible to use weak lensing to measure the three-dimensional matter distribution in the universe without having to rely on assumptions about cosmological evolution (as for example is the case in CMB measurements). Acquiring a model independent and accurate three-dimensional mass distribution as a function of time can allow for considerable improvement of the estimates we currently have of cosmological parameters that are sensitive of the mass distribution of the universe such as the Hubble constant H_0, matter density parameter Ω_m and matter fluctuation amplitude σ_8. On top of this, since the path diversion is caused by the total mass and not only the luminous mass of the foreground galaxies or clusters, combining weak lensing with observations of the luminous matter in these sources

Corresponding author: Kutay Nazli
nazli@strw.leidenuniv.nl

* Released on June, 10th, 2023

can lead to better understanding of the dark matter content of galaxies (Heavens 2009), which allows for detection of dark matter halos in galaxies using wide field cameras such as the Suprime-Cam on the Subaru Telescope (Miyazaki et al. 2002).

With upcoming stage-IV spectroscopic surveys, such as Euclid (Laureijs et al. 2011) and the Large Scale Survey Telescope (LSST) (LSST Science Collaboration et al. 2009), it is valuable to ensure that the approach taken to extract summary statistics from these extremely high dimensional datasets are accurate enough for scientific inference of cosmological parameters. The approach taken by ALMANAC is to create a Bayesian hierarchical model to infer the cosmic shear fields and their power spectra that do not depend on the assumed cosmological model (Alsing et al. 2016; Loureiro et al. 2023). Unlike its predecessors, ALMANAC is applicable in both full-sky and flat-sky approximation, via probing the entire posterior distributions of the E- and B-modes of the cosmic shear power spectra C_l. ALMANAC also recovers the marginalized posteriors of the power spectra, which are highly non-Gaussian for low-ℓ, which become Gaussian in higher-ℓ due to the central limit theorem. ALMANAC then samples the power spectra in harmonic space, while taking into account the uncertainties of the sky maps and the regions where it is not possible to observe distance galaxies (stellar and galaxy masks). However, the non-uniformity of the noise in the maps and the discontinuities caused by the masks causes the marginalized form of the posteriors to be unobtainable.

Since the analytic form of the marginalized power spectra posteriors is not obtainable, most parameter constraining methods that assume a given model are therefore left inaccurate. This attribute of this problem makes it a perfect use case for normalizing flows, which allow for mapping of unknown probability distributions to known ones through learnable variable transformations. Normalizing flows are used as a flexible and powerful tool in many science applications for simulation-based inference, both within and outside of astronomy (Dai & Seljak 2023; Lim et al. 2023; Wen et al. 2023). This thesis aims to provide a mathematical proof for employing normalizing flows in a Bayesian hierarchical model, to address concerns regarding parameter priors and to test the model in an inference problem, thus ensuring the validity of the mathematical conclusions.

Therefore, this work is structured as follows. In Section 2, I provide 1) an intuitive derivation of weak lensing and its usage as a probe for cosmological parameters, 2) a description of Bayesian statistics and inference problems in astronomy and 3) a summary of ALMANAC working principles and goals. In Section 3, I discuss 1) the mathematical proof of existence of solutions for function mapping via normalizing flows, 2) the code basis developed for implementing and training normalizing flows and 3) motivate the extension of the work to non-Gaussian distributions. In Section 4, I provide the aforementioned mathematical proof for marrying normalizing flows and Bayesian hierarchical models, followed by the results of an inference problem using normalizing flows. Finally, in Section 5, I provide the conclusions and the closing remarks and motivate the future work for this thesis and its parent project ALMANAC.

2. BACKGROUND

2.1. *Weak Lensing as a Probe for Cosmological Parameters*

—Put in simplest terms, the presence of strong gravitational potentials cause the trajectories of light rays to be noticeably bent. In astronomical contexts, this is referred to as *gravitational lensing*. To understand the more complex formulation that will be discussed later in this section, I will intuitively derive some of the key equations and concepts pertaining to weak lensing and how they can be used to probe some of the cosmological parameters of our universe.

Inspecting Figure 1 shows the simple setup in which gravitational lensing occurs. The light emitted from a background galaxy at comoving distance r_s is distorted by two foreground galaxies located at r_1 and r_2 by angles $\mathrm{d}\alpha_1$ and $\mathrm{d}\alpha_2$ respectively. In the plane of the sky (for example in an image taken of this field), this corresponds to a linear shift of $\delta x_\perp = \mathrm{d}\alpha_1(r_s - r_1) + \mathrm{d}\alpha_2(r_s - r_2)$. Under the assumptions of a flat ($K = 0$) universe, this can be generalized into an integral of the form

$$\delta x_\perp = \int_0^{r_s} (r_s - r)\,\mathrm{d}\alpha$$

Using the fact that $\frac{\mathrm{d}\alpha}{\mathrm{d}r} = -\frac{2}{c^2}\nabla_\perp\Phi$ (where $\nabla_\perp\Phi$ is the gradient of the Newtonian potential perpendicular to the line of sight given by r), we can write the angular shift in the sky as

$$\theta_\mathrm{s} - \theta_\mathrm{l} = -\frac{2}{c^2}\int_0^{r_s} \frac{(r_s - r)}{r_s}\nabla_\perp\Phi(r)\,\mathrm{d}r$$

where θ_s and θ_l are the angular positions of the source and the lens respectively. The associated lensing potential

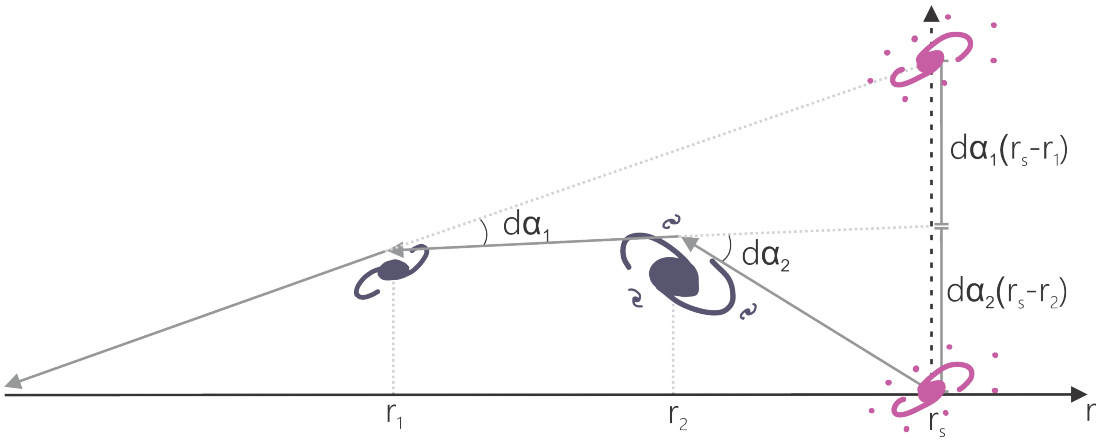

Figure 1. A simple visual representation of gravitational lensing. The figure depicts the bending of the path of the light rays emitted from a pink galaxy due to two foreground blue galaxies.

$\Psi(r_s)$ at the location of the source galaxy can be defined as following

$$\Psi(r_s) = \frac{2}{c^2}\int_0^{r_s} \frac{(r_s - r)}{r \cdot r_s}\Phi(r)\,\mathrm{d}r$$

such that the angular shift can be written now as

$$\theta_\mathrm{s} - \theta_\mathrm{l} = -\nabla_\theta \Psi$$

where ∇_θ is now the gradient along the angular directions in the sky. Please note that this is a simplified treatment of this theory, and a more detailed derivation can be found in Bartelmann & Schneider (2001). This above relation can also be written in the form

$$\vec{\theta_\mathrm{s}} = \mathbf{A}\vec{\theta_\mathrm{l}} \quad \text{where} \quad \mathbf{A} = \begin{pmatrix} 1-\kappa-\gamma_1 & -\gamma_2 \\ -\gamma_2 & 1-\kappa+\gamma_1 \end{pmatrix}$$

where κ is the spin-0 lensing convergence field and $\gamma = \gamma_1 + i\gamma_2$ is the spin-2 shear field, related to the lensing potential via

$$\kappa = \frac{1}{4}(\eth\bar{\eth} + \bar{\eth}\eth)\Psi \qquad \gamma = \frac{1}{2}\eth\eth\Psi$$

where $\eth$ and $\bar{\eth}$ are the spin raising and spin lowering operators defined in **?**. The effects of κ, γ_1 and γ_2 are intuitive in the flat-sky assumption where they are reduced to

$$\kappa = \frac{1}{2}(\Psi_{11} + \Psi_{22}) \quad \gamma_1 = \frac{1}{2}(\Psi_{11} - \Psi_{22}) \quad \gamma_2 = \Psi_{12}$$

where $\Psi_{12} = \frac{\partial}{\partial\theta_2}(\frac{\partial\Psi}{\partial\theta_1})$ are the partial derivatives of the lensing field with respect to the flat-sky coordinates (θ_1, θ_2). With this definition, the effects of the different convergence and shear terms on the images of galaxies is given in Figure 2. The convergence κ causes the entire field to magnify, while the two components of the shear γ causes stretching in the "cardinal" and diagonal directions respectively.

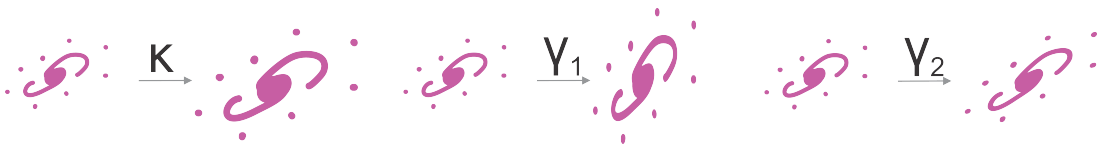

Figure 2. A simple visual representation of shear of galaxies due to weak gravitational lensing.

Weak lensing specifically refers to the region where $\kappa, \gamma_1, \gamma_2 << 1$. Hence, it is detected through the assumption that,

in a large enough given patch of the sky, the orientation of galaxies should be random. If this assumption is violated due to an observed preferred alignment of galaxies, this points to a field effected by weak gravitational lensing. In practice it causes a very faint warping of the field, such as in Figure 3.

Figure 3. Weak and strong gravitational lensing caused by galaxy cluster RCS2 J2327 depicted in this composite image from the VLT and Hubble. While some strongly lensed galaxies are immediately visible, a statistical study of the orientations of the surrounding galaxies allows the calculation of the lensing field and the Newtonian potential of the galaxy cluster. Image Credit: ESO & ESA/Hubble & NASA at https://esahubble.org/images/potw1752b/

Generalizing this idea to the entire night sky allows for creation of shear maps of the entire sky, which in turn can be used to derive the power spectra of κ and γ, which can in turn be used to constrain the Hubble's constant H_0, the total mass density Ω_m, and the magnitude of the density fluctuations σ_8.

We can look at how this can be achieved by taking a more detailed look at how the shear field γ and its power spectrum can be represented for the 3D sky. Any spin-s field on the sphere $f(\omega)$ can be decomposed into a basis of spin-s spherical harmonic functions ${}_sY_{lm}$ as

$$f(\omega) = \sum_{l=0}^{\infty} \sum_{m=-l}^{l} f_{lm}\ {}_sY_{lm}(\omega) \quad \text{where} \quad f_{lm} = \int f(\omega) {}_sY^*_{lm}(\omega)\, \mathrm{d}\Omega$$

where $\mathrm{d}\Omega$ represents an integral over the solid angles of sky. Using this basis, one can decompose $\Psi(r, \theta, \phi)$ into Ψ_{lm}, which makes it possible to relate the shear and convergence terms as

$$\kappa_{lm} = -\frac{1}{2} l(l+1)\Psi_{lm} \quad \gamma_{lm} = \frac{1}{2}\sqrt{(l-1)l(l+1)(l+2)}\Psi_{lm}$$

Additionally, the γ_{lm} components can be further decomposed into scalar gradient E and scalar curl B modes, known as a spin-s generalization of the Helmholtz decomposition in Newman & Penrose (1966). This yields the well documented

components (Loureiro et al. 2023)

$$\gamma^E_{lm} = -\frac{1}{2}\int [\gamma(\omega)\ {}_2Y^*_{lm}(\omega) + \gamma^*(\omega)\ {}_{-2}Y^*_{lm}(\omega)]\,\mathrm{d}\Omega \quad \gamma^B_{lm} = \frac{i}{2}\int [\gamma(\omega)\ {}_2Y^*_{lm}(\omega) - \gamma^*(\omega)\ {}_{-2}Y^*_{lm}(\omega)]\,\mathrm{d}\Omega$$

For two redshift bins (represented by α and β), the covariance between the E-mode, B-mode and cross-EB are given by (Alsing et al. 2016)

$$\langle \gamma^{E(\alpha)*}_{lm}\gamma^{E(\beta)}_{l'm'}\rangle = C^{EE}_{l,\alpha\beta}\ \delta_{ll'}\delta_{mm'} \qquad \langle \gamma^{E(\alpha)*}_{lm}\gamma^{B(\beta)}_{l'm'}\rangle = C^{EB}_{l,\alpha\beta}\ \delta_{ll'}\delta_{mm'} \qquad \langle \gamma^{B(\alpha)*}_{lm}\gamma^{B(\beta)}_{l'm'}\rangle = C^{BB}_{l,\alpha\beta}\ \delta_{ll'}\delta_{mm'}$$

where δ_{ab} is the Kronecker-delta. Most cosmological models predict $C^{BB}_{l,\alpha\beta} = 0$, and parity considerations lead to $C^{EB}_{l,\alpha\beta} = 0$. Thus, we are only left with $C^{EE}_{l,\alpha\beta}$ as the non-zero terms. To finally see how these C^{EE}_ls can be used to constrain cosmological parameters, in a short aside, I will give some elementary background about the Limber's approximation (Limber 1954). It is common is astronomy to observe a quantity on sky that is the projection of the properties of some three-dimensional field along the line-of-sight. Then using Limber's approximation it is possible to relate

$$F(\theta,\phi) = \int q(z) f(\theta,\phi,z)\,\mathrm{d}z$$

where $q(z)$ is some radial weighting function, F is the quantity observed on sky and f is the three-dimensional field causing the observed behavior (as presented in Kaiser (1998)). Then, we can go back and use this approximation to write

$$C^{EE}_{l,\alpha\beta} = \int q_{(\alpha)}(r) q_{(\beta)}(r) \frac{(1+z)^2}{r^2_m(r)} P_\delta\left(\frac{l}{r_m(r)}, r\right)\mathrm{d}r$$

where r is the comoving radial distance, $P(k,r)$ is the 3D matter power spectrum and $r_m(r)$ is the transverse comoving distance corresponding to comoving distance r. Finally, the weighting functions of lensing are given by, for a given redshift

$$q_{(\beta)}(r) = \frac{3\Omega_m H_0^2}{2}\int_r^{r_H} n_{(\beta)}(r')\frac{r_m(r'-r)}{r_m(r')}\,\mathrm{d}r'$$

where $n_\beta(r)\,\mathrm{d}r = p_\beta(z)\,\mathrm{d}z$ is the redshift distribution for galaxies around redshift of α. With this formulation, $C^{EE}_{l,\alpha\beta}$ are sensitive to the models of ΛCDM cosmology thorough the dependence of $q_{(i)}(r)$ on Ω_m (these functions are actually invariant to changes in H_0 even though the constant explicitly appears in their definition due to the integral over the comoving coordinate also being sensitive to H_0). $C^{EE}_{l,\alpha\beta}$ are also sensitive to H_0 through another means, as the 3D matter power spectrum $P(k,r)$ define its volume scales in $h^{-3}\mathrm{Mpc}^3$ where $h = H_0/100$ (Hall 2021). Thus, we can see how the cosmological models can effect our expectations of the shear field in large scales, and how studying the shear field can help constrain parameters of our cosmological models.

2.2. *Inference in Astronomy and Bayesian Statistical Analysis*

—After having established the astrophysical foundation this thesis rests on, I would also like to take the time to jump disciplines and do the same on the mathematical side. Both this thesis and the much larger project it is a part of (which will be summarized later in this section) are astrostatistics projects. Therefore to understand this thesis and its goals, I will summarize how statistics is generally conducted in astronomy.

For starters, let us look at a familiar problem from outside astronomy. Wide clinical cohort studies have proven that smoking at least 1 cigarette a day for an extended period of time is a strong predictor for the 4 most common types of lung cancer (Tran et al. 2013). In this specific study and its peers that aim to derive a statistical relationship between smoking and lung cancer (for example, something along the likes of "possibility of getting lung cancer in lifetime as a function of number of cigarettes smoked in a day"), both of the random variables, namely how much someone smokes in a day and whether they develop lung cancer, are both directly observable to the medical scientists conducting this study. Therefore, with a large enough sample in smokers and lung cancer patients, it is possible to derive this relationship without any (or very little) assumptions about both of these variables.

In astronomy, however, we are often not as lucky. Most of the core variables of astrophysical theory are not directly

observable quantities, and only exist as necessities of our theories of physics. We can think of a simple relevant example is astronomy: the current age of the universe. Direct observations of the beginning of the universe (as far as our technology and our understanding of physics points out) is unfeasible, meaning that the age of the universe is not directly observable. We can only *infer* the age of the universe from other directly observable quantities, such as the cosmic microwave background. Problems such as this where only one of the variables is observable, and the path to the claims about the statistics about the unobservable quantity passes through theories about our understanding of physics are called *inference problems*.

The last data release of the Planck satellite from 2018 predicts the age of the universe to be 13.787 ± 0.020 billion years (Planck Collaboration et al. 2020). However this number and its accompanying precision are undeniably dependant on out theories of the early universe, where the cosmic microwave background and its anisotropies predict this age of the universe assuming the connection going from the anisotropies to the beginning of the universe is the truth. Then, how do we make sure that our mesaurements of the fundamental unobservable variables of the universe are not biased heavily due to our current conceptions of our theory? The key is *Bayesian statistical analysis*, where we can test and quantify the believability of our models using our data.

The complete Bayesian statistical analysis process is very well known and discussed, so I will only motivate it here and discuss it further when I discuss the specifics of the project this thesis is a part of. Let us look at Bayes' theorem in the context of a Bayesian statistical analysis

$$P(\theta|\vec{x}) = \frac{\mathcal{L}(\vec{x}|\theta)\pi(\theta)}{\epsilon(\vec{x})}$$

Here, the left-hand side probability $P(\theta|\vec{x})$ is the posterior, the quantity that tells us how likely our model parameters θ are given that we have observed some data $\vec{x}$. In the above example, this could be the probability that the age of the universe is ~ 13 billion years old given the Planck CMB data. The $\mathcal{L}(\vec{x}|\theta)$ is the likelihood, the probability of the current model parameters giving rise to the observed data. This can, for example, be acquired with simulating mock universes with the same model parameters (in this case could be the Hubble constant H_0, matter density parameter Ω_m and matter fluctuation amplitude σ_8), evolving them till the hot plasma of the early universe cools down and the photons decouple, and comparing the anisotropies with the Planck observations. The $\pi(\theta)$ parameter is a prior on the model parameters, and is the term where uncertainty from the specific model parameters is input into the analysis. It is the answer to the question "How would the credibility our age estimate change if the assumed different values for the (above) model parameters?". Finally, the last term $\epsilon(\vec{x}) = \int \pi(\vec{x}|\theta)\,\mathrm{d}\theta$ is the evidence, a prior weighing the confidence in our model. Since now we integrate over all the possible parameters θ, it shows the probability the model (λCDM, for example) produces this CMB data at all. If this likelihood is not high, we would have to give theoretical physicists a visit.

Even with this simple setup, it is possible to see how Bayesian statistical analysis aims to use the information available in the theory, the data and the errors involved to model the outcomes while working under assumed uncertainty.

2.3. Almanac: *Inference of Cosmological Parameters with Weak Lensing*

—As I have alluded to before, this thesis exists as a part of a larger project named Almanac, which aims to provide a parameter inference algorithm with shear maps and power spectra (as discussed in 2.1) that are sampled independently of the cosmological theory assumed. The details of this work can be found in Loureiro et al. (2023), but I will illustrate the key process here for completeness. The posterior of Almanac looks like the following

$$P(C, \vec{a}|\vec{d}, N) \propto \mathcal{L}(\vec{d}|\vec{a}, N)\mathcal{G}(\vec{a}|C)\pi(C)$$

And the process behind the Bayesian hierarchical model of Almanac is illustrated and discussed in Figure 4 below.

If one inspects the marginalized posterior distributions for C_l^{EE} from Almanac for two tomographic redshift bins (bin-1 and bin-2), an expected yet challenging result appears.

Figure 5 display this challenge, where the apparent non-Gaussianity of the lower-l multipoles poses a problem. Without any knowledge the analytical form of the underlying family of distributions available, it is not possible to fit a function to these probability distributions using any of the conventional methods of parameter fitting or likelihood minimization. This lays the scientific goal of this thesis.

The method that is utilized is known as *normalizing flows*, which allows for the mapping of analytically unknown (but can be sampled from) probability distributions to be mapped to probability distributions with known analytic forms

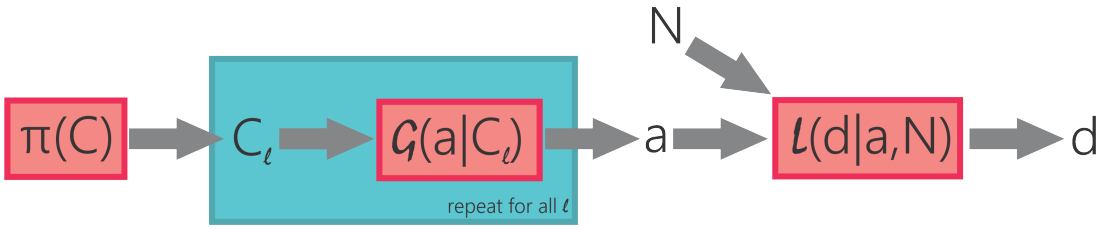

Figure 4. A diagram representing the Bayesian hierarchical model of ALMANAC. Adapted from Fig. 1 in Loureiro et al. (2023). Begin by sampling a series of C_l from a posterior distribution $\pi(C_l)$. These are then used to sample shear field coefficients $\vec{a}$ (following the original paper notation, would correspond to a vector of Y_{lm} in Section 2.1). These shear field coefficients are used in conjunction with noise covariance N to sample the actual full-sky shear field maps d (again, in this work, denoted with γ).

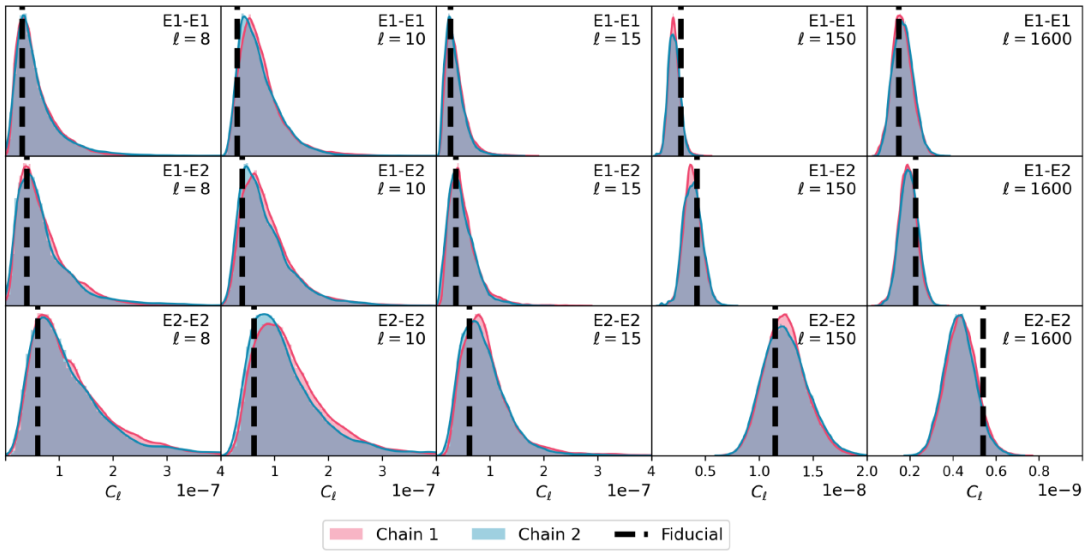

Figure 5. Fig. 6 of Loureiro et al. (2023) displayed here for discussion. The figure displays the marginalized one dimensional posteriors of C_l^{EE} for $l = 8, 10, 15.150, 1600$ and two separate Hamiltonian Monte Carlo chains. The posteriors for large-l are fairly Gaussian due to being sums of a large number of spherical harmonics (more Y_{lm} for larger l) in combination with central limit theorem. This however, is not the case for low-l.

via series of variable transformations learned iteratively using neural networks. The next section lays the mathematical foundation behind normalizing flows, and further discusses their implementation for this thesis.

3. METHODOLOGY

3.1. *Mathematical Proof of Normalizing Flows*

—At the conclusion of the last section I have introduced the concept of normalizing flows to obtain the probabilities from analytically unknown distributions through mapping known probability distributions to them via a series of variable transformations. This begs the natural question of whether normalizing flows can map to any unknown probability distribution. This section aims to adapt the proof from Papamakarios et al. (2019) to show that it is indeed possible to do so without any prior knowledge on the unknown probability distribution (hereafter called the target) but just some basic assumptions about it.

Label $P_x(\vec{x})$ the probability of $\vec{x}$ in x-space the target distribution and $P_u(\vec{u})$ the well-known prior distribution of $\vec{u}$. Safely assume that $P_x(\vec{x}) > 0 \ \forall \vec{x} \in \mathbb{R}^D$ and that $Pr(x_i|\vec{x}_{<i})$, namely the cumulative probability of the i-th element of $\vec{x}$ conditional on the vector of all the previous elements of $\vec{x}$, here given as $\vec{x}_{<i}$ is differentiable with respect to $(x_i, \vec{x}_{<i})$ for all indices i for the vector $\vec{x}$. Then we can decompose $P_x(\vec{x})$ into a product of conditional probabilities

$$P_x(\vec{x}) = \prod_{i=0}^{D} P_x(x_i|\vec{x}_{<i})$$

where $P_x(x_i|\vec{x}_{<i})$ is now the probability (non-cumulative) of the i-th element of $\vec{x}$ conditional on the previous elements, and the product goes up to D since $\vec{x}$ is assumed to be D-dimensional. In addition, we can say that $P_x(x_i|\vec{x}_{<i}) > 0$ for all i and $\vec{x}$ since we have previously assumed $P_x(\vec{x}) > 0$ everywhere. Then, define the transformation $F : \vec{x} \mapsto \vec{z} \in (0,1)^D$ whose i-th element is defined as the cumulative probability of the i-th element, $Pr(x_i|\vec{x}_{<i})$ as denoted above. Or, more explicitly

$$z_i = F_i(x_i, \vec{x}_{<i}) = \int_{-\infty}^{x_i} P_x(x_i'|\vec{x}_{<i})\,\mathrm{d}x_i' = Pr(x_i|\vec{x}_{<i})$$

as we have defined. Note a couple of facts about F. First off, F is differentiable with respect to $\vec{x}$ since all of its constituent elements F_i are differentiable with respect to their inputs by one of the initial assumptions. Additionally, each $F_i(\_, \vec{x}_{<i}) : \mathbb{R} \mapsto (0,1)$ are invertible since its derivative $\frac{\partial F_i}{\partial x_i} = P_x(x_i|\vec{x}_{<i})$ is positive everywhere by assumption. This means we can invert F element by element

$$x_i = (F^{-1})_i(z_i) = (F_i(\_, \vec{x}_{<i}))^{-1}(z_i)$$

for $i \in 1, ..., D$. This shows that F is invertible by this formulation. Since the derivatives $\frac{\partial F_i}{\partial x_i}$ do not depend on x_j for $j > i$, the Jacobian of F is lower triangular. For a triangular matrix, the determinant is simply given by the product of the elements on the diagonal $\frac{\partial F_i}{\partial x_i}$. Thus

$$\det J_F(x) = \prod_{i=0}^{D} \frac{\partial F_i}{\partial x_i} = \prod_{i=0}^{D} P_x(x_i|\vec{x}_{<i}) = P_x(\vec{x})$$

and hence is non-zero everywhere. This suggests the existence of the inverse of $J_F(x)$ and it is simple to show that it is equal to the Jacobian of F^{-1}. This proves that F is a diffeomorphism, a differentiable bijection with a differentiable inverse, which is necessary for a flow-like model to operate. Finally, using a change of variables, we can show

$$P_z(\vec{z}) = P_x(\vec{x})\ |\det J_F(\vec{x})|^{-1} = P_x(\vec{x})\ |P_x(\vec{x})|^{-1} = 1$$

and that $\vec{z}$ is distributed uniformly over the D-dimensional cube in $(0,1)^D$. This proves that any target probability distribution can be mapped to a uniform distribution of the same dimension. To further generalize this to any prior distribution, it is possible to find a diffeomorphism between the prior and the uniform distribution, and use it to map from the prior distribution to the uniform first, followed by using F^{-1} as defined above to go to the target distribution. Chaining multiple of these transformations with different prior and target distributions in intermediate steps is the mathematical working principle of normalizing flows.

3.2. *Building Normalizing Flows*

—In the previous section I have shown that it is theoretically always possible to find a transformation between two probability distributions. In practice, it is difficult to find this transformation analytically, as we go not have access to the original probability distribution $P_x(\vec{x})$ (or the cumulative one by extension). Thus, the idea is to chain multiple well-known transformations together to approximate the theoretical transformation F from the previous section. Then, the change of variables rule looks like the following, given in log form

$$\log P_x(\vec{x}) = \log P_z(f_n(\vec{z}_{n-1})) + \log\left|\det\frac{\mathrm{d}f_1(\vec{x})}{\mathrm{d}\vec{x}}\right| + \log\left|\det\frac{\mathrm{d}f_2(\vec{z_1})}{\mathrm{d}\vec{z_1}}\right| + \log\left|\det\frac{\mathrm{d}f_3(\vec{z_2})}{\mathrm{d}\vec{z_2}}\right| + ... + \log\left|\det\frac{\mathrm{d}f_n(\vec{z}_{n-1})}{\mathrm{d}\vec{z}_{n-1}}\right| \quad (*)$$

where $f_1 : x \mapsto z_1$ and every other $f_i : z_{i-1} \mapsto z_i$ for i in $2, ..., n$. In this case, $P_x(\vec{x})$ represents the target distribution, and $P_z(\vec{z}_n)$ would be the prior distribution, generally selected as a standard normal distribution of the same dimensionality as $\vec{x}$. Even though the discussion of this is beyond the scope of this thesis, this causes flow-based models to be far more expensive to train and sample from compared to other models that can learn and produce samples from data distributions. Unlike the other options, however, this fact combined with the invertibility of the flow transformations allows for the modeling of the true data distribution, lossless reconstruction, and accurate probability estimates[1]. In a more visual form, the working principle of a chain of flow transformations looks like Figure 6.

[1] Other generative models such as generative adversarial networks (GANs) or variational autoencoders (VAEs) all map the data distribution to a latent space representation of a much smaller dimension than the data itself. This allows for faster training and sampling, however causes losses in expressivity.

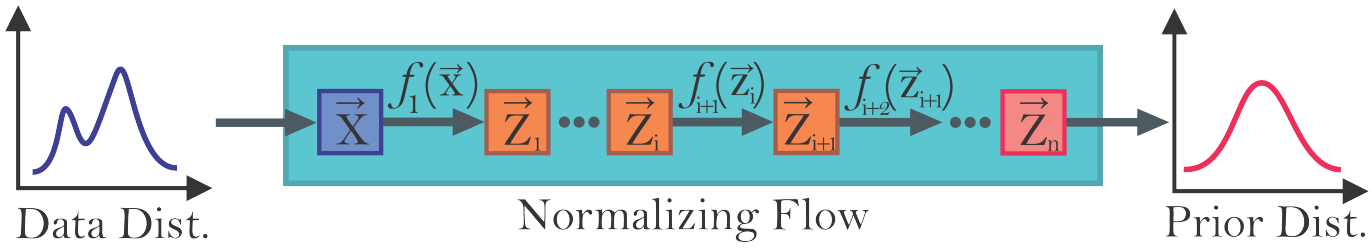

Figure 6. A simple visual representation of normalizing flows. The blue curve on the left represents the original data (or target) distribution $P_x(\vec{x})$, while the pink one on the right represents the prior distribution $P_z(\vec{z}_n)$. Note that the flow transformations f_i are all invertible, so even though the arrows point in the (what will be soon defined as) forward direction, they can also be used in the reverse direction to go from the prior to the data distribution. This is the generative aspect of normalizing flows.

—Now that we have seen the overall structure of normalizing flows, I will talk about a natural choice for the well-known transformation functions I have alluded to in the previous paragraphs. Thinking about the discussion of normalizing flows in general, we require the transformation functions to be diffeomorphisms, differentible bijections with differentible inverses as discussed in Section 3.1. However, expressive flexibility (aka. possessing free parameters that allow for the range of the transformation to be variable) is also of utmost importance. Many functions satisfy these conditions for normalizing flows, and Papamakarios et al. (2019) offers detailed descriptions for many. For now however, I will limit the discussion to a simple shift-scale transformation called *affine coupling transformation*, but a more complex and expressive transformation will be discussed in the next section. This section of the discussion follows material taught in the Deep Learning lectures from University of Amsterdam (UvA), available online (Lippe 2022), which I have used to build up most of my initial understanding and code base for this thesis.

The coupling aspect of the affine coupling transformation comes from the splitting of the input vector $\vec{z}$ into two pseudo-random parts, $\vec{z_1}$ and $\vec{z_2}$. I will talk about how this is implemented, but for now, assume such a split is conveniently available. The two parts are coupled because the shift parameters $\vec{T}$ and scale parameters $\vec{S}$ returned from the deep neural network for the first part of the input $\vec{z_1}$ are used instead to transform the second part of the input $\vec{z_2}$ and vice versa. Then in this sense, every f_i of the flow looks like the following

$$\vec{z}_{i,1} = f_i(\vec{z}_{i-1}) = \exp\left(\vec{S}(\vec{z}_{i-1,2})\right) \cdot (\vec{z}_{i-1,1} + \vec{T}(\vec{z}_{i-1,2}))$$

where the operation in the right-hand side of the second equals sign is performed element-by-element to the involved matrices and $\vec{S}(\vec{z}_{i-1,2}))$ represents the scale transformation variables learned from the second half of the input $\vec{z}_{i-1}$ and so on. I will also discuss the neural networks used in this implementation, but for now, the affine coupling transforms used in this thesis are implemented as follows

```
class AffineCouplingTransform(nn.Module):
    def __init__(self, network, mask, input_channels) -> None:
        super().__init__()
        self.network = network
        self.scaling_factor = nn.Parameter(torch.zeros(input_channels))
        self.register_buffer('mask', mask)

    def forward(self, z, log_det_jacob, reverse = False):
        z_in = z * self.mask
        nn_out = self.network(z_in)
        s, t = nn_out.chunk(2, dim = 1)

        stabilizing_factor = self.scaling_factor.exp().view(1,-1,1,1)
        s = torch.tanh(s / stabilizing_factor) * stabilizing_factor

        s = s * (1 - self.mask)
        t = t * (1 - self.mask)

        if not reverse:
            z = (z + t) * torch.exp(s)
            log_det_jacob += s.sum(dim=[1,2,3])

        else:
            z = (z * torch.exp(-s)) - t
            log_det_jacob -= s.sum(dim=[1,2,3])

        return z, log_det_jacob
```

where the modules are built onto the existing PyTorch neural network architecture[2]. Note that the affine coupling layer takes and returns two variables in its `forward` implementation, `z` and `log_det_jacob`. Here `z` is defined exactly as described earlier in this section, and `log_det_jacob` refers to the cumulative sum of the determinants of the Jacobian matrices of the transformations that are passed from one layer to the next. Notice the input `z` is passed through a mask that is passed to the layer at initialization. This mask performs the splitting of the input into the two parts $\vec{z_1}$ and $\vec{z_2}$ as discussed above. There are multiple methods in how this mask can be implemented, but here I mainly work with a checkerboard mask that masks every other element from the original input `z`, created as such

```
def make_checkerboard_mask(height, width, invert = False):
    x, y = torch.arange(height, dtype=torch.int32), torch.arange(width, dtype=torch.int32)
    xx, yy = torch.meshgrid(x, y, indexing = 'ij')
    mask = torch.fmod(xx + yy, 2)
    mask = mask.to(torch.float32).view(1,1,height,width)
    if invert:
        mask = 1 - mask
    return mask
```

where the `invert` flag allows for either starting with an accepted or rejected element (aka. inverts the mask). Once the data has been masked, the remaining half is passed to the `network` associated with this affine coupling layer. This network is same as the network used in the course tutorials from UvA at Lippe (2022), which itself is inspired by the more robust implementation known as Flow++ (Ho et al. 2019) where its full description and discussion is available. The specifics are beyond this work's scope, but the implementation can be found in Appendix A. For this discussion, it is important to realize that the shift parameters $\vec{T}$ and scale parameters $\vec{S}$ are acquired from the neural network at this point, and multiplied by the inverted mask. Finally, as alluded before, for this affine layer and all layers that will be discussed from hereon, the `reverse` direction refers to the direction of the flow going from the prior distribution to the data distribution, while the `not reverse`, or forward direction refers to transforming from the data distribution to the prior distribution.

—With the above components in mind, the general flow (aka. composed of multiple flow transformations) can be built using the following code, where I have left out some methods of the class that have to do with training the models using the PyTorch Lightning module[3].

```
class Flow(pl.LightningModule):
    def __init__(self, flows, importance_samples = 8) -> None:
        super().__init__()
        self.flows = nn.ModuleList(flows)
        self.importance_samples = importance_samples
        self.prior = torch.distributions.normal.Normal(loc = 0., scale = 1.0)

    def forward(self, input):
        return self._get_likelihood(input)

    def encode(self, input):
        z, log_det_jacob = input, torch.zeros(input.shape[0], device = self.device)
        for flow in self.flows:
            z, log_det_jacob = flow(z, log_det_jacob, reverse = False)
        return z, log_det_jacob

    def _get_likelihood(self, input, return_ll = False):
        z, log_det_jacob = self.encode(input)
        log_pz = self.prior.log_prob(z).sum(dim=[1,2,3])
        log_px = log_pz + log_det_jacob
        neg_log_llh = -log_px
        bits_per_dim = neg_log_llh * np.log2(np.exp(1)) / np.prod(input.shape[1:])
        return bits_per_dim.mean() if not return_ll else log_px

    @torch.no_grad()
    def sample(self, shape, z_init = None):
        if z_init is None:
            z = self.prior.sample(sample_shape = shape).to(self.device)
        else:
            z = z_init.to(self.device)

        log_det_jacob = torch.zeros(shape[0], device = self.device)
        for flow in reversed(self.flows):
```

[2] PyTorch documentation available at: https://pytorch.org/

[3] The documentation for Lightning available here: https://lightning.ai/docs/pytorch/latest/

```
            z, log_det_jacob = flow(z, log_det_jacob, reverse = True)

        return z
```

The `Flow` object is initialized with its flow transformation layers, objects such as the `AffineCouplingTransform` from above, in a list in `self.flows`. The prior distribution that I have extensively referred to in my mathematical and conceptual descriptions is initialized in the `self.prior` as a standard normal distribution. Here, the `encode` method allows the input to the flow (which would be the data $\vec{x}$ in this case) to be passed through all of the layers of flow transformations in the flow, and additionally accumulates the `log_det_jacob` along the way.

This `encode` method is used the method that the `Flow` executes in its forward pass, defined here as `_get_likelihood`. This method runs the input data $\vec{x}$ through the forward direction to get the representation of it in the prior distribution (or generally known as the latent) space, previously labeled $\vec{z_n}$. Once the data is transformed, the latent space probabilities can be obtained from the prior distribution, mathematically $\sum \log P_z(z_n)$, or in this case `log_pz`. We can then use the transformation formula $(*)$ to get the data distribution space probability, mathematically previously referred to as $\log P_x(\vec{x})$, or here `log_px`. Since this is inherently a minimization problem, instead of the log-likelihood, one uses either the negative log-likelihood, or in this case the bits-per-dimension, defined as

$$\text{bits-per-dimension} = \text{negative log-likelihood} \cdot \log_2(e) \cdot (\text{data dimension})^{-1}$$

which can be seen as a dimension-normalized negative log-likelihood, allowing for a generalized tracker regardless of the dimensionality of the problem. Hence the training algorithm presented in its entirety in Appendix B uses the bits-per-dimension metric to find the best fitting transformation variables, $\vec{T}$ and $\vec{S}$ for the case of the affine coupling transform.

Finally, the `sample` method represents the previously mentioned generative of `reverse` direction of the flow. Here, a random sample from the prior distribution is acquired (which is a trivial problem since the prior is a well-known function). Then, this sample is passed through the flow backwards to transform it from the prior space to the data space, resulting in a randomly generated (and most likely never-before-seen) sample of data (hencewhy flow algorithms are generative algorithms). While the forward direction of the flow can be used to acquire probabilities of available data, the reverse direction can be used in generating new data for parameter fitting and inference, which will be demonstrated in Section 4 in a case study.

3.3. *Gaussian vs. Non-Gaussian Normalizing Flows*

—In the previous section I have laid out the mathematical proof of the existence of flow transformations that can map any probability distribution to another, and went over some of the specifics of implementing and training such normalizing flows using affine coupling transforms as the basis. Although this approach is very powerful for many cases such as image generation, in this case it falls short for a specific reason. Affine coupling transforms, their combinations, and moreover the inverses of such combinations, are linear transformations. This means that, if the prior is selected as a standard Gaussian function, the target functions it can mathematically map to are limited to other Gaussian functions, as Gaussians conserve their Gaussianity under linear transformations. In Section 2 I had alluded that the requirement for normalizing flows arose from the apparent non-Gaussianity of the distributions of some of the C_l's. Hence, to accurately model these distributions, affine coupling transforms are not satisfactory on their own.

To resolve this limitation, it is possible to add other diffeomorphisms as transformation options to the flow besides the affine coupling transforms. One such is candidate is the logarithmic transformation. It is remarkably simple in its implementation

```
class LogTransform(nn.Module):
    def __init__(self) -> None:
        super().__init__()

    def forward(self, z, log_det_jacob, reverse = False):
        z = z.view(-1, 1, z.shape[-2], z.shape[-1])

        if not reverse:
            z = torch.log(z)
            log_det_jacob += torch.log(torch.abs(1/z)).sum(dim=[1,2,3])

        else:
            z = torch.exp(z)
            log_det_jacob -= torch.log(torch.abs(1/z)).sum(dim=[1,2,3])
```

```
        return z, log_det_jacob
```

yet when combined with an affine coupling transformation, can allow for mapping to any logarithmic function of the Gaussian variable of the form $z \sim a^{b \cdot x + c}$ where $x \sim \mathbb{G}(\mu, \sigma)$. Even though the transformation only transforms with respect to the natural base e, this is possible because

$$\ln a^{b \cdot x + c} = (b \cdot x + c) \cdot (\ln a) = (b \cdot \ln a)x + (c \cdot \ln a)$$

and a shift of $(c \cdot \ln a)$ followed by a scale by $(b \cdot \ln a)$ allows the original Gaussian distribution to be recovered. Figure 7 shows a case of this transform combined with the aforementioned affine coupling transform used together. The prior in this case is a 3×3 standard normal distribution, and the target distribution is a 3×3 matrix of data coming from 2 separate lognormal distributions, where 5 entries in the matrix come from one distribution and 4 from the other. This test is a simple yet interesting result, as the two distributions are independent. Even though the CNN used to learn the flow parameters sees data from both distributions at every training step, it is able to distinguish between them and fit to them separately. Thus Figure 7, which stows the histogram for the flattened data, shows two well-fit lognormal peaks.

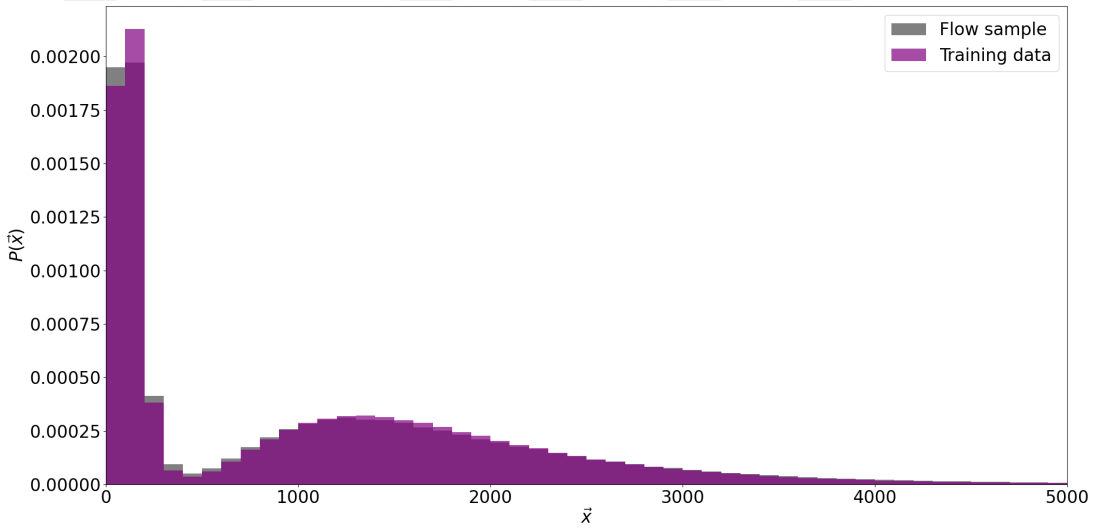

Figure 7. Depicting a sample generated from a flow trained on a set of lognormal data coming from 2 separate distributions. The flow coefficients learned accurately distinguish between the two independent distributions.

Another simple yet powerful option in combination with the previous two mentioned is the coupled power transformation

```
class PowerTransform(nn.Module):
    def __init__(self, network, mask, input_channels) -> None:
        super().__init__()
        self.network = network
        self.scaling_factor = nn.Parameter(torch.zeros(input_channels))
        self.register_buffer('mask', mask)

    def forward(self, z, log_det_jacob, reverse = False):
        z = z.view(1, 1, z.shape[-2], z.shape[-1])

        z_in = z * self.mask
        nn_out = self.network(z_in)

        power, sign = nn_out.chunk(2, dim = 1)

        stabilizing_factor = self.scaling_factor.exp().view(1,-1,1,1)
        power = 1 + (torch.tanh(power / stabilizing_factor) * stabilizing_factor)
        sign = torch.sign(torch.tanh(sign) + torch.rand_like(sign)*1e-8)
        power = sign * power * (1 - self.mask) + self.mask

        if not reverse:
```

```
            z = z**(1/power)
            log_det_jacob += torch.log(torch.abs(((1/power)*z**(1/power -1)))).sum(dim=[1,2,3])

        else:
            z = z**power
            log_det_jacob += (torch.log(torch.abs(power*z**(power-1)))).sum(dim=[1,2,3])

        return z, log_det_jacob
```

where the parameters learned are the magnitude and sign of the power the prior distribution should be raised to instead of the shift and scale of the distribution.

Note that even though these additional flow transformations increase the expressivity of the normalizing flow significantly, they still do not match the universality in the mathematical derivation. For that, more complicated flow transformations have to be utilized, such as cubic splines, which will be the continuation of this work going into my PhD.

4. RESULTS

4.1. *Bayesian Hierarchical Discussion of Normalizing Flows*

—Throughout this work I have extensively talked about the modus operandi of Bayesian statistics and the mathematics behind normalizing flows, however never the latter in the context of the former. By formulation, in calculating the posterior distribution of any parameters of interest, the Bayesian statistics approach requires the introduction of all variables present, subsequently followed by the integration over all variables the parameters only depend on by association. Thus, the posterior of a set of a hypothetical parameters $\vec{\theta}$ dependent on some data $\vec{x}$ can be written as follows

$$P(\vec{\theta}|\vec{x}) = \int P(\vec{\theta}, \vec{f}, \vec{\mu}, \vec{\sigma}|\vec{x})\, \mathrm{d}\vec{f}\, \mathrm{d}\vec{\mu}\, \mathrm{d}\vec{\sigma}$$

where $\vec{f}$ represents the unknown variables of the flow (such as the weights of the neural network, their outputs etc.), $\vec{\mu}$ and $\vec{\sigma}$ represent the mean and covariance matrices of the data $\vec{x}$ respectively. As alluded to above, they are introduced and integrated over their full assumed support (aka. this in practice would be a definite integral, albeit left here indefinite for legibility purposes). From here, we can go to the full joined probability distribution

$$P(\vec{\theta}|\vec{x}) = \int \frac{1}{\epsilon(\vec{x})} P(\vec{\theta}, \vec{f}, \vec{\mu}, \vec{\sigma}, \vec{x})\, \mathrm{d}\vec{f}\, \mathrm{d}\vec{\mu}\, \mathrm{d}\vec{\sigma}$$

via introducing the evidence of the data, $\epsilon(\vec{x})$. Then, continuing in a familiar fashion

$$P(\vec{\theta}|\vec{x}) = \int \frac{1}{\epsilon(\vec{x})}\ P_{NF}(\vec{x}|\vec{f}, \vec{\mu}, \vec{\sigma}, \vec{\theta})\ \pi(\vec{f}, \vec{\mu}, \vec{\sigma}, \vec{\theta})\ \ \mathrm{d}\vec{f}\, \mathrm{d}\vec{\mu}\, \mathrm{d}\vec{\sigma}$$

we can label the first probability P_{NF} as the output of the normalizing flow. This step leads us to the major hurdle of this implementation. The posterior calculation seems to require a prior $\pi(\vec{f}, \vec{\mu}, \vec{\sigma}, \vec{\theta})$ on the flow variables $\vec{f}$. As the variables of the flow are hard to reach and even harder to predict analytically, it is a significant challenge to come up with a prior distribution that accurately predicts the numerical values these flow variables can take. Admitting the total lack of knowledge and assuming a uniform prior would also be dangerous, as for example, variables such as neural network weights are controlled against large absolute values for stability. This choice then would end up biasing the results against most likely values of $\vec{f}$ unnecessarily. I will deal with this hurdle momentarily, but this requires some other assumptions to be made about the formulation first.

We can split the joint prior into constituents as follows

$$\begin{aligned} P(\vec{\theta}|\vec{x}) &= \int \frac{1}{\epsilon(\vec{x})}\ P_{NF}(\vec{x}|\vec{f}, \vec{\mu}, \vec{\sigma}, \vec{\theta})\ \pi(\vec{\theta}|\vec{f}, \vec{\mu}, \vec{\sigma})\ \pi(\vec{f}, \vec{\mu}, \vec{\sigma})\ \ \mathrm{d}\vec{f}\, \mathrm{d}\vec{\mu}\, \mathrm{d}\vec{\sigma} \\ &= \int \frac{1}{\epsilon(\vec{x})}\ P_{NF}(\vec{x}|\vec{f}, \vec{\mu}, \vec{\sigma}, \vec{\theta})\ \pi(\vec{\theta}|\vec{\mu}, \vec{\sigma})\ \pi(\vec{f}, \vec{\mu}, \vec{\sigma})\ \ \mathrm{d}\vec{f}\, \mathrm{d}\vec{\mu}\, \mathrm{d}\vec{\sigma} \\ &= \int \frac{1}{\epsilon(\vec{x})}\ P_{NF}(\vec{x}|\vec{f}, \vec{\mu}, \vec{\sigma}, \vec{\theta})\ \pi(\vec{\theta}, \vec{\mu}, \vec{\sigma})\ \pi(\vec{f}|\vec{\mu}, \vec{\sigma})\ \ \mathrm{d}\vec{f}\, \mathrm{d}\vec{\mu}\, \mathrm{d}\vec{\sigma} \end{aligned}$$

where we have assumed the true parameters of interest $\vec{\theta}$ are independent of the flow variables $\vec{f}$, and recombined $\pi(\vec{\theta}, \vec{\mu}, \vec{\sigma}) = \pi(\vec{\theta}|\vec{\mu}, \vec{\sigma})\pi(\vec{\mu}, \vec{\sigma})$ in the final step. This leaves us with a final dependence $\pi(\vec{f}|\vec{\mu}, \vec{\sigma})$ to be resolved.

—Now, I argue that there exists a function I will label h that takes as input $\vec{\mu}$ and $\vec{\sigma}$ and returns the flow variables $\vec{f}$. One can justify the existence of such a function h via considering how the normalizing flow learns its variables:

- The normalizing flow training is achieved via minimizing the negative log-likelihood of the training data, where the variable of interest are the coefficients of the transformations of the normalizing flow. This is a classical minimization problem, and there exists a unique set of coefficients that minimize the negative log-likelihood that can be found in a reproducible manner for a given set of data (assuming the minimization algorithm is robust).
- The transformation coefficients are predicted using a set of neural networks that take the data as input. Assuming that the coefficients are unique by above for a given set of data, there must also exist a mapping between the data and the coefficients. This implies that there exists a unique set of flow variables (for example network weights given a pre-determined architecture) that return the transformation coefficients when the data in inputted.
- Thus, for a given normalizing flow objective and set of data, there exists a unique set of transformation coefficients and a unique set of network weights, or more generally, flow variables $\vec{f}$.

Thus I justify the existence of such a function $h(\vec{\mu}, \vec{\sigma})$. Then, we can rewrite $\pi(\vec{f}|\vec{\mu}, \vec{\sigma})$ as $\delta(\vec{f} - h(\vec{\mu}, \vec{\sigma}))$. While the function h is necessitated by the existence of a mapping, the Dirac delta is necessitated by the uniqueness of the solution as discussed above. Then, continuing from where we left off and utilizing this discussion

$$
\begin{aligned}
P(\vec{\theta}|\vec{x}) &= \int \frac{1}{\epsilon(\vec{x})}\ P_{NF}(\vec{x}|\vec{f}, \vec{\mu}, \vec{\sigma}, \vec{\theta})\ \pi(\vec{\theta}, \vec{\mu}, \vec{\sigma})\ \pi(\vec{f}|\vec{\mu}, \vec{\sigma})\ \mathrm{d}\vec{f}\,\mathrm{d}\vec{\mu}\,\mathrm{d}\vec{\sigma} \\
&= \int \frac{1}{\epsilon(\vec{x})}\ P_{NF}(\vec{x}|\vec{f}, \vec{\mu}, \vec{\sigma}, \vec{\theta})\ \pi(\vec{\theta}, \vec{\mu}, \vec{\sigma})\ \delta(\vec{f} - h(\vec{\mu}, \vec{\sigma}))\ \mathrm{d}\vec{f}\,\mathrm{d}\vec{\mu}\,\mathrm{d}\vec{\sigma} \\
&= \int \frac{1}{\epsilon(\vec{x})}\ P_{NF}(\vec{x}|h(\vec{\mu}, \vec{\sigma}), \vec{\mu}, \vec{\sigma}, \vec{\theta})\ \pi(\vec{\theta}, \vec{\mu}, \vec{\sigma})\ \mathrm{d}\vec{\mu}\,\mathrm{d}\vec{\sigma} \\
&= \int \frac{1}{\epsilon(\vec{x})}\ P_{NF}(\vec{x}|\vec{\mu}, \vec{\sigma}, \vec{\theta})\ \pi(\vec{\theta}, \vec{\mu}, \vec{\sigma})\ \mathrm{d}\vec{\mu}\,\mathrm{d}\vec{\sigma}
\end{aligned}
$$

where I have used the Dirac delta to eliminate the integral $\mathrm{d}\vec{f}$ and subsequently eliminated $h(\vec{\mu}, \vec{\sigma})$ from the right-hand side of the normalizing flow output as $\vec{\mu}$ and σ are already included. Thus it is possible to completely eliminate the dependence on the flow parameters in this Bayesian problem, ensuring that the probabilities returned by the normalizing flows can be used, completely avoiding the aforementioned hurdle without having to make a difficult decision about the priors on the flow variables. This is a key result in implementing normalizing flow-driven probabilities in Bayesian models where probability the likelihood distribution of the data is not analytically available.

4.2. *Simple Parameter Inference with Normalizing Flows*

—In the previous section I have discussed the theoretical proof of integrating normalizing flows into a Bayesian hierarchical model. The aim of this discussion was to show that it is possible to use normalizing flows in Bayesian integration, as the parameters introduced by the normalizing flows can be represented as a function of the data (or its summary statistics), and this can be canceled out of the integral, as $\delta(\vec{f} - h(\vec{\mu}, \vec{\sigma}))$. If this formulation holds, then the probabilities returned by the normalizing flow trained on some data set $P_{NF}(\vec{x}|\vec{f}, \vec{\mu}, \vec{\sigma}, \vec{\theta})$ would be unbiased, or in other words, would be the same probabilities with the case where we had access to the analytic form of the function. This claim is possible to test thanks to one of the fundamental strengths of normalizing flows.

As the flow transformations are diffeomorphisms, it is possible to run the flow in the reverse direction and generate new data samples from it, as discussed in Section 3. If the flow has been trained to the point where the above assumption holds, then these samples, granted they are large enough, would follow the sampling theorem and have the same summary statistics as the training data set and the analytic distribution. And the requirement that $P_{NF}(\vec{x}|\vec{f}, \vec{\mu}, \vec{\sigma}, \vec{\theta})$ be unbiased is true if and only if the generated samples from the flow also follow the sampling theorem for the analytic

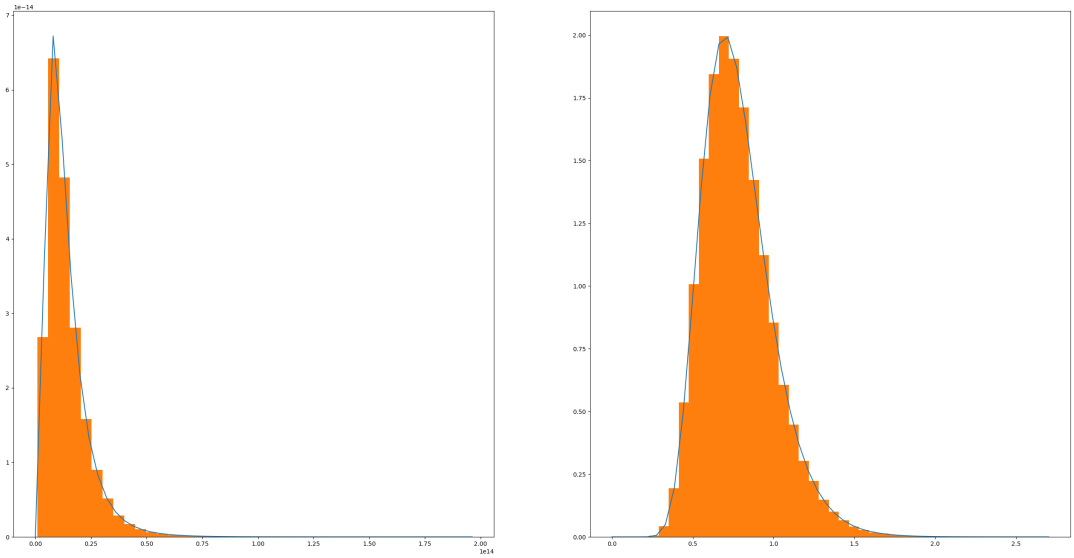

Figure 8. Parameter fitting to the data generated from the normalizing flow to recover the summary statistics and θ.

distribution they come from. Thus, I formulate and present a test of my previous claim as follows.

First, create a dummy "physical" parameter θ that is distributed as

$$\vec{\theta} \sim \mathbb{G}\left([1.47, 0, 47], \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}\right)$$

Then, assume that there exists a function f that maps the dummy parameter to the mean of the observable data μ

$$\vec{\mu} = f(\vec{\theta})$$

where in this example I take $f(\vec{\theta}) = [\theta_1^3, \theta_2^3]$ where θ_i represent individual elements of the the $\vec{\theta}$ vector. Finally, generate the observable data with this

$$\vec{x} \sim \text{lognorm}(\vec{\mu}, \text{I})$$

The normalizing flow is trained on this data set $\vec{x}$. Once it has been trained, a new data set is sampled from the trained flow, $\vec{z}$. Now, we can modify the original requirement for this test to read that, if the summary statistics of $\vec{z}$ and the associated $\vec{\theta}$ match the original analytic distribution, the normalizing flow introduces no biases to the Bayesian hierarchical model.

In every individual case, a dummy parameter θ is sampled from the prior distribution, the mean are generated with the function mapping f, and a data set is created from the data prior. Then, the normalizing flow is trained on this data set. The trained flow is then used to generate a new sample of data, the marginalized distributions of which can be seen in Figure 8. Then, these marginalized samples are used to fit to the mean and the standard deviation of the data, which allows for the recovery of the $\vec{\theta}$ parameter through inverting f. Please note that in "real-life" use cases of normalizing flows, this is of course not feasible, as the relation function f cannot be assumed to be available. But this information is precisely what makes this a case to test for any biases that can be introduced, as every other step of the experiment other than the normalizing flows are controlled.

Figure 9 shows the results of an extended study of this simple parameter inference experiment, which consists of over 540 repetitions of the experiment described in the previous paragraph. Most of the recovered $(\vec{\theta})$ lie very close to the peak, with the densest region of the scatter lying exactly on top of the analytic peak of the distribution at $[1.47, 0.47]$. Considering both the location and the shape of the sample distribution, it is safe to conclude that the

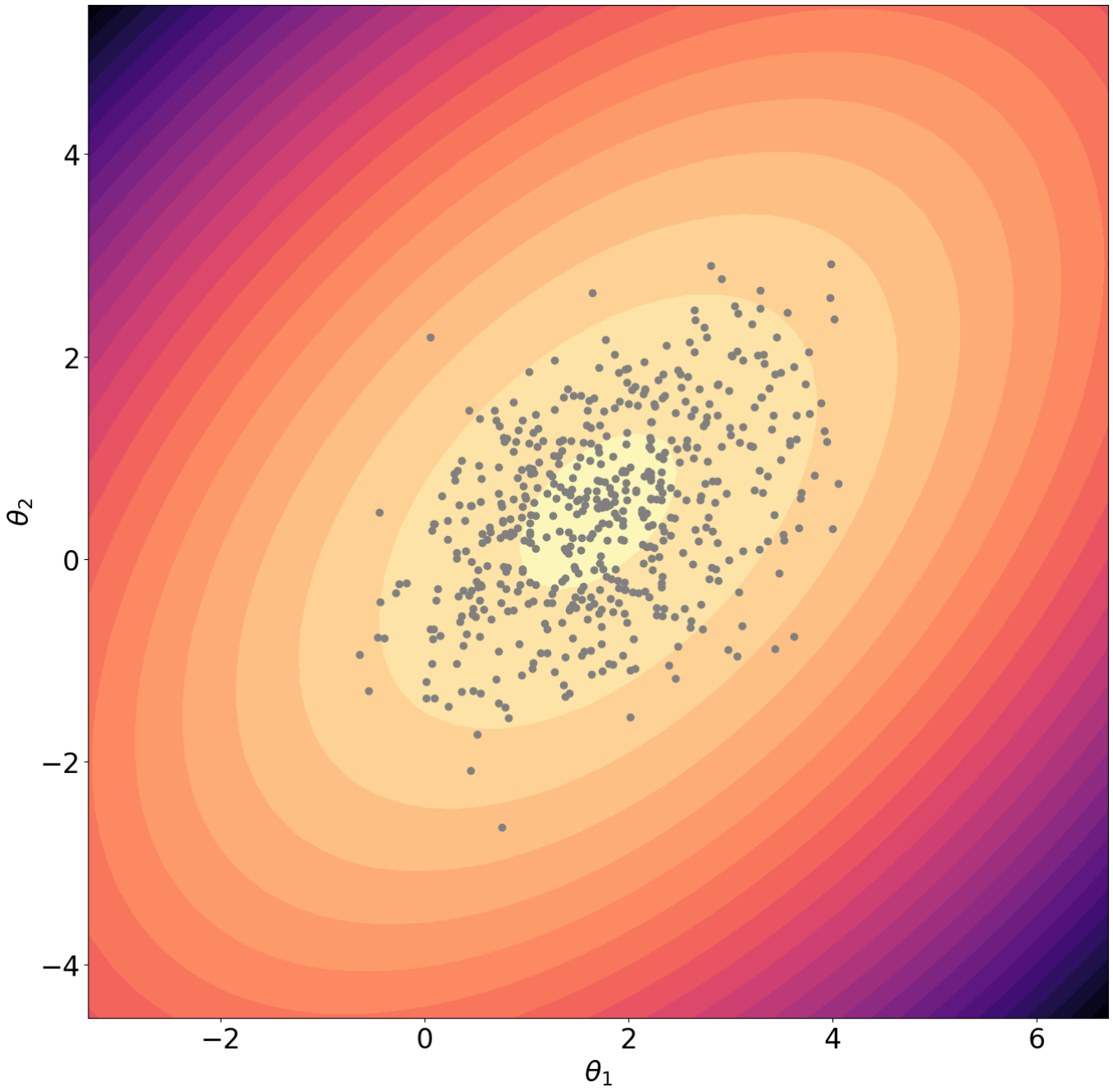

Figure 9. The result of a simple parameter inference test case with normalizing flows to investigate the possibility of any biases incurred due to possible missing priors on the flow parameters. The plot shows, in color, the analytic prior contour for the dummy physical parameter $\vec{\theta}$, and the overplotted grey scatter points are the $\vec{\theta}$ recovered from the normalizing flow.

above mathematical formulation is correct, and that the normalizing flows incur no biases in the Bayesian hierarchical model integration.

5. CONCLUSIONS

—Light emitted by distant galaxies interact with matter along their path, and observations show a systematic distortion of the shapes of these distant galaxies from what should be a uniform distribution of shapes and orientations. Thus, studies of this weak lensing phenomenon use this shear field to measure the three-dimensional matter distribution in the universe in a model-agnostic way, which allows for considerable improvement of the estimates we currently have of

cosmological parameters that are sensitive of the mass distribution of the universe such as the Hubble constant H_0, matter density parameter Ω_m and matter fluctuation amplitude σ_8.

ALMANAC aims to create a Bayesian hierarchical model to infer the cosmic shear fields and their power spectra independent of cosmological models assumed, which ensures the inference methods used for upcoming weak lensing surveys such as LSST and Euclid are fault-proof. However, ALMANAC priors show that the marginalized one-dimensional posteriors for the low multipole ($l < 50$) E-modes of the weak lensing power spectra are highly non-Gaussian. This violates the Gaussianity assumption in the Bayesian hiererchical model. Thus this thesis aims to understand and model these functions without having to rely on the many-dimensional full ALMANAC prior.

To achieve this, this thesis develops and implements normalizing flows, which allow for mapping of unknown probability distributions to known ones through learnable variable transformations. In this work, I have presented the code base I have put together to build and train normalizing flows for Bayesian parameter inference. The modularity of the code presented allows for convenient implementation of various flow transformations. This means that, even though the expressive capability of the flow tranformations yet implemented is limited compared to the desired scope for modeling the ALMANAC posterior (which is mainly a computational challenge), the expansion to achieve that should not require the overhauling of the already existing framework, but mainly building on it.

In addition to the code developed, this work has addressed the concerns with using normalizing flows in a Bayesian hierarchical model, specifically speaking the introduction of a possible bias in the posterior integration due to the emergence of new nuisance parameters, the hidden parameters the flow uses to transform the data. The mathematical formulation, even though it makes some simplifying assumptions about the underlying likelihood minimization via gradient descent using neural networks, the controlled parameter inference case results presented show that no such bias is sustained in repeated experimentation with flows.

Finally, the next step in expanding this project is expected to be twofold. The first part is the complete implementation of cubic splines, which seems to have issues with its gradient descent in finding its parameters. Fixing this should increase the expressivity of the flows, allowing for a broader range of functions to be mapped to. The second part is the computational fine-tuning. The sample inference case has proven an expected conclusion, that normalizing flows are expensive to train due to the latent space representation sharing the dimensionality of the data. Thus, minimizing the computational time requires fine tuning of the code base to minimize unnecessary overhead as much as possible, which will be the second focus as the project moves to working with the ALMANAC data.

APPENDIX

A. SIMPLE FLOW++ CNN WITH GATED RESNET

```python
#Here is the NN that learns the affine transform for the flows. Insipred by Flow++.
import torch
import torch.nn as nn
import torch.nn.functional as f

class ConcatELU(nn.Module):
    def forward(self, x):
        output = torch.cat([
            f.elu(x),
            f.elu(-x),
        ], dim = 1)
        return output

class LayerNormChannels(nn.Module):
    def __init__(self, input_channels, epsilon = 1e-5) -> None:
        super().__init__()
        self.gamma = nn.Parameter(torch.ones(1, input_channels, 1, 1))
        self.beta = nn.Parameter(torch.zeros(1, input_channels, 1, 1))
        self.epsilon = epsilon

    def forward(self, x):
        mean = x.mean(dim=1, keepdim=True)
        var = x.var(dim=1, unbiased=False, keepdim=True)
        y = (x - mean) / torch.sqrt(var + self.epsilon)
        y = y * self.gamma + self.beta
        return y

class GatedConvolution(nn.Module):
```

```
    def __init__(self, input_channels, hidden_variables) -> None:
        super().__init__()
        self.net = nn.Sequential(
            ConcatELU(),
            nn.Conv2d(2*input_channels, hidden_variables, kernel_size=3, padding=1),
            ConcatELU(),
            nn.Conv2d(2*hidden_variables, 2*input_channels, kernel_size=1)
        )

    def forward(self, x):
        out = self.net(x)
        val, gate = out.chunk(2, dim=1)
        return x + val * torch.sigmoid(gate)

class GatedConvNet(nn.Module):
    def __init__(self, input_channels, hidden_variables = 32, output_channels = None, num_layers = 3) -> None:
        super().__init__()
        output_channels = output_channels if output_channels is not None else 2*input_channels

        layers = []
        layers += [nn.Conv2d(input_channels, hidden_variables, kernel_size=3, padding=1)]
        for index in range(num_layers):
            layers += [
                GatedConvolution(hidden_variables, hidden_variables),
                LayerNormChannels(hidden_variables)
            ]
        layers += [
            ConcatELU(),
            nn.Conv2d(2*hidden_variables, output_channels, kernel_size=3, padding=1)
        ]

        self.net = nn.Sequential(*layers)
        self.net[-1].weight.data.zero_()
        self.net[-1].bias.data.zero_()

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

B. TRAINING THE FLOW WITH PYTORCH LIGHTNING

```
#This file contains funcitons for training and testing.
import os
import time
import sys

import torch
import torch.utils.data as data

import pytorch_lightning as pl
from pytorch_lightning.callbacks import LearningRateMonitor, ModelCheckpoint

def train_flow(flow, epochs, train_set, validation_set, test_set, model_name):
    MODEL_PATH = os.path.join(sys.path[0], 'models')
    N_DEVICES = 1
    NUM_NODES = 1

    trainer = pl.Trainer(
        default_root_dir = os.path.join(MODEL_PATH, model_name),
        accelerator = 'gpu' if torch.cuda.is_available() else None,
        devices = N_DEVICES if torch.cuda.is_available() else 0,
        num_nodes = NUM_NODES,
        max_epochs = epochs,
        #strategy = 'ddp', #Enable on ALICE.
        gradient_clip_val = 0.5,
        callbacks = [
            ModelCheckpoint(save_weights_only=True, mode='min', monitor = 'validation_bits_per_dim'),
            LearningRateMonitor('epoch')
        ],
        check_val_every_n_epoch = 3
    )

    train_data_loader = data.DataLoader(train_set, batch_size=128, shuffle=True, drop_last=True, pin_memory=True, num_workers=0)
    validation_data_loader = data.DataLoader(validation_set, batch_size=64, shuffle=False, drop_last=False, num_workers=0)
    test_data_loader = data.DataLoader(test_set, batch_size=64, shuffle=False, drop_last=False, num_workers=0)

    trainer.fit(flow, train_data_loader, validation_data_loader)

    validation_result = trainer.test(flow, validation_data_loader, verbose = False)
    start_time = time.time()
```

```
        test_result = trainer.test(flow, test_data_loader, verbose = False)
        duration = time.time() - start_time
        result = {"test": test_result, "val": validation_result, "time": duration / len(test_data_loader) / flow.importance_samples}

        return flow, result
```

REFERENCES

Alsing, J., Heavens, A., Jaffe, A. H., et al. 2016, MNRAS, 455, 4452, doi: 10.1093/mnras/stv2501

Bartelmann, M., & Schneider, P. 2001, PhR, 340, 291, doi: 10.1016/S0370-1573(00)00082-X

Castro, P. G., Heavens, A. F., & Kitching, T. D. 2005, PhRvD, 72, 023516, doi: 10.1103/PhysRevD.72.023516

Dai, B., & Seljak, U. 2023, arXiv e-prints, arXiv:2306.04689, doi: 10.48550/arXiv.2306.04689

Hall, A. 2021, Monthly Notices of the Royal Astronomical Society, 505, 4935, doi: 10.1093/mnras/stab1563

Heavens, A. 2009, Nuclear Physics B - Proceedings Supplements, 194, 76, doi: https://doi.org/10.1016/j.nuclphysbps.2009.07.005

Ho, J., Chen, X., Srinivas, A., Duan, Y., & Abbeel, P. 2019, arXiv e-prints, arXiv:1902.00275, doi: 10.48550/arXiv.1902.00275

Kaiser, N. 1998, ApJ, 498, 26, doi: 10.1086/305515

Laureijs, R., Amiaux, J., Arduini, S., et al. 2011, arXiv e-prints, arXiv:1110.3193, doi: 10.48550/arXiv.1110.3193

Lim, S. H., Putney, E., Buckley, M. R., & Shih, D. 2023, arXiv e-prints, arXiv:2305.13358, doi: 10.48550/arXiv.2305.13358

Limber, D. N. 1954, ApJ, 119, 655, doi: 10.1086/145870

Lippe, P. 2022, UvA Deep Learning Tutorials, https://uvadlc-notebooks.readthedocs.io/en/latest/

Loureiro, A., Whiteway, L., Sellentin, E., et al. 2023, The Open Journal of Astrophysics, 6, 6, doi: 10.21105/astro.2210.13260

LSST Science Collaboration, Abell, P. A., Allison, J., et al. 2009, arXiv e-prints, arXiv:0912.0201, doi: 10.48550/arXiv.0912.0201

Miyazaki, S., Hamana, T., Shimasaku, K., et al. 2002, ApJL, 580, L97, doi: 10.1086/345613

Newman, E. T., & Penrose, R. 1966, Journal of Mathematical Physics, 7, 863, doi: 10.1063/1.1931221

Papamakarios, G., Nalisnick, E., Jimenez Rezende, D., Mohamed, S., & Lakshminarayanan, B. 2019, arXiv e-prints, arXiv:1912.02762, doi: 10.48550/arXiv.1912.02762

Planck Collaboration, Aghanim, N., Akrami, Y., et al. 2020, A&A, 641, A6, doi: 10.1051/0004-6361/201833910

Tran, H. N., Li, Y., Siu, S., et al. 2013, Perm. J., 17, 23

Wen, J., Ahmad, R., & Schniter, P. 2023, arXiv e-prints, arXiv:2306.01630, doi: 10.48550/arXiv.2306.01630