BAYESFLOW: LEARNING COMPLEX STOCHASTIC MODELS WITH INVERTIBLE NEURAL NETWORKS

Stefan T. Radev
Institute of Psychology
Heidelberg University
Hauptstr. 47-51, 69117 Heidelberg
stefan.radev93@gmail.com

Ulf K. Mertens
Institute of Psychology
Heidelberg University
Hauptstr. 47-51, 69117 Heidelberg
mertens.ulf@gmail.com

Andreas Voss
Institute of Psychology
Heidelberg University
Hauptstr. 47-51, 69117 Heidelberg
andreas.voss@psychologie.uni-heidelberg.de

Lynton Ardizzone
Visual Learning Lab, IWR
Heidelberg University
Im Neuenheimer Feld 205, 69120 Heidelberg
lynton.ardizzone@iwr.uni-heidelberg.de

Ullrich Köthe
Visual Learning Lab, IWR
Heidelberg University
Im Neuenheimer Feld 205, 69120 Heidelberg
ullrich.koethe@iwr.uni-heidelberg.de

April 20, 2020

ABSTRACT

Estimating the parameters of mathematical models is a common problem in almost all branches of science. However, this problem can prove notably difficult when processes and model descriptions become increasingly complex and an explicit likelihood function is not available. With this work, we propose a novel method for globally amortized Bayesian inference based on invertible neural networks which we call BayesFlow. The method uses simulation to learn a global estimator for the probabilistic mapping from observed data to underlying model parameters. A neural network pre-trained in this way can then, without additional training or optimization, infer full posteriors on arbitrary many real data sets involving the same model family. In addition, our method incorporates a summary network trained to embed the observed data into maximally informative summary statistics. Learning summary statistics from data makes the method applicable to modeling scenarios where standard inference techniques with hand-crafted summary statistics fail. We demonstrate the utility of BayesFlow on challenging intractable models from population dynamics, epidemiology, cognitive science and ecology. We argue that BayesFlow provides a general framework for building reusable Bayesian parameter estimation machines for any process model from which data can be simulated.

Keywords Deep learning · Generative learning · Bayesian inference · Parameter estimation · Stochastic models

1 Introduction

The goal of Bayesian analysis is to infer the underlying characteristics of some natural process of interest given observable manifestations x. In a Bayesian setting, we assume that we already posses sufficient understanding of the forward problem, that is, a suitable model of the mechanism that generates observations from a given configuration of the hidden parameters θ. This knowledge can be available in two forms. In likelihood-based approaches, we are able to

evaluate the likelihood function $p(\boldsymbol{x} \mid \boldsymbol{\theta})$ analytically or numerically for any pair $(\boldsymbol{x}, \boldsymbol{\theta})$. In contrast, likelihood-free approaches only require the ability to sample from the likelihood. This can be done either directly or, equivalently, by simulating data from some forward model, which generates instances of the observables via a deterministic function g of parameters $\boldsymbol{\theta}$ and independent noise ξ:

$$\boldsymbol{x}_i \sim p(\boldsymbol{x} \mid \boldsymbol{\theta}) \quad \Longleftrightarrow \quad \boldsymbol{x}_i = g(\boldsymbol{\theta}, \xi_i) \text{ with } \xi_i \sim p(\xi) \tag{1}$$

In this case, the likelihood function is only defined implicitly, and we cannot calculate its actual numeric value for the resulting instances $\boldsymbol{x}_i$, which, in turn, prohibits standard statistical inference. Likelihood-free problems arise, for example, when $p(\boldsymbol{x} \mid \boldsymbol{\theta})$ is not available in closed-form, or when the forward process is defined by a stochastic differential equation, a computer simulation, or a complicated algorithm [32, 54, 53, 58]. In this paper, we propose a new Bayesian solution to the likelihood-free setting in terms of *invertible neural networks*.

Bayesian modeling leverages the available knowledge about the forward problem to get the best possible estimate of the posterior distribution of the inverse problem

$$p(\boldsymbol{\theta} \mid \boldsymbol{x}_{1:N}) = \frac{p(\boldsymbol{x}_{1:N} \mid \boldsymbol{\theta}) \, p(\boldsymbol{\theta})}{\int p(\boldsymbol{x}_{1:N} \mid \boldsymbol{\theta}) \, p(\boldsymbol{\theta}) \, d\boldsymbol{\theta}}$$

In Bayesian inference, the posterior encodes all information about $\boldsymbol{\theta}$ obtainable from a set of observations $\boldsymbol{x}_{1:N} = \{\boldsymbol{x}_i\}_{i=1}^{N}$. The observations are assumed to arise from N runs of the forward process with fixed, but unknown, true parameters $\boldsymbol{\theta}^*$. Bayesian inverse modeling is challenging for three reasons:

1. The RHS of Bayes' formula above is always intractable in the likelihood-free case and must be approximated.
2. The forward process is usually non-deterministic, so that there is intrinsic uncertainty about the true value of $\boldsymbol{\theta}$.
3. The forward process is typically not information-preserving, so that there is ambiguity among possible values of $\boldsymbol{\theta}$.

The standard solution to these problems is offered by *approximate Bayesian computation* (ABC) methods [51, 10, 44, 53]. ABC methods approximate the posterior by repeatedly sampling parameters from a proposal (prior) distribution $\boldsymbol{\theta}^{(l)} \sim p(\boldsymbol{\theta})$ and then simulating multiple data sets by running the forward model $\boldsymbol{x}_i \sim p(\boldsymbol{x} \mid \boldsymbol{\theta}^{(l)})$ for $i = 1...N$. If the resulting data set is sufficiently similar to the actually observed data set $\boldsymbol{x}_{1:N}^o$, the corresponding $\boldsymbol{\theta}_k$ is retained as a sample from the desired posterior, otherwise rejected. Stricter similarity criteria lead to more accurate approximations of the desired posterior at the price of higher and oftentimes prohibitive rejection rates.

More efficient methods for approximate inference, such as sequential Monte Carlo (ABC-SMC), Markov-Chain Monte Carlo variants [50], or the recent sequential neural posterior estimation methods (SNPE) [20, 43, 35], optimize sampling from a proposal distribution in order to balance the speed-accuracy trade-off of vanilla ABC methods. More details about these methods can be found in the section **Related Work** or in the excellent review by [9].

All sampling methods described above operate on the level of individual data sets: For each observation sequence $\boldsymbol{x}_{1:N}$, the entire estimation procedure for the posterior must be run again from scratch. Therefore, we refer to this approach as *case-based inference*. Running estimation for each individual data set separately stands in contrast to *amortized inference*, where estimation is split into a potentially expensive *upfront* training phase, followed by a much cheaper inference phase. The goal of the upfront training phase is to learn an approximate posterior $\widehat{p}(\boldsymbol{\theta} \mid \boldsymbol{x}_{1:N})$ that works well for *any* observation sequence $\boldsymbol{x}_{1:N}$. Evaluating this model for specific observations $\boldsymbol{x}_{1:N}^o$ is then very fast, so that the training effort amortizes over repeated evaluations (see Figure 1 for a graphical illustration). The break-even between case-based and amortized inference depends on the application and model types, and we will report comparisons in the experimental section. Our main aim in this paper, however, is the introduction of a general approach to amortized Bayesian inference and the demonstration of its excellent accuracy in posterior estimation for a variety of popular forward models.

To make amortized inference feasible in practice, it must work well for arbitrary data set sizes N. Depending on data acquisition circumstances, the number of available observations for a fixed model parameter setting may vary considerably, ranging from $N = 1$ to several hundreds and beyond. This has not only consequences for the required architecture of our density approximators, but also for their behavior: They must exhibit correct *posterior contraction*. Accordingly, the estimated posterior $\widehat{p}(\boldsymbol{\theta} \mid \boldsymbol{x}_{1:N})$ should get sharper (i.e., more peaked) as the number N of available observations increases. In the simplest case, the posterior variance should decrease at rate $1/N$, but more complex behavior can occur for difficult (e.g. multi-modal) true posteriors $p(\boldsymbol{\theta} \mid \boldsymbol{x}_{1:N})$.

We incorporate these considerations into our method by integrating two separate deep neural networks modules (detailed in the **Methods** section; see also Figure 1), which are trained jointly on simulated data from the forward model: a *summary network* and an *inference network*.
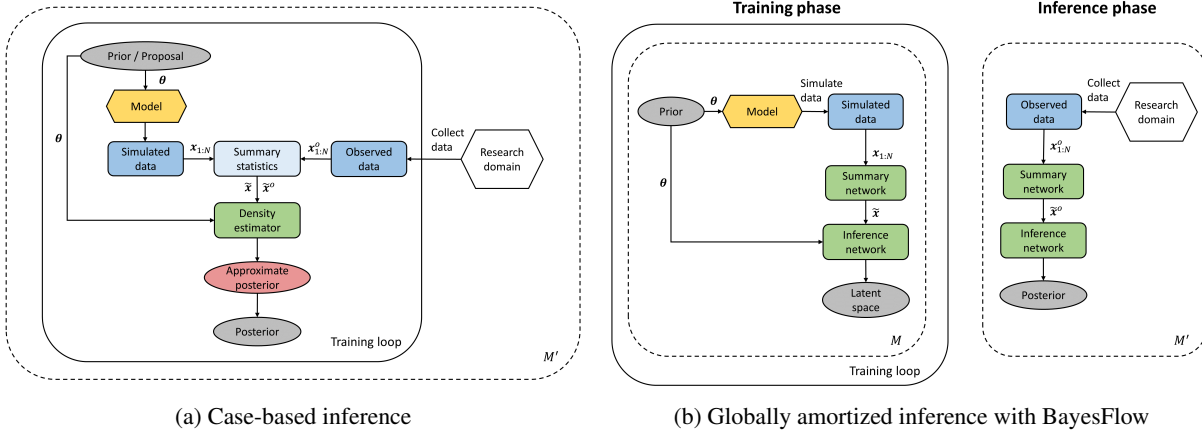
(a) Case-based inference   (b) Globally amortized inference with BayesFlow

Figure 1: Graphical illustration of the main differences between case-based (neural) density estimation methods and BayesFlow. (**a**) Case-based methods require a separate optimization loop for each observed data set from a given research domain. When case-based methods incorporate a training phase (e.g., SNPE with iterative proposal refinement), it must be repeated for each new data set. Summary statistics are manually selected and may thus be sub-optimal; (**b**) BayesFlow incorporates a global upfront training (before any real data are collected) via simulations from the forward model (left panel). Summary and inference network are trained jointly, resulting in higher accuracy than hand-crafted summary statistics. In the inference phase (right panel), BayesFlow works entirely in a feed-forward manner, that is, no training or optimization happens in this phase. The upfront training effort is therefore amortized over arbitrary many observed data sets from a research domain working on the same model family. Note that the solid and dashed plates are swapped between case-based Bayesian inference and the training phase of BayesFlow.

The *summary network* is responsible for reducing a set of observations $x_{1:N}$ of variable size to a fixed-size vector of *learned* summary statistics. In traditional likelihood-free approaches, the method designer is responsible for selecting suitable statistics for each application *a priori* [38, 37, 48, 51]. In contrast, our summary networks learn the most informative statistics directly from data, and we will show experimentally (see **Experiment 3.7**) that these statistics are superior to manually constructed ones. Summary networks differ from standard feed-forward networks because they should be independent of the input size N and respect the inherent functional and probabilistic symmetries of the data. For example, permutation invariant networks are required for *i.i.d.* observations [5], and recurrent networks [17] or convolutional networks [34] for data with temporal or spatial dependencies.

The *inference network* is responsible for learning the true posterior of model parameters given the summary statistics of the observed data. Since it sees the data only through the lens of the summary network, all symmetries captured by the latter are automatically inherited by the posterior. We implement the inference network as an *invertible neural network*. Invertible neural networks are based on the recent theory and applications of *normalizing flows* [3, 31, 22, 14, 29]. Flow-based methods can perform asymptotically exact inference and scale favourably from simple low-dimensional problems to high-dimensional distributions with complex dependencies, for instance, the pixels of an image [31]. For each application/model of interest, we train an invertible network jointly with a corresponding summary network using simulated data from the respective known forward process with reasonable priors. After convergence of this forward training, the network's invertibility ensures that a model for the inverse process is obtained for free, simply by running inference through the model backwards. Thus, our networks can perform fast amortized Bayesian inference on arbitrary many data sets from a given application domain without expensive case-based optimization. We call our method *BayesFlow*, as it combines ideas from Bayesian inference and flow-based deep learning.

BayesFlow draws on major advances in modern deep probabilistic modeling, also referred to as deep generative modeling [5, 31, 2, 30]. A hallmark idea in deep probabilistic modeling is to represent a complicated target distribution as a non-linear bijective transformation of some simpler latent distribution (e.g., Gaussian or uniform), a so called *pushforward*. Density estimation of the target distribution, a very complex problem, is thus reduced to learning a non-linear transformation, a task that is ideally suited for gradient-based neural network training via standard backpropagation. During the *inference phase*, samples from the target distribution are obtained by sampling from the simpler latent distribution and applying the inverse transformation learned during the training phase (see Figure 1b for a high-level overview). Using this approach, recent applications of deep probabilistic models have achieved unprecedented performance on hitherto intractable high-dimensional problems [5, 31, 22].

In the context of Bayesian inference, the target distribution is the posterior $p(\boldsymbol{\theta} \mid \boldsymbol{x}_{1:N})$ of model parameters given observed data. We leverage the fact that we can simulate arbitrarily large amounts of training data from the forward model in order to ensure that the summary and invertible networks approximate the true posterior as well as possible. During the inference phase, our model can either numerically evaluate the posterior probability of any candidate parameter $\boldsymbol{\theta}$, or can generate a posterior sample $\boldsymbol{\theta}^{(1)}, \boldsymbol{\theta}^{(2)}, ..., \boldsymbol{\theta}^{(L)}$ of likely parameters for the observed data $\boldsymbol{x}_{1:N}^{o}$. In the **Methods** section, we show that our networks indeed sample from the correct posterior under perfect convergence. In summary, the contributions of our BayesFlow method are the following:

- Globally amortized approximate Bayesian inference with reusable invertible neural networks;

- Learning maximally informative summary statistics from raw data sets with variable number of observations instead of relying on restrictive hand-crafted summary statistics;

- Asymptotic theoretical guarantee for sampling from the true posterior distribution without assumptions on the distributional forms of the posterior and prior;

- Parallel computations applicable to both forward simulations and neural network optimization;

To illustrate the utility of BayesFlow, we first apply it to two toy models with analytically tractable posteriors. The first is a multivariate Gaussian with a full covariance matrix and a unimodal posterior. The second is a Gaussian mixture model with a multimodal posterior. Then, we present applications to challenging models with intractable likelihoods from population dynamics, cognitive science, epidemiology, and ecology and demonstrate the utility of BayesFlow in terms of speed, accuracy of recovery, and probabilistic calibration. Across the examples, we introduce multiple tools to validate the performance of our method.

1.1 Related Work

BayesFlow incorporates ideas from previous machine learning and deep learning approaches to likelihood-free inference [36, 46, 38, 48, 26]. The most common approach has been to cast the problem of parameter estimation as a supervised learning task. In this setting, a large data set of the form $\boldsymbol{D} = \{(\psi(\boldsymbol{x}_{1:N}^{(i)}), \boldsymbol{\theta}^{(i)})\}_{i=1}^{M}$ is created by repeatedly sampling from $p(\boldsymbol{\theta})$ and simulating an artificial datasets $\boldsymbol{x}_{1:N}$ by running $g(\boldsymbol{\theta}, \xi)$ with the sampled parameters. Usually, the dimensionality of the simulated data is reduced by computing summary statistics with a fixed summary function $\psi(\boldsymbol{x}_{1:N})$. Then, a supervised learning algorithm (e.g., random forest [48], or a neural network [46]) is trained on the summary statistics of the simulated data to output an estimate of the true data generating parameters. Thus, an attempt is made to *learn* the intractable inverse model $\boldsymbol{\theta} = g^{-1}(\boldsymbol{x}, \xi)$. A main shortcoming of supervised approaches is that they provide only limited information about the posterior (e.g., point-estimates, quantiles or variance estimates) or impose restrictive distributional assumptions on the shape of the posterior (e.g., normality).

Our ideas are also closely related to the concept of *optimal transport maps* and its application in Bayesian inference [13, 45, 7, 4]. A transport map defines a transformation between (probability) measures which can be constructed in a way to *warp* a simple probability distribution into a more complex one. In the context of Bayesian inference, transport maps have been applied to accelerate MCMC sampling [45], to perform sequential inference [13], and to solve inference problems via direct optimization [4]. In fact, BayesFlow can be viewed as a parameterization of invertible transport maps via invertible neural networks. An important distinction is that BayesFlow does not require an explicit likelihood function for approximating the target posteriors and is capable of amortized inference.

Similar ideas for likelihood-free inference are incorporated in the recent automatic posterior transformation (APT) [20], and the sequential neural likelihood (SNL) [43] methods. APT iteratively refines a proposal distribution via an expressive neural density estimator (e.g., an autoregressive flow network) to generate parameter samples which closely match a particular observed data set. SNL, in turn, trains a masked autoencoder density estimator (MADE) neural network within an MCMC loop to increase proposal acceptance rates and thus speed-up convergence to the true posterior. Even though these methods also entail a relatively expensive learning phase and a cheap inference phase, posterior inference is amortized only for a single data set. Note, that APT can be applied in a single-round regime which amortizes over multiple data sets, as in [18]. However, the learning phase for all neural density methods with sequential proposal refinement needs to be repeated for each individual data set (see Figure 1a). In contrast, we propose to learn the posterior globally over the entire range of plausible parameters and data sets by employing a conditional invertible neural network (cINN) estimator (see Figure 1b). Previously, INNs have been successfully employed to model data from astrophysics and medicine [2]. We adapt the model to suit the task of parameter estimation in the context of mathematical modeling and develop a reusable probabilistic architecture for fully Bayesian and globally amortized inference on complex mathematical models.

2 Methods

2.1 Notation

In the following, the number of parameters of a mathematical model will be denoted as D, and the number of observations in a data set as N. We denote data simulated from the mathematical model of interest as $\boldsymbol{x}_{1:N} = (\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_N)$, where each individual $\boldsymbol{x}_i$ can represent a scalar or a vector. Observed or test data will be marked with a superscript o (i.e., $\boldsymbol{x}^o_{1:N}$). . The parameters of a mathematical model are represented as a vector $\boldsymbol{\theta} = (\theta_1, \theta_2, ..., \theta_D)$, and all trainable parameters of the invertible and summary neural networks as $\boldsymbol{\phi}$ and $\boldsymbol{\psi}$, respectively. When a data set consists of observations over a period of time, the number of observations will be denotes as T.

2.2 Learning the Posterior

Assume that we have an invertible function $f_{\boldsymbol{\phi}} : \mathbb{R}^D \to \mathbb{R}^D$, parameterized by a vector of parameters $\boldsymbol{\phi}$, for which the inverse $f_{\boldsymbol{\phi}}^{-1} : \mathbb{R}^D \to \mathbb{R}^D$ exists. For now, consider the case when raw simulated data $\boldsymbol{x}_{1:N}$ of size $N = 1$ is entered directly into the invertible network without using a summary network. Our goal is to train an invertible neural network which approximates the true posterior as accurately as possible

$$p_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \boldsymbol{x}) \approx p(\boldsymbol{\theta} \,|\, \boldsymbol{x}) \tag{2}$$

for all $\boldsymbol{\theta}$ and $\boldsymbol{x}$. We reparameterize the approximate posterior $p_{\boldsymbol{\phi}}$ in terms of a conditional invertible neural network (cINN) $f_{\boldsymbol{\phi}}$ which implements a normalizing flow between $\boldsymbol{\theta}$ and a Gaussian latent variable $\boldsymbol{z}$:

$$\boldsymbol{\theta} \sim p_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \boldsymbol{x}) \Longleftrightarrow \boldsymbol{\theta} = f_{\boldsymbol{\phi}}^{-1}(\boldsymbol{z}; \boldsymbol{x}) \text{ with } \boldsymbol{z} \sim \mathcal{N}_D(\boldsymbol{z} \,|\, \boldsymbol{0}, \mathbb{I}) \tag{3}$$

Accordingly, we need to ensure that outputs of $f_{\boldsymbol{\phi}}^{-1}(\boldsymbol{z}; \boldsymbol{x})$ follow the target posterior $p(\boldsymbol{\theta} \,|\, \boldsymbol{x})$. Thus, we seek neural network parameters $\widehat{\boldsymbol{\phi}}$ which minimize the Kullback-Leibler (KL) divergence between the true and the model-induced posterior for all possible data sets $\boldsymbol{x}$. Therefore, our objective becomes:

$$\widehat{\boldsymbol{\phi}} = \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \, \mathbb{E}_{\boldsymbol{x} \sim p(\boldsymbol{x})} \left[\mathbb{KL}(p(\boldsymbol{\theta} \,|\, \boldsymbol{x}) \,||\, p_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \boldsymbol{x})) \right] \tag{4}$$

$$= \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \, \mathbb{E}_{\boldsymbol{x} \sim p(\boldsymbol{x})} \left[\mathbb{E}_{\boldsymbol{\theta} \sim p(\boldsymbol{\theta} \,|\, \boldsymbol{x})} \left[\log p(\boldsymbol{\theta} \,|\, \boldsymbol{x}) - \log p_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \boldsymbol{x}) \right] \right] \tag{5}$$

$$= \underset{\boldsymbol{\phi}}{\operatorname{argmax}} \, \mathbb{E}_{\boldsymbol{x} \sim p(\boldsymbol{x})} \left[\mathbb{E}_{\boldsymbol{\theta} \sim p(\boldsymbol{\theta} \,|\, \boldsymbol{x})} \left[\log p_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \boldsymbol{x}) \right] \right] \tag{6}$$

$$= \underset{\boldsymbol{\phi}}{\operatorname{argmax}} \int \int p(\boldsymbol{x}, \boldsymbol{\theta}) \log p_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \boldsymbol{x}) d\boldsymbol{x} d\boldsymbol{\theta} \tag{7}$$

Note, that the log posterior density $p(\boldsymbol{\theta} \,|\, \boldsymbol{x})$ can be dropped from the optimization objective in Eq.6, as it does not depend on the neural network parameters $\boldsymbol{\phi}$. In other words, we seek neural network parameters $\widehat{\boldsymbol{\phi}}$ which maximize the posterior probability of data-generating parameters $\boldsymbol{\theta}$ given observed data $\boldsymbol{x}$ for all $\boldsymbol{\theta}$ and $\boldsymbol{x}$. Since $f_{\boldsymbol{\phi}}(\boldsymbol{\theta}; \boldsymbol{x}) = \boldsymbol{z}$ by design, the change of variable rule of probability yields:

$$p_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \boldsymbol{x}) = p(\boldsymbol{z} = f_{\boldsymbol{\phi}}(\boldsymbol{\theta}; \boldsymbol{x})) \left| \det \left(\frac{\partial f_{\boldsymbol{\phi}}(\boldsymbol{\theta}; \boldsymbol{x})}{\partial \boldsymbol{\theta}} \right) \right| \tag{8}$$

Thus, we can re-write our objective as:

$$\widehat{\boldsymbol{\phi}} = \underset{\boldsymbol{\phi}}{\operatorname{argmax}} \int \int p(\boldsymbol{x}, \boldsymbol{\theta}) \log p_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \boldsymbol{x}) d\boldsymbol{x} d\boldsymbol{\theta} \tag{9}$$

$$= \underset{\boldsymbol{\phi}}{\operatorname{argmax}} \int \int p(\boldsymbol{x}, \boldsymbol{\theta}) \left(\log p(f_{\boldsymbol{\phi}}(\boldsymbol{\theta}; \boldsymbol{x})) + \log \left| \det \boldsymbol{J}_{f_{\boldsymbol{\phi}}} \right| \right) d\boldsymbol{x} d\boldsymbol{\theta} \tag{10}$$

where we have abbreviated $\partial f_{\boldsymbol{\phi}}(\boldsymbol{\theta}; \boldsymbol{x}) / \partial \boldsymbol{\theta}$ (the Jacobian of $f_{\boldsymbol{\phi}}$ evaluated at $\boldsymbol{\theta}$ and $\boldsymbol{x}$) as $\boldsymbol{J}_{f_{\boldsymbol{\phi}}}$. Due to the architecture of our cINN, the $\log \left| \det \boldsymbol{J}_{f_{\boldsymbol{\phi}}} \right|$ is easy to compute (see next section for details).

Utilizing simulations from the forward process (Eq.1), we can approximate the expectations by minimizing the Monte-Carlo estimate of the negative of Eq.10. Accordingly, for a batch of m simulated data sets and data-generating

parameters $\{(\boldsymbol{x}^{(i)}, \boldsymbol{\theta}^{(i)})\}_{i=1}^{M}$ we have:

$$\widehat{\boldsymbol{\phi}} = \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \frac{1}{M} \sum_{i=1}^{M} - \log p_{\boldsymbol{\phi}}(\boldsymbol{\theta}^{(i)} \mid \boldsymbol{x}^{(i)}) \tag{11}$$

$$= \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \frac{1}{M} \sum_{i=1}^{M} \left(- \log p(f_{\boldsymbol{\phi}}(\boldsymbol{\theta}^{(i)}; \boldsymbol{x}^{(i)})) - \log \left| \det \boldsymbol{J}_{f_{\boldsymbol{\phi}}}^{(i)} \right| \right) \tag{12}$$

$$= \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \frac{1}{M} \sum_{i=1}^{M} \left(\frac{\left\| f_{\boldsymbol{\phi}}(\boldsymbol{\theta}^{(i)}; \boldsymbol{x}^{(i)}) \right\|_2^2}{2} - \log \left| \det \boldsymbol{J}_{f_{\boldsymbol{\phi}}}^{(i)} \right| \right) \tag{13}$$

We treat Eq.13 as a loss function $\mathcal{L}(\boldsymbol{\phi})$ which can be minimized with any stochastic gradient descent method. The first term follows from Eq.12 due to the fact that we have prescribed a unit Gaussian distribution to $\boldsymbol{z}$ and represents the negative log of $\mathcal{N}_D(\boldsymbol{z} \mid \mathbf{0}, \mathbb{I}) \propto \exp(\left\| -\frac{1}{2} \boldsymbol{z} \right\|_2^2)$. The second term controls the rate of volume change induced by the learned non-linear transformation from $\boldsymbol{\theta}$ to $\boldsymbol{z}$ achieved by $f_{\boldsymbol{\phi}}$. Thus, minimizing Eq.13 ensures that $\boldsymbol{z}$ follows the prescribed unit Gaussian.

Our cINN is perfectly converged when $\mathbb{KL}(p(\boldsymbol{\theta} \mid \boldsymbol{x}) \| p_{\boldsymbol{\phi}}(\boldsymbol{\theta} \mid \boldsymbol{x})) = 0$ for all possible $\boldsymbol{x}$. This, in turn, implies equality in distributions $p(\boldsymbol{\theta} \mid \boldsymbol{x}) = p_{\boldsymbol{\phi}}(\boldsymbol{\theta} \mid \boldsymbol{x})$. Following a similar line of reasoning as in [2], we can prove that samples from a perfectly converged cINN follow the true posterior $p(\boldsymbol{\theta} \mid \boldsymbol{x})$.

Proposition 1. *Assume that we have a perfectly converged cINN $f_{\boldsymbol{\phi}}$ and an arbitrary but fixed observation $\boldsymbol{x}^*$. If $f_{\boldsymbol{\phi}}(\cdot; \boldsymbol{x}^*)$ transforms a probability density $p(\boldsymbol{\theta} \mid \boldsymbol{x} = \boldsymbol{x}^*)$ into $p(\boldsymbol{z})$, then the inverse $f_{\boldsymbol{\phi}}^{-1}(\cdot; \boldsymbol{x}^*)$ transforms $p(\boldsymbol{z})$ back into $p(\boldsymbol{\theta} \mid \boldsymbol{x} = \boldsymbol{x}^*)$. Therefore, repeatedly sampling $\boldsymbol{z} \sim p(\boldsymbol{z})$ and applying $f_{\boldsymbol{\phi}}^{-1}(\boldsymbol{z}; \boldsymbol{x}^*)$ to each $\boldsymbol{z}$ yields samples from $p(\boldsymbol{\theta} \mid \boldsymbol{x} = \boldsymbol{x}^*)$.*

Proof. Denote the conditional probability density of $\boldsymbol{\theta}$ obtained by inverse sampling as $p_{\boldsymbol{\phi}}(\boldsymbol{\theta} \mid \boldsymbol{x}^*)$. We need to show that $p_{\boldsymbol{\phi}}(\boldsymbol{\theta} \mid \boldsymbol{x}^*) = p(\boldsymbol{\theta} \mid \boldsymbol{x}^*)$. Applying the change of variable formula to the inverse transformation, we obtain:

$$p_{\boldsymbol{\phi}}(\boldsymbol{\theta} \mid \boldsymbol{x}^*) = p(\boldsymbol{z}) \left| \det \boldsymbol{J}_{f_{\boldsymbol{\phi}}} \right| \text{ with } \boldsymbol{z} = f_{\boldsymbol{\phi}}(\boldsymbol{\theta}; \boldsymbol{x}^*) \tag{14}$$

Correspondingly, for the forward transformation we have:

$$p(\boldsymbol{z}) = p(\boldsymbol{\theta} \mid \boldsymbol{x}^*) \left| \det \boldsymbol{J}_{f_{\boldsymbol{\phi}}^{-1}} \right| \text{ with } \boldsymbol{\theta} = f_{\boldsymbol{\phi}}^{-1}(\boldsymbol{z}; \boldsymbol{x}^*) \tag{15}$$

Substituting for $p(\boldsymbol{z})$ into Eq.14 leads to:

$$p_{\boldsymbol{\phi}}(\boldsymbol{\theta} \mid \boldsymbol{x}^*) = p(\boldsymbol{\theta} \mid \boldsymbol{x}^*) \left| \det \boldsymbol{J}_{f_{\boldsymbol{\phi}}^{-1}} \right| \left| \det \boldsymbol{J}_{f_{\boldsymbol{\phi}}} \right| \tag{16}$$

$$= p(\boldsymbol{\theta} \mid \boldsymbol{x}^*) \left| \det \left(\boldsymbol{J}_{f_{\boldsymbol{\phi}}}^{-1} \boldsymbol{J}_{f_{\boldsymbol{\phi}}} \right) \right| \tag{17}$$

$$= p(\boldsymbol{\theta} \mid \boldsymbol{x}^*) \tag{18}$$

which is true for all possible $\boldsymbol{x}^*$ due to the assumption of perfect convergence, that is, $\mathbb{KL}(p(\boldsymbol{\theta} \mid \boldsymbol{x}) \| p_{\boldsymbol{\phi}}(\boldsymbol{\theta} \mid \boldsymbol{x})) = 0$. This completes the proof. □

We now generalize our formulation to data sets with arbitrary numbers of observations. If we let the number of observations N vary and train a summary network $\tilde{\boldsymbol{x}} = f_{\boldsymbol{\psi}}(\boldsymbol{x}_{1:N})$ together with the cINN, our main objective changes to:

$$\widehat{\boldsymbol{\phi}}, \widehat{\boldsymbol{\psi}} = \underset{\boldsymbol{\phi}, \boldsymbol{\psi}}{\operatorname{argmax}} \, \mathbb{E}_{\boldsymbol{x}_{1:N} \sim p(\boldsymbol{x}, N)} \left[\mathbb{E}_{\boldsymbol{\theta} \sim p(\boldsymbol{\theta} \mid \boldsymbol{x}_{1:N})} \left[\log p_{\boldsymbol{\phi}}(\boldsymbol{\theta} \mid f_{\boldsymbol{\psi}}(\boldsymbol{x}_{1:N})) \right] \right] \tag{19}$$

and its Monte Carlo estimate to:

$$\widehat{\boldsymbol{\phi}}, \widehat{\boldsymbol{\psi}} = \underset{\boldsymbol{\phi}, \boldsymbol{\psi}}{\operatorname{argmin}} \frac{1}{M} \sum_{i=1}^{M} \left(\frac{\left\| f_{\boldsymbol{\phi}}(\boldsymbol{\theta}^{(i)}; f_{\boldsymbol{\psi}}(\boldsymbol{x}_{1:N}^{(i)})) \right\|_2^2}{2} - \log \left| \det \left(\boldsymbol{J}_{f_{\boldsymbol{\phi}}}^{(i)} \right) \right| \right) \tag{20}$$

In order to make estimation of $p(\boldsymbol{\theta} \,|\, \boldsymbol{x}_{1:N})$ tractable, we assume that there exists a vector $\boldsymbol{\eta}$ of sufficient statistics that captures all information about $\boldsymbol{\theta}$ contained in $\boldsymbol{x}_{1:N}$ in a fixed-size representation. For $f_{\boldsymbol{\psi}}(\boldsymbol{x}_{1:N})$ to be a useful estimator for $\boldsymbol{\eta}$, both should convey the same information about $\boldsymbol{\theta}$, as measured by the mutual information:

$$MI(\boldsymbol{\theta}, f_{\boldsymbol{\psi}}(\boldsymbol{x}_{1:N})) \approx MI(\boldsymbol{\theta}, \boldsymbol{\eta}) \tag{21}$$

where the mutual information between $\boldsymbol{\theta}$ and $f_{\boldsymbol{\psi}}(\boldsymbol{x}_{1:N})$ (analogously, between $\boldsymbol{\theta}$ and $\boldsymbol{\eta}$) is defined as:

$$MI(\boldsymbol{\theta}, f_{\boldsymbol{\psi}}(\boldsymbol{x}_{1:N})) = \mathbb{KL}(p(\boldsymbol{\theta}, f_{\boldsymbol{\psi}}(\boldsymbol{x}_{1:N})) \,||\, p(\boldsymbol{\theta})p(f_{\boldsymbol{\psi}}(\boldsymbol{x}_{1:N}))) \tag{22}$$

Since we do not know $\boldsymbol{\eta}$, we can enforce this requirement only indirectly by minimizing the Monte Carlo estimate of Eq.19. The following proposition states that, under perfect convergence, samples from a cINN still follow the true posterior given the outputs of a summary networks.

Proposition 2. *Assume that we have a perfectly converged cINN $f_{\boldsymbol{\phi}}$ and a perfectly converged summary network $f_{\boldsymbol{\psi}}$. Assume also, that there exists a vector $\boldsymbol{\eta}$ of sufficient summary statistics for $\boldsymbol{x}_{1:N}$. Then, repeatedly sampling $\boldsymbol{z} \sim p(\boldsymbol{z})$ and applying $f_{\boldsymbol{\phi}}^{-1}(\boldsymbol{z}; f_{\boldsymbol{\psi}}(\boldsymbol{x}_{1:N}))$ to each $\boldsymbol{z}$ yields samples from $p(\boldsymbol{\theta} \,|\, \boldsymbol{x}_{1:N})$.*

Proof. Perfect convergence of the networks under Eq.18 implies $\mathbb{KL}(p(\boldsymbol{\theta} \,|\, \boldsymbol{x}_{1:N}) \,||\, p_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, f_{\boldsymbol{\psi}}(\boldsymbol{x}_{1:N}))) = 0$. This, in turn, implies that $MI(\boldsymbol{\theta}, f_{\boldsymbol{\psi}}(\boldsymbol{x}_{1:N})) = MI(\boldsymbol{\theta}, \boldsymbol{\eta})$, because a perfect match of the densities would be impossible if $f_{\boldsymbol{\psi}}(\boldsymbol{x}_{1:N})$ contained less information about $\boldsymbol{\theta}$ than $\boldsymbol{\eta}$. Therefore, the proof reduces to that of **Proposition 1**. Note, that whenever the KL divergence is driven to a minimum, $f_{\boldsymbol{\psi}}(\boldsymbol{x}_{1:N})$ is a *maximally informative statistic* [12]. $\square$

In summary, the approximate posteriors obtained by the BayesFlow method are correct if the summary and invertible networks are perfectly converged. In practice, however, perfect convergence is unrealistic, and there are three sources of error which can lead to incorrect posteriors. The first is the Monte Carlo error introduced by using simulations from $g(\boldsymbol{\theta}, \xi)$ to approximate the expectation in Eq.19. The second is due to a summary network which may not fully capture the relevant structure of the data or when sufficient summary statistics do not exist. The third is due to an invertible network which does not accurately transform the true posterior into the prescribed Gaussian latent space. Even though we can mitigate the Monte Carlo error by running the simulator $g(\boldsymbol{\theta}, \xi)$ for a longer time, the latter two can be harder to detect and alleviate in a principled way. Nevertheless, recent work on *probabilistic symmetry* [5] and *algorithmic alignment* [59] can provide some guidelines on how to choose the right summary network for a particular problem. Additionally, the number as well as the structure of ACBs in an invertible chain can be tuned to increase the expressiveness of the learned transformation from $\boldsymbol{\theta}$-space to $\boldsymbol{z}$-space. The benefits of neural network depth has been confirmed both in theory [33] and practice [19], so we expect better performance in complex settings with increasing number of ACBs.

2.3 Composing Invertible Networks

The basic building block of our cINN is the affine coupling block (ACB) [14]. Each ACB consists of four separate fully connected neural networks denoted as $s_1(\cdot), s_2(\cdot), t_1(\cdot), t_2(\cdot)$. An ACB performs an invertible non-linear transformation, which means that in addition to a parametric mapping $f_{\boldsymbol{\phi}} : \mathbb{R}^D \to \mathbb{R}^D$ it also learns the inverse mapping $f_{\boldsymbol{\phi}}^{-1} : \mathbb{R}^D \to \mathbb{R}^D$ *for free*. Denoting the input vector of $f_{\boldsymbol{\phi}}$ as $\boldsymbol{u}$ and the output vector as $\boldsymbol{v}$, it follows that $f_{\boldsymbol{\phi}}(\boldsymbol{u}) = \boldsymbol{v}$ and $f_{\boldsymbol{\phi}}^{-1}(\boldsymbol{v}) = \boldsymbol{u}$. Invertibility is achieved by splitting the input vector into two parts $\boldsymbol{u} = (\boldsymbol{u}_1, \boldsymbol{u}_2)$ with $\boldsymbol{u}_1 = u_{1:D/2}$ and $\boldsymbol{u}_2 = u_{D/2+1:D}$ and performing the following operations on the split input:

$$\boldsymbol{v}_1 = \boldsymbol{u}_1 \odot \exp(s_1(\boldsymbol{u}_2)) + t_1(\boldsymbol{u}_2) \tag{23}$$
$$\boldsymbol{v}_2 = \boldsymbol{u}_2 \odot \exp(s_2(\boldsymbol{v}_1)) + t_2(\boldsymbol{v}_1) \tag{24}$$

where $\odot$ denotes element-wise multiplication. The outputs $\boldsymbol{v} = (\boldsymbol{v}_1, \boldsymbol{v}_2)$ are then concatenated again and passed to the next ACB. The inverse operation is given by:

$$\boldsymbol{u}_2 = (\boldsymbol{v}_2 - t_2(\boldsymbol{v}_1)) \odot \exp(-s_2(\boldsymbol{v}_1)) \tag{25}$$
$$\boldsymbol{u}_1 = (\boldsymbol{v}_1 - t_1(\boldsymbol{u}_2)) \odot \exp(-s_1(\boldsymbol{u}_2)) \tag{26}$$

This formulation ensures that the Jacobian of the affine transformation is a strictly upper or a lower triangular matrix and therefore its determinant is very cheap to compute. Furthermore, the internal functions $s_1(\cdot), s_2(\cdot), t_1(\cdot), t_2(\cdot)$ can be represented by arbitrarily complex neural networks, which themselves need not be invertible, since they are only ever evaluated in the forward direction during both the forward and the inverse pass through the ACBs. In our applications, we parameterize the internal functions as fully connected neural networks with exponential linear units (ELU).

In order to ensure that the neural network architecture is expressive enough to represent complex distributions, we chain multiple ACBs, so that the output of each ACB becomes the input to the next one. In this way, the whole chain remains invertible from the first input to the last output and can be viewed as a single function parameterized by trainable parameters ϕ.

In our applications, the input to the first ACB is the parameter vector θ, and the output of the final ACB is a d-dimensional vector z representing the non-linear transformation of the parameters. As described in the previous section, we ensure that z follows a unit Gaussian distribution via optimization, that is, $p(z) = \mathcal{N}_D(z \,|\, \mathbf{0}, \mathbb{I})$. Fixed permutation matrices are used before each ACB to ensure that each axis of the transformed parameter space z encodes information from all components of θ.

In order to account for the observed data, we feed the learned summary vectors into each internal network of each ACB (explained shortly). Intuitively, in this way we realize the following process: the forward pass maps data-generating parameters θ to z-space using conditional information from the data $x_{1:N}$, while the inverse pass maps data points from z-space to the data-generating parameters of interest using the same conditional information.

2.4   Summary Network

Since the number of observations usually varies in practical scenarios (e.g., different number of measurements or time points) and since data sets might exhibit various redundancies, the cINN can profit from some form of dimensionality reduction. As previously mentioned, we want to avoid information loss through restrictive hand-crafted summary statistics and, instead, learn the most informative summary statistics directly from data. Therefore, instead of feeding the raw simulated or observed data to each ACB, we pass the data through an additional summary network to obtain a fixed-sized vector of learned summary statistics $\tilde{x} = f_{\psi}(x_{1:N})$.

The architecture of the summary network should be aligned with the structure of the observed data. An obvious choice for time series-data is an LSTM-network [17], since recurrent networks can naturally deal with long sequences of variable size. Another choice might be a 1D fully convolutional network [34], which has already been applied in the context of likelihood-free inference [46]. A different architecture is needed when dealing with *i.i.d.* samples of variable size. Such data are often referred to as *exchangeable*, or *permutation invariant*, since changing the order of individual elements does not change the associated likelihood or posterior. In other words, if $\mathbb{S}_N(\cdot)$ is an arbitrary permutation of N elements, the following should hold for the posterior:

$$p(\theta \,|\, x_{1:N}) = p(\theta \,|\, \mathbb{S}_N(x_{1:N})) \tag{27}$$

Following [5], we encode probabilistic permutation invariance by implementing a permutation invariant function through an equivariant non-linear transformation followed by a pooling operator (e.g., sum or mean) and another non-linear transformation:

$$\tilde{x} = f_{\psi_1}\left(\sum_{i=1}^{N} f_{\psi_2}(x_i)\right) \tag{28}$$

where f_{ψ_1} and f_{ψ_2} are two different fully connected neural networks. In practice, we stack multiple equivariant and invariant functions into an invariant network in order to achieve higher expressiveness [5]. Additionally, we use an attention mechanism which computes learnable weights to each individual sample:

$$\tilde{w}_i = f_{\psi_3}(x_i) \tag{29}$$

$$w_i = \exp(\tilde{w}_i) \oslash \sum_{j=1}^{N} \exp(\tilde{w}_j) \tag{30}$$

$$\tilde{x} = f_{\psi_1}\left(\sum_{i=1}^{N} w_i \odot f_{\psi_2}(x_i)\right) \tag{31}$$

where Eq.30 performs an element-wise *softmax* operation to ensure that the weight vectors sum to 1 across each dimension. Note, that the function remains permutation invariant. The computation of attention weights is optional and introduces some computational overhead, but we observe much faster convergence and better performance when the summary network implements it.

We optimize the parameters ψ of the summary network jointly with those of the cINN chain via backpropagation. Thus, the method remains completely end-to-end and is capable of generalizing to data of variable size (within a given domain, by applying the same network) and structure (between different domains, by changing the architecture of the summary network).

To incorporate the observed or simulated data $\boldsymbol{x}_{1:N}$, each of the internal networks of each ACB is augmented to take the learned summary vector $\tilde{\boldsymbol{x}}$ of the data as an additional input. The output of each ACB then becomes:

$$\boldsymbol{v}_1 = \boldsymbol{u}_1 \odot \exp(s_1(\boldsymbol{u}_2, \tilde{\boldsymbol{x}})) + t_1(\boldsymbol{u}_2, \tilde{\boldsymbol{x}}) \tag{32}$$
$$\boldsymbol{v}_2 = \boldsymbol{u}_2 \odot \exp(s_2(\boldsymbol{v}_1, \tilde{\boldsymbol{x}})) + t_2(\boldsymbol{v}_1, \tilde{\boldsymbol{x}}) \tag{33}$$

Thus, a complete pass through the entire conditional invertible chain can be expressed as $f_\phi(\boldsymbol{\theta}; \tilde{\boldsymbol{x}}) = \boldsymbol{z}$ together with the inverse operation $f_\phi^{-1}(\boldsymbol{z}; \tilde{\boldsymbol{x}}) = \boldsymbol{\theta}$. The inverse transformation during inference is depicted in Figure 2.

2.5 Putting It All Together

Algorithm 1 describes the essential steps of the BayesFlow method using an arbitrary summary network and employing an online learning approach.

Algorithm 1 Bayesian inference with the BayesFlow method

1: *Training (via online learning):*
2: **repeat**
3:     Sample number of observations $N \sim \mathcal{U}(N_{min}, N_{max})$
4:     Sample a batch of parameters $\{\boldsymbol{\theta}^{(i)}\}_{i=1}^{M}$ from prior $p(\boldsymbol{\theta})$
5:     Simulate M data sets of size N via Eq.1 to get $\{\boldsymbol{x}_{1:N}^{(i)}\}_{i=1}^{M}$
6:     Pass $\{\boldsymbol{x}_{1:N}^{(i)}\}_{i=1}^{M}$ through summary network to obtain $\{\tilde{\boldsymbol{x}}^{(i)} = f_\psi(\boldsymbol{x}_{1:N}^{(i)})\}_{i=1}^{M}$
7:     Pass $\{(\boldsymbol{\theta}^{(i)}, \tilde{\boldsymbol{x}}^{(i)})\}_{i=1}^{M}$ through inference network to obtain $\{\boldsymbol{z}^{(i)} = f_\phi(\boldsymbol{\theta}^{(i)}; \tilde{\boldsymbol{x}}^{(i)})\}_{i=1}^{M}$
8:     Compute loss according to Eq.20
9:     Update neural network parameters ϕ, ψ via backpropagation
10: **until** convergence to $\widehat{\phi}, \widehat{\psi}$
11:
12: *Inference (given observed or test data $\boldsymbol{x}^o$):*
13: Compute learned summary of the data $\tilde{\boldsymbol{x}}^o = f_{\widehat{\psi}}(\boldsymbol{x}_{1:N}^o)$
14: **for** $l = 1, ..., L$ **do**
15:     Sample $\boldsymbol{z}^{(l)} \sim \mathcal{N}_D(\boldsymbol{0}, \mathbb{I})$
16:     Compute inverse $\boldsymbol{\theta}^{(l)} = f_{\widehat{\phi}}^{-1}(\boldsymbol{z}^{(l)}; \tilde{\boldsymbol{x}}^o)$
17: **end for**
18: Return $\{\boldsymbol{\theta}^{(l)}\}_{l=1}^{L}$ as a sample from $p(\boldsymbol{\theta} \mid \boldsymbol{x}^o)$

The backpropagation algorithm works by computing the gradients of the loss function with respect to the parameters of the neural networks and then adjusting the parameters, so as to drive the loss function to a minimum. We experienced no instability or convergence issues during training with the loss function given by Eq.19. Note, that steps 2-10 and 13-17 of **Algorithm** 1 can be executed in parallel with GPU support in order to dramatically accelerate convergence and inference. Moreover, steps 13-17 can be applied in parallel to an arbitrary number of observed data sets after convergence of the networks (see Figure 2 for a graphical illustration).

In what follows, we apply BayesFlow to two toy models with a unimodal and multimodal posteriors, respectively, and then use it to perform Bayesian inference on challenging models from population dynamics, cognitive science, epidemiology, and ecology.[1] We deem these models suitable for an initial validation, since they differ widely in the generative mechanisms they implement and the observed data they model. Therefore, good performance on these disparate examples underlines the broad empirical utility of the BayesFlow method. Details for models' setup can be found in **Appendix B**.

3 Experiments

3.1 Training the Networks

We train all invertible and summary networks described in this paper jointly via backpropagation. For all following experiments, we use the Adam optimizer with a starter learning rate of 10^{-3} and an exponential decay rate of .95.

---

[1]Code and simulation scripts for all current applications are available at `https://github.com/stefanradev93/cINN`.
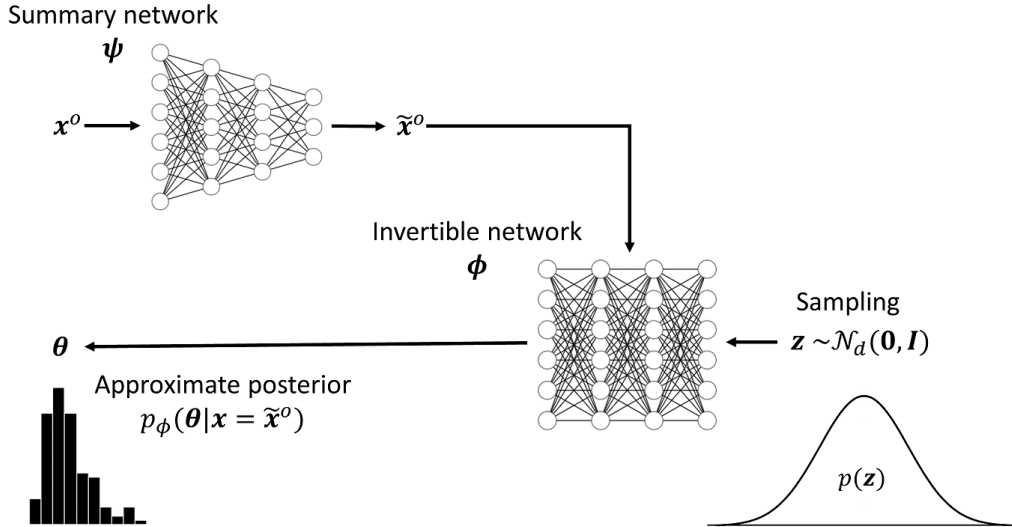
Figure 2: Inference with pre-trained summary and inference networks. The posterior is approximated given real observed data via independent samples from a learned pushforward distribution. Thus, knowledge about the mapping between data and parameters (the inverse model) is compactly encoded within the weights of the two networks.

We perform 50 000 to 100 000 iterations (i.e., mini-batch update steps) for each experiment, and report the results obtained by the converged networks. Note, that we did not perform an extensive search for optimal values of network hyperparameters, but use a default BayesFlow with 5 to 10 ACBs and a summary vector of size 128 for all examples in this paper (see **Appendix B** for more details on summary network architectures). All networks were implemented in Python using the *TensorFlow* library [1] and trained on a single-GPU machine equipped with NVIDIA® GTX1060 graphics card. Regarding the data generation step, we take an approach which incorporates ideas from *online learning* [41] where data are stimulated by Eq.1 on demand.

Correspondingly, a data set $x_{1:N}$, or a batch of M data sets $\{x_{1:N}^{(i)}\}_{i=1}^{M}$, is generated on the fly and then passed through the neural network. This training approach has the advantage that the network never *experiences* the same input data twice. Moreover, training can continue as long as the network keeps improving (i.e., the loss keeps decreasing), since overfitting in the classical sense is nearly impossible. However, if simulations are computationally expensive and researchers need to experiment with different networks or training hyperparameters, it might be beneficial to store and re-use simulations, since simulation and training in online learning are tightly intertwined.

Once the networks have converged, we store the trained networks and re-use them to perform amortized inference on a separate validation set of data sets. The pre-trained networks can also be shared among a research community so that multiple researchers/labs can benefit from the amortization of inference.

3.2   Performance Validation

To evaluate the performance of BayesFlow in the following application examples, we consider a number of different metrics:

- Normalized root mean squared error (NRMSE) - to asses accuracy of point-estimates in recovering ground-truth parameter values;

- Coefficient of determination (R^2) - to asses the proportion of variance in ground-truth parameters that is captured by the point estimates;

- Re-simulation error (Err_{sim}) - to asses the predictive mismatch between the true data distribution and the data distribution generated with the estimated parameters (i.e., posterior predictive check);

(a) Analytic vs. estimated posteriors on the 5-D MVN model.

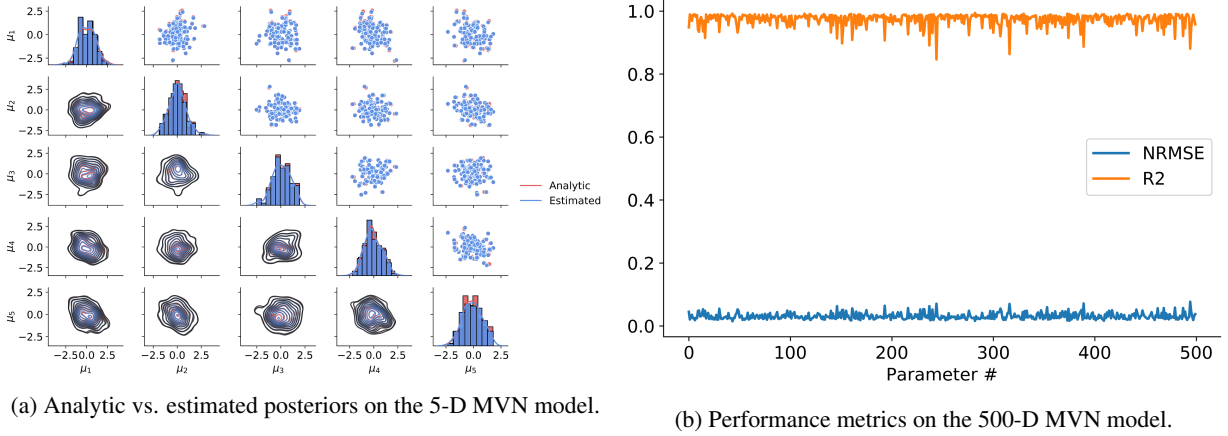(b) Performance metrics on the 500-D MVN model.

Figure 3: Results on the MVN toy examples. **(a)** Pair-plots of the analytic vs. estimated posterior means on the 5-D Gaussian example. The main diagonal represents histograms of the analytic and approximate means. The lower and upper portions of the figure represent bivariate KDE estimates and scatter plots, respectively. We observe a near-perfect overlap between analytic and approximate posteriors. Further, the model correctly captures the true posterior covariance; **(b)** NRMSE and R^2 performance metrics on the 500-D MVN model for all 500 Gaussian mean parameters.

- Calibration error (Err_{cal}, [2]) - to asses the coverage of the approximate posteriors (i.e., whether credibility intervals are indeed credible);
- Simulation-based calibration (SBC, [52]) - to visually detect systematic biases in the approximate posteriors;

Details for computing all metrics can be found in **Appendix A**.

3.3 Proof of Concept: Multivariate Normal Distribution

As a proof-of-concept, we apply the BayesFlow method to recover the posterior mean vector of a toy multivariate normal (MVN) example. For a D-dimensional MVN vector, the forward model is given by:

$$\boldsymbol{\mu}^{(i)} \sim \mathcal{N}_D(\boldsymbol{\mu} \,|\, \mathbf{0}, \mathbb{I}) \tag{34}$$

$$\boldsymbol{x}^{(i)} \sim \mathcal{N}_D(\boldsymbol{x} \,|\, \boldsymbol{\mu}^{(i)}, \boldsymbol{\Sigma}) \tag{35}$$

If the covariance matrix $\boldsymbol{\Sigma}$ is known, the posterior of the mean vector $\boldsymbol{\mu}$ has a closed-form which is also a MVN $p(\boldsymbol{\mu} \,|\, \boldsymbol{x}, \boldsymbol{\Sigma}) = \mathcal{N}_d(\boldsymbol{\mu} \,|\, \boldsymbol{m}, \boldsymbol{\Lambda})$ with posterior precision matrix given by $\boldsymbol{\Lambda}^{-1} = \mathbb{I} + \boldsymbol{\Sigma}^{-1}$ and posterior mean given by $\boldsymbol{m} = \boldsymbol{\Lambda}\boldsymbol{\Sigma}^{-1}\boldsymbol{x}$ [6]. We can thus generate multiple batches of the form $\{(\boldsymbol{x}^{(i)}, \boldsymbol{\mu}^{(i)})\}_{i=1}^M$ and pass them directly through an invertible network. Since the ground-truth posterior is Gaussian, we can compute the KL divergence as a measure of mismatch between the true and approximate posteriors in closed form:

$$\mathbb{KL}(p(\boldsymbol{\mu} \,|\, \boldsymbol{x}, \boldsymbol{\Lambda}) \,||\, p_\phi(\boldsymbol{\mu} \,|\, \boldsymbol{x})) = \mathbb{KL}(\mathcal{N}_D(\boldsymbol{\mu} \,|\, \boldsymbol{m}, \boldsymbol{\Lambda}) \,||\, \mathcal{N}_D(\boldsymbol{\mu} \,|\, \boldsymbol{m}_\phi, \boldsymbol{\Lambda}_\phi)) \tag{36}$$

$$= \frac{1}{2}\left[\log \frac{\det \boldsymbol{\Lambda}_\phi}{\det \boldsymbol{\Lambda}} + \mathrm{Tr}(\boldsymbol{\Lambda}_\phi^{-1}\boldsymbol{\Lambda}) - D + (\boldsymbol{m} - \boldsymbol{m}_\phi)^T \boldsymbol{\Lambda}_\phi^{-1}(\boldsymbol{m} - \boldsymbol{m}_\phi)\right] \tag{37}$$

where $\boldsymbol{\Lambda}_\phi$ and $\boldsymbol{m}_\phi$ denote the covariance matrix and mean vector computed from samples obtained by BayesFlow. We run three experiments with $D \in \{5, 50, 500\}$ where the size of the ACB blocks was doubled for each successive D. To asses results, we compute the R^2 and NRMSE between approximate and true means as well as the KL divergence between approximate and true distributions on 100 test datasets. To compute the approximate covariance matrix, we draw 5000 samples from the approximate posteriors for $D = 5$ and $D = 50$ and 50000 samples for $D = 500$.

The KL divergence for the 5-D and 50-D MVNs reached 0 after 2-3 epochs of 1000 iterations indicating perfect recovery of the true posteriors. The results on the 5-D MVN are illustrated in Figure 3a. We observe that BayesFlow is able to capture the true covariance of the posterior. The same applies for the 50-D model. The KL divergence for the 500-D MVN model reached 0.37 after 50 epochs, indicating very good approximation with respect to the high dimensionality of the problem. Figure 3b depicts the recovery of the true posterior means. We observe low NRMSE scores and high R^2 across all 500 mean parameters (*x*-axes).
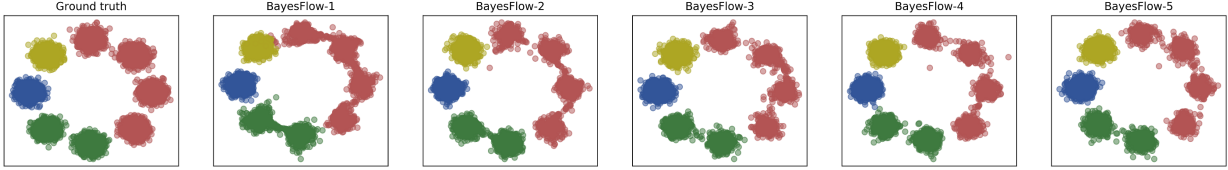
Figure 4: Results on the GMM toy example with colors indicating cluster assignment. Approximation of the multimodal posterior become closer to the ground truth distribution with increasing depth (number of ACBs) of the conditional invertible network.

3.4 Multimodal Posterior: Gaussian Mixture Model

In order to test whether the BayesFlow method can recover multimodal posteriors, we apply it to a generative Gaussian mixture model (GMM). Multimodal posteriors arise in practice, for example, when forward models are defined as mixtures between different processes, or when models exhibit large multivariate trade-offs in their parameter space (e.g., there are multiple separate regions of posterior density with plausible parameter values). Therefore, it is important to show that our method is able to capture such behavior and does not suffer from mode collapse.

Following [2], we construct a scenario in which the observed data x is a one-hot encoded vector representing one of the labels *red*, *green*, *blue*, or *yellow* (hard assignment of labels). The parameters $\theta = (\theta_1, \theta_2)$ are the 2D coordinates [2] of points drawn from a mixture of eight Gaussian clusters with centers distributed around the origin in a clockwise manner and unit variance (Figure 4). The first four clusters are assigned the label *red*, the next two the label *green*, and the remaining two the labels *blue* and *yellow*. The posterior $p(\theta \mid x)$ is composed of the clusters indexed by the corresponding label. We perform the experiment multiple times by increasing the depth of the BayesFlow starting from 1 ACB block up to 5 ACB blocks. In this way, we can investigate the effects of cINN depth on the quality of the approximate multimodal posteriors. We train each BayesFlow for 50 epochs and draw 8000 samples from the approximate posteriors obtained by the trained models.

Results for all BayesFlows are depicted in Figure 4. We observe that approximations profit from having a deeper cINN chain, with cluster separation becoming clearer when using more ACBs. This confirms that our method is capable of recovering multimodal posteriors.

3.5 Stochastic Time-Series Model: The Ricker Model

In the following, we estimate the parameters of a well-known discrete stochastic population dynamics model [58]. With this example, we are pursuing several goals: First, we want to demonstrate that the BayesFlow method is able to accurately recover the parameters of an actual model with intractable likelihood by learning summary statistics from raw data. Second, we show that BayesFlow can deal adequately with parameters that are completely unrelated to the data by reducing estimates to the corresponding parameters' prior. Third, we compare the global performance of the BayesFlow method to that of related methods capable of amortized likelihood-free inference. Finally, we demonstrate the desired posterior contraction and improvement in estimation with increasing number of observations.

Discrete population dynamics models describe how the number of individuals in a population changes over discrete units of time [58]. In particular, the Ricker model describes the number of individuals x_t in generation t as a function of the expected number of individuals in the previous generation by the following non-linear equations:

$$x_t \sim Pois(\rho N_t) \tag{38}$$

$$\xi_t \sim \mathcal{N}(0, \sigma^2) \tag{39}$$

$$N_{t+1} = r N_t e^{-N_t + \xi_t} \tag{40}$$

for $t = 1, ..., T$ where N_t is the expected number of individuals at time t, r is the growth rate, ρ is a scaling parameter and ξ_t is random Gaussian noise. The likelihood function for the Ricker model is not available in closed form, and the model is known to exhibit chaotic behavior [38]. Thus, it is a suitable candidate for likelihood-free inference. The parameter estimation task consists of recovering $\theta = (\rho, r, \sigma)$ from the observed one-dimensional time-series data $x_{1:T}$ where each $x_t \in \mathbb{N}$.

What if the data does not contain any information about a particular parameter? In this case, any good estimation method should detect this, and return the prior of the particular parameter. To test this, we append a random uniform

---

[2]Note that this is not the typical GMM setup, as we construct the example such that the mixture assignments (labels) are observed and the data coordinates are the latent parameters.

(a) Full posteriors from all methods for an example Ricker dataset.

(b) Performance over all Ts

(c) Parameter recovery ($T = 500$)

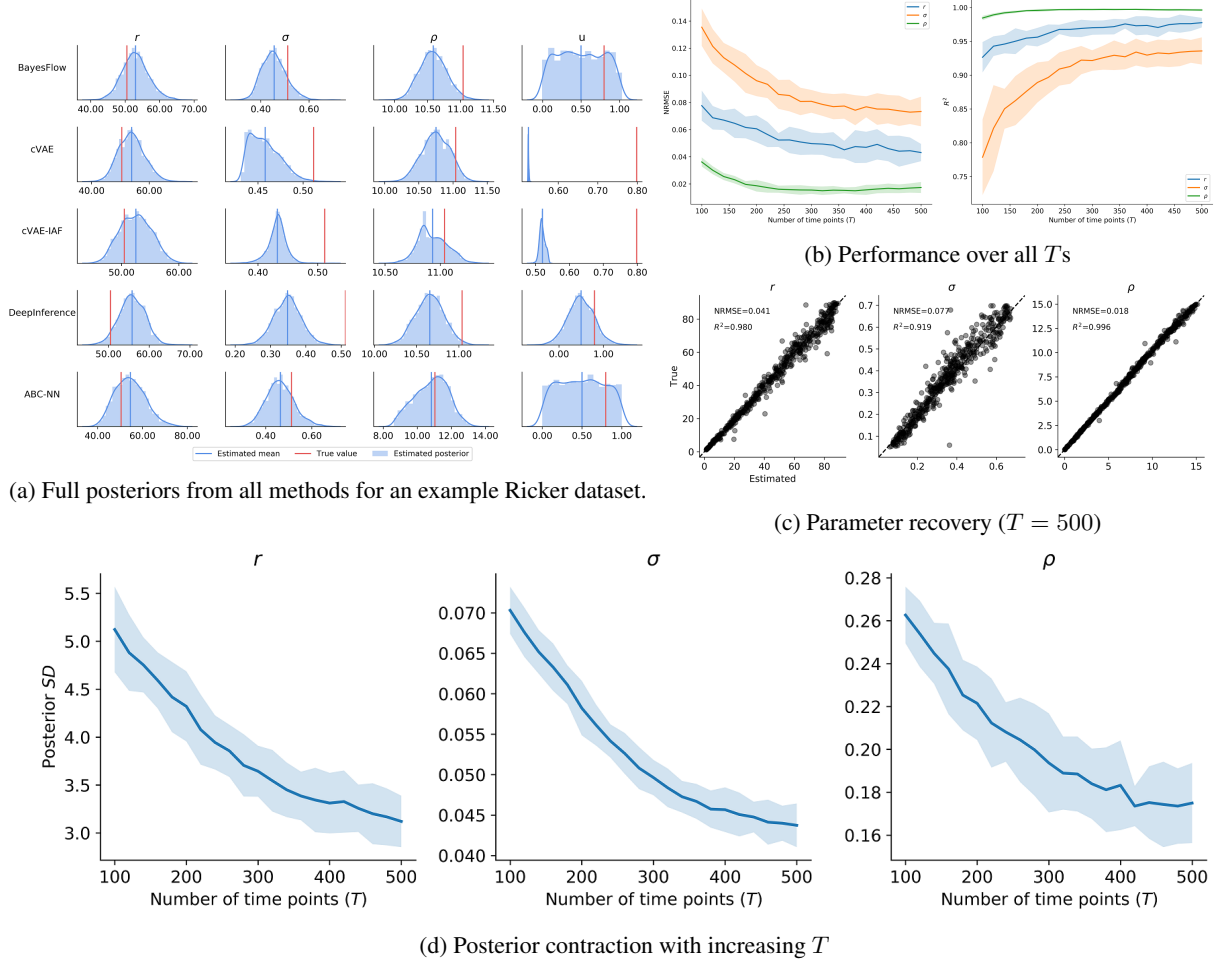(d) Posterior contraction with increasing T

Figure 5: Results on the Ricker model. **(a)** Approximate posteriors obtained by all implemented methods on a single Ricker dataset. Note that only BayesFlow and ABC-NN are able to approximate the uniform posterior of u; **(b)** NRMSE and R^2 performance metrics over all Ts obtained by the BayesFlow method. We observe that parameter estimation remains good over all Ts, and becomes progressively better as more data is available (shaded regions indicate bootstrap 95% CIs); **(c)** Parameter recovery with BayesFlow for the maximum number of generations used during training ($T = 500$); **(d)** Posterior contraction in terms of posterior standard deviation for each parameter across increasing number of available generations (shaded regions indicate bootstrap 95% CIs).

variable $u \sim \mathcal{U}(0, 1)$ to the parameter vector $\boldsymbol{\theta}$ and train BayesFlow with this additional dummy parameter. We expect that the networks ignore this dummy parameter, that is, we assume that the estimated posterior of u resembles the uniform prior.

We compare the performance of BayesFlow to the following recent methods capable of amortized likelihood-free inference: conditional variational autoencoder (cVAE) [40], cVAE with autoregressive flow (cVAE-IAF) [29], *DeepInference* with heteroscedastic loss [46], approximate Bayesian computation with an LSTM neural network for learning informative summary statistics (ABC-NN) [26] and quantile random forest (ABC-RF) [48]. For training the models, we simulate time-series from the Ricker model with varying lengths. The number of time points T is drawn from a uniform distribution $T \sim \mathcal{U}(100, 500)$ at each training iteration. All neural network methods were trained for 100 epochs with 1000 iterations each on simulated data from the Ricker model. The ABC-RF method was fitted on a reference table with 200 000 datasets, since the method does not allow for online learning and increasing the reference table did not seem to improve performance. In order to avoid using hand-crafted summary statistics for the ABC-RF method, we input summary vectors obtained by applying the summary network trained jointly with the cINN. Thus, the ABC-RF method has the advantage of using maximally informative statistics as input. We validate the performance of all methods on an independent test set of 500 datasets generated with $T = 500$. We report performance metrics for each method and each parameter in Table 1.

Table 1: Performance results on the Ricker model across all estimation methods

		BayesFlow	cVAE	cVAE-IAF	DeepInference	ABC-NN	ABC-RF
Err_{cal}	r	0.017 ± 0.007	$\mathbf{0.014 \pm 0.007}$	0.058 ± 0.017	0.122 ± 0.016	0.164 ± 0.015	-
	σ	$\mathbf{0.013 \pm 0.007}$	0.419 ± 0.011	0.382 ± 0.013	0.184 ± 0.021	0.119 ± 0.014	-
	ρ	$\mathbf{0.084 \pm 0.018}$	0.121 ± 0.017	0.188 ± 0.018	0.111 ± 0.019	0.283 ± 0.012	-
$NRMSE$	r	$\mathbf{0.041 \pm 0.002}$	0.047 ± 0.004	0.047 ± 0.006	0.052 ± 0.003	0.053 ± 0.003	0.044 ± 0.004
	σ	$\mathbf{0.077 \pm 0.005}$	0.137 ± 0.004	0.124 ± 0.006	0.108 ± 0.004	$\mathbf{0.077 \pm 0.004}$	0.081 ± 0.005
	ρ	0.018 ± 0.001	$\mathbf{0.016 \pm 0.002}$	0.019 ± 0.002	0.019 ± 0.002	0.033 ± 0.002	0.021 ± 0.001
R^2	r	$\mathbf{0.980 \pm 0.003}$	0.973 ± 0.005	0.973 ± 0.007	0.968 ± 0.005	0.966 ± 0.004	0.977 ± 0.004
	σ	$\mathbf{0.919 \pm 0.011}$	0.745 ± 0.020	0.792 ± 0.020	0.841 ± 0.014	$\mathbf{0.919 \pm 0.010}$	0.912 ± 0.011
	ρ	0.996 ± 0.001	$\mathbf{0.997 \pm 0.001}$	0.996 ± 0.001	0.996 ± 0.001	0.986 ± 0.002	0.994 ± 0.001
Err_{sim}	-	$\mathbf{0.038 \pm 0.001}$	0.041 ± 0.001	0.042 ± 0.001	0.041 ± 0.001	0.048 ± 0.002	0.041 ± 0.002

Note: For each parameter, bootstrapped means (± 1 standard error) of different performance metrics are displayed for all tested methods. For each metric and each parameter, the best performance across methods is printed in bold font.

Parameters r and ρ seem to be well recoverable by all methods considered here. The σ parameter turns out to be harder to estimate, with BayesFlow and the ABC-NN method performing best. Further, BayesFlow performs very well across all parameters and metrics. Importantly, the calibration error Err_{cal} obtained by BayesFlow is always low, indicating that the shape of the approximate posterior closely matches that of the true posteriors. Variational methods (cVAE, cVAE-IAF) experience some problems recovering the posterior of σ. The ABC-NN and ABC-RF methods seem to recover point estimates with high accuracy but the approximate posteriors of the former exhibit relatively high calibration error. The ABC-RF method can only estimate posterior quantiles, so no comparable calibration metric could be computed.

Further results are depicted in Figure 5. Inspecting the full posteriors obtained by all methods on an example test dataset, we note that only BayesFlow and the ABC-NN methods are able to recover the uninformative posterior distribution of the dummy noise variable u (Figure 5a). Moreover, the importance of a Bayesian treatment of the Ricker model becomes clear when looking at the posteriors of σ. On most test datasets, the posterior density spreads over the entire prior range (high posterior variance) indicating large uncertainty in the obtained estimates. Some datasets even resulted in bimodal posteriors for σ, which highlight the importance of imposing no *ad hoc* restrictions on the posteriors. We also observe that parameter estimation with BayesFlow becomes increasingly accurate when more time points are available (Figure 5b). Parameter recovery is especially good with the maximum number of time points (see Figure 5c). Finally, (Figure 5d) reveals a notable posterior contraction across increasing number of time points available to the summary network.

3.6 A Model of Perceptual Decision Making: The Lévy-Flight Model

In the following, we estimate the parameters of a stochastic differential equation model of human decision making. We perform the first Bayesian treatment of the recently proposed Lévy-Flight Model (LFM), as its intractability has so far rendered traditional non-amortized Bayesian inference methods prohibitively slow [56].

With this example, we first want to show empirically that BayesFlow is able to deal with $i.i.d.$ data sets of variable size arising from N independent runs of a complex stochastic simulator. For this, we inspect global performance of BayesFlow over a wide range of data set sizes. Additionally, we want to show the advantage of amortized inference compared to case-based inference in terms of efficiency and recovery. For this, we apply BayesFlow along with four other recent methods for likelihood-free inference to a single data set and show that in some cases the speed advantage of amortized inference becomes noticeable even after as few as 5 data sets. Crucially, researchers often fit the same models to different datasets, so if a pre-trained model exists, it would present a huge advantage in terms of efficiency and productivity.

We focus on the family of evidence accumulator models (EAMs) which describe human decision making by a set of neurocognitively motivated parameters [47]. EAMs are most often applied to choice reaction times (RT) data to obtain an estimate of the underlying processes governing categorization and (perceptual) decision making. In its most general formulation, the forward model of EAMs takes the form of a stochastic ordinary differential equation (ODE) given by [54]:

$$dx = v dt + \xi \sqrt{dt} \tag{41}$$

where dx denotes a change in activation of an accumulator, v denotes the average speed of information accumulation (often termed the drift rate), and ξ represents a stochastic additive component, usually modeled as coming from a Gaussian distribution centered around 0: $\xi \sim \mathcal{N}(0, c^2)$.

EAMs are particularly amenable for likelihood-free inference, since the likelihood of most interesting members of this model family turn out to be intractable [39]. This intractability has precluded many interesting applications and empirically driven model refinements. Here, we apply BayesFlow to estimate the parameters of the recently proposed Lévy-Flight Model (LFM) [56]. The LFM assumes an α-stable noise distribution of the evidence accumulation process which allows to model discontinuities in the decision process. However, the inclusion of α-stable noise (instead of the typically assumed Gaussian noise) leads to a model with intractable likelihood:

$$dx = vdt + \xi dt^{1/\alpha} \tag{42}$$
$$\xi \sim AlphaStable(\alpha, 0, 1, 0) \tag{43}$$

where α controls the probability of outliers in the noise distribution. The LFM has three additional parameters: the threshold a determining the amount of evidence needed for the termination of a decision process; a relative starting point, zr, determining the amount of starting evidence available to the accumulator before the actual decision alternatives are presented; and an additive non-decision time t_0.

During training of the networks, we simulate response times data from two experimental conditions with two different drift rates, since such a design is often encountered in psychological research. The parameter estimation task is thus to recover the parameters $\boldsymbol{\theta} = (v_0, v_1, a, t_0, zr, \alpha)$ from two-dimensional *i.i.d.* RT data $\boldsymbol{x}_{1:N}$ where each $\boldsymbol{x}_i \in \mathbb{R}^2$ represents RTs obtained in the two conditions. The number of trials is drawn from a uniform distribution $N \sim \mathcal{U}(100, 1000)$ at each training iteration. Training the networks took a little less than a day with the online learning approach. Inference on 1000 datasets with 2000 posterior samples per parameter took approximately 7.39 seconds.
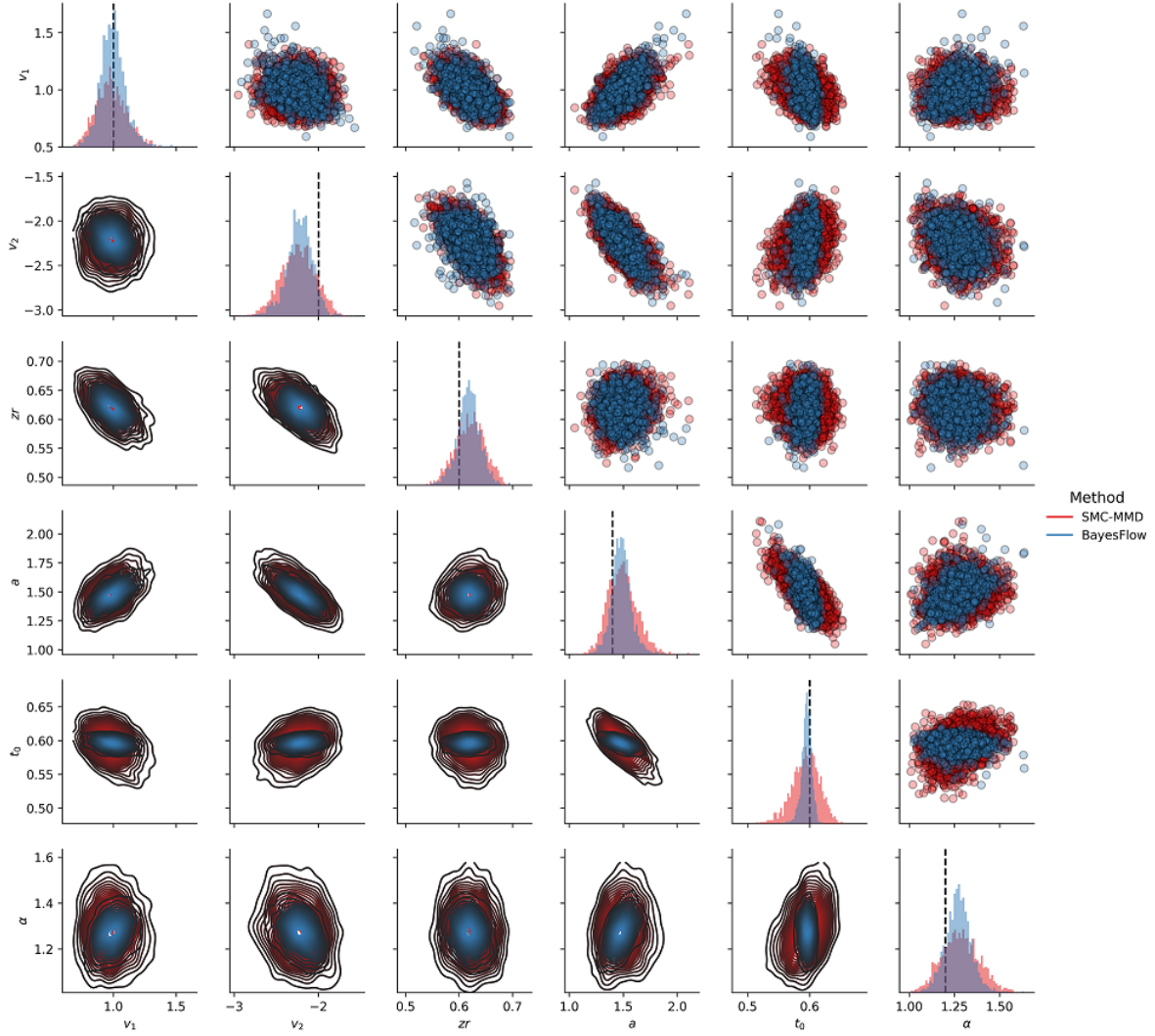
In order to investigate whether amortized inference is advantageous for this model, we additionally apply a version of the SMC-ABC algorithm available in the *pyABC* package [32] to a single data set with $N = 500$. Since no sufficient summary statistics are available for EAM data, we apply the maximum mean discrepancy (MMD) metric as a distance between the full raw empirical RT distributions, in order to prevent information loss [44]. Since the MMD is expensive to compute, we use a GPU implementation to ensure that computation of MMD is not a bottleneck for the comparison. In order to achieve good approximation with 2000 samples from the SMC-MMD approximate posterior, we run the algorithm for 20 populations with a final rejection threshold $\epsilon = 0.04$. We also draw 2000 samples from the approximate posterior obtained by applying our pre-trained BayesFlow networks to the same data set.

Along SMC-MMD, we apply three recent methods for neural density estimation, SNPE-A [42], SNPE-B [35], and SNPE-C ([20], also dubbed APT). Since these methods all depend on summary statistics of the data, we compute the first 6 moments of each empirical response time distribution as well as the fractions of correct/wrong responses. We train each method for a single round with 100 epochs and 5000 simulated datasets, in order to keep running time at a minimum. Also, we did not observe improvement in performance when training for more than one round. For each model, we sample 2000 samples from the approximate joint posterior to align the number of samples with those obtained via SMC-MMD.
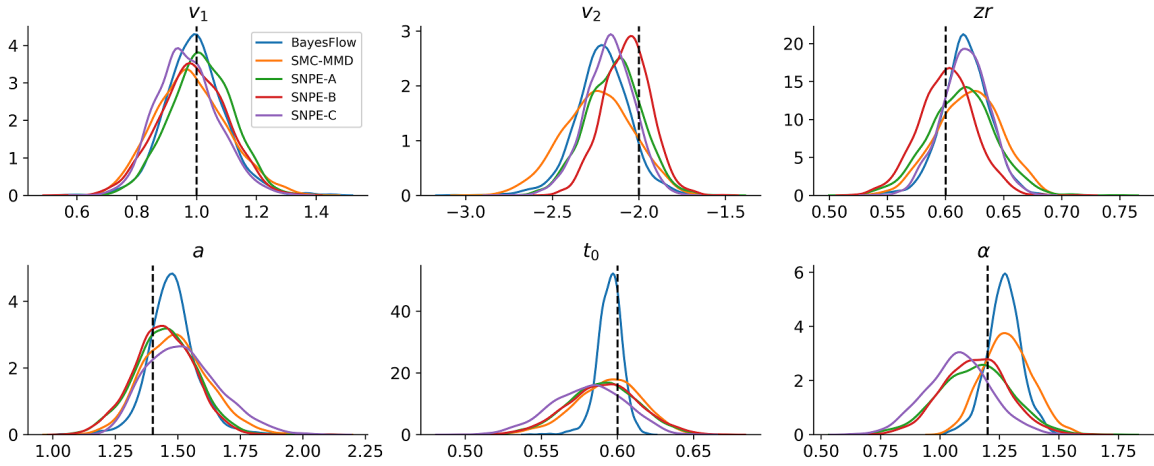
The comparison results are depicted in Figure 6. We first focus on the comparison with SMC-MMD on the single data set. Figure 6a depicts marginal and bivariate posteriors obtained by BayesFlow and SMC-MMD. The approximate posteriors of BayesFlow appear noticeably sharper. Observing the SCB plots (Figure 7b), we can conclude that the approximate posteriors of BayesFlow mirror the sharpness of the true posterior, since otherwise the SCB plots would show marked deviations from uniformity. Further, Figure 6b depicts the marginal posteriors obtained from the application of each method. Noticeably, performance and sharpness varies across the methods and parameters, with all methods yielding good point-estimate recovery via posterior means in terms of the NRMSE and R^2 metrics.

Importantly, Table 2 lists the advantage of amortized inference for the LFM model. For instance, compared to SMC-MMD, the extra effort of learning a global BayesFlow model upfront is worthwhile even after as few as 5 datasets, as inference with SMC-MMD would have taken more than a day to finish. On the other hand, the break-even for SNPE-C/APT occurs after 75 datasets, so in cases where only a few dozens of data sets are considered, case-based inference might be preferable. However, the difficulties in manually finding meaningful and efficiently computable summary statistics may eat up possible savings even in this situation. We acknowledge that our choices in this respect might be sub-optimal, so performance comparisons should be treated with some caution.

We note, that after a day of training, the pre-trained networks of BayesFlow take less than 5 seconds to perform inference on 500 datasets even with maximum number of trials $N = 1000$. Using the case-based SMC-MMD algorithm, 500 inference runs would have taken more than half a year to complete. We also note, that parallelizing separate inference threads across multiple cores or across nodes of a (GPU) computing cluster can dramatically increase the wall-clock

(a) Joint posteriors from BayesFlow and SMC-MMD

(b) Marginal posteriors from all methods

Figure 6: Comparison results on the LFM model. **(a)** Marginal and bivariate posteriors obtained by BayesFlow and SMC-MMD on the single validation dataset. We observe markedly better sharpness in the BayesFlow posteriors; **(b)** Marginal posteriors obtained from all methods under comparison.
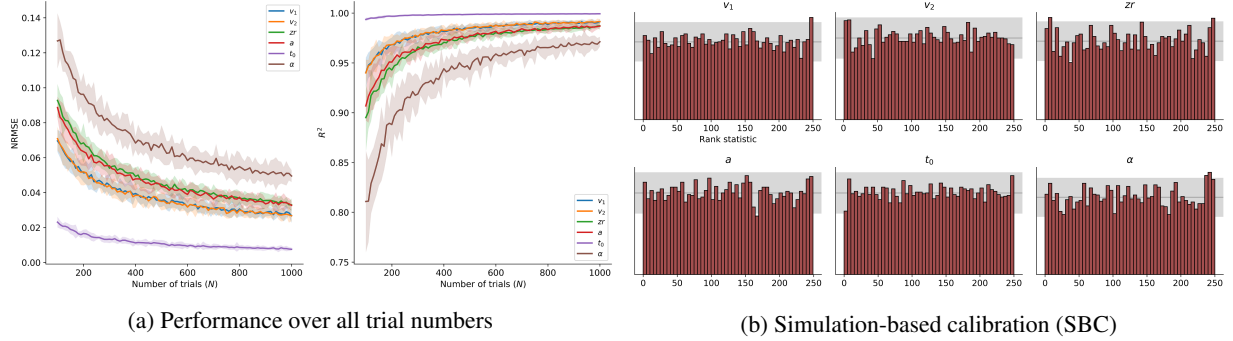
(a) Performance over all trial numbers

(b) Simulation-based calibration (SBC)

Figure 7: Results on the LFM model. **(a)** NRMSE and R^2 over all N trials used during training. Again, we observe that recovery remains very good overall, and becomes progressively better as more data becomes available; **(b)** Plots of the rank statistics suggesting no systematic deviations in the approximate posteriors (the shaded gray region indicates a 99% confidence bound);

Table 2: Speed of inference and break-even for amortized inference for the LFM model

	Upfront Training	Inference (1 data set)	Inference (500 data sets)	Break-even after
BayesFlow	23.2 h	60 ms	3.7 s	-
SMC-MMD	-	5.5 h	2700 h	5 data sets
SNPE-A	-	0.65 h	325 h	37 data sets
SNPE-B	-	0.65 h	325 h	37 data sets
SNPE-C	-	0.35 h	175 h	75 data sets

Note: Inference times for 500 datasets as well as the number of data sets for break-even with BayesFlow for the SMC-MMD, SNPE-A, SNPE-B, and SNPE-C methods are extrapolated from the wall-clock running time on a single data set, so these are approximate quantities.

speed of the case-based methods considered here. However, the same applies to BayesFlow training, since its most expensive part, the simulation from the forward model, would profit the most from parallel computing.

The global performance of BayesFlow over all validation data sets and all trial sizes N is depicted in (Figure 7). First, we observe excellent recovery of all LFM parameters with NRMSEs ranging between 0.008 and 0.048 and R^2 between 0.972 for the maximum number of trials. Importantly, estimation remains very good across all trial numbers, and improves as more trials become available (Figure 7a). The parameter α appears to be most challenging to estimate, requiring more data for good estimation, whereas the non-decision time parameter t_0 can be recovered almost perfectly for all trial sizes. Last, the SCB histograms indicate no systematic deviations across the marginal posteriors (Figure 7b).

3.7 Stochastic Differential Equations: The SIR Epidemiology Model

With this example, we want to further corroborate the excellent global performance and probabilistic calibration observed for the LFM model on a non-*i.i.d.* stochastic ODE model. Even though the generative mechanism of the LFM is formulated as a stochastic ODE, its output consists of *i.i.d.* data sets comprising multiple runs of the simulator. Here, we study a compartmental model from epidemiology, whose output comprises variable-sized multidimensional and inter-dependent time-series. It is therefore of interest to investigate whether our method is applicable to data which is the direct output of an ODE simulator.

Compartmental models in epidemiology describe the stochastic dynamics of infectious diseases as they spread over a population of individuals [49, 27, 24]. The parameters of compartmental models encode important characteristics of diseases, such as the rates of infection or recovery from the disease. The stochastic SIR model describes the transition dynamics of N individuals between three discrete states: susceptible (S), infected (I), and recovered (R). The transition

(a) Parameter recovery ($T = 500$)

(b) Performance over all Ts

(c) Simulation-based calibration (SBC)
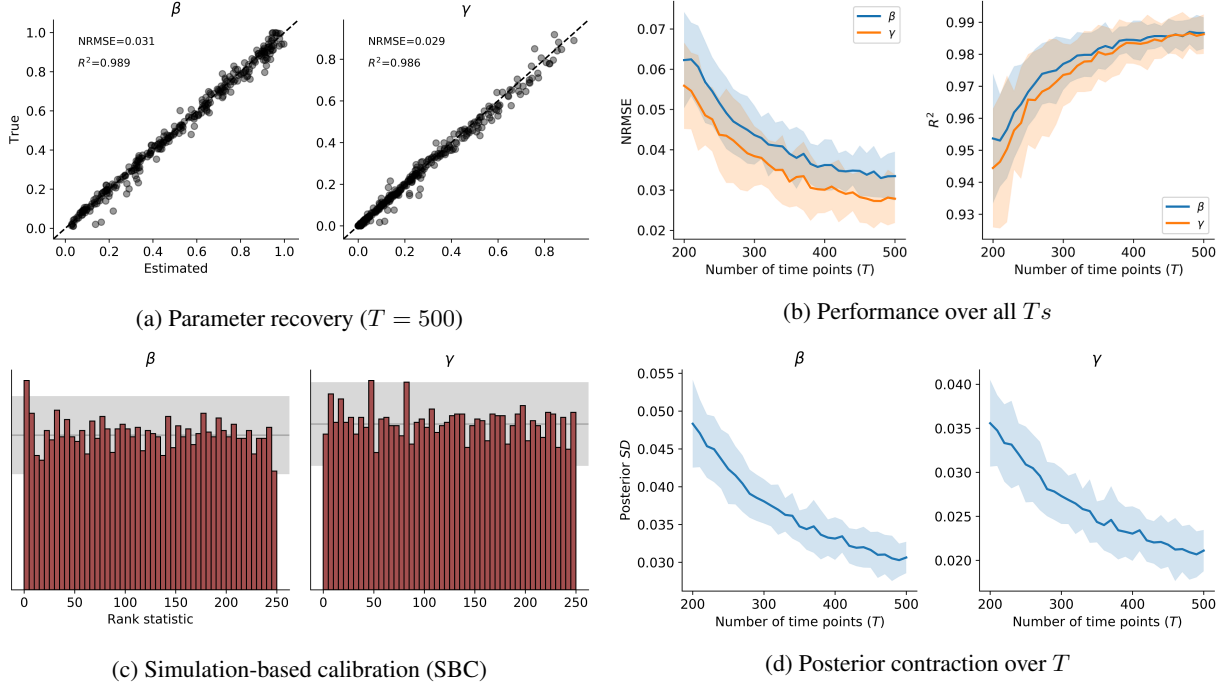
(d) Posterior contraction over T

Figure 8: Results obtained on the stochastic SIR model. **(a)** Parameter recovery depicting also NRMSE and R^2 metrics; **(b)** NRMSE and R^2 performance over all Ts seen by the networks during training; **(c)** Plots of the rank statistics suggesting a slight overestimation by posterior mean of β in the lower range (the shaded gray region indicates a 99% confidence bound); **(d)** Reduction in posterior variance (posterior contraction) over increasing T.

dynamics are given by the following equations:

$$\triangle S = -\triangle N_{SI} \tag{44}$$

$$\triangle I = \triangle N_{SI} - \triangle N_{IR} \tag{45}$$

$$\triangle R = \triangle N_{IR} \tag{46}$$

$$\triangle N_{SI} \sim Binomial\left(S, 1 - \exp\left(-\beta \frac{I}{N}\triangle t\right)\right) \tag{47}$$

$$\triangle N_{IR} \sim Binomial(I, 1 - \exp(-\gamma\triangle t)) \tag{48}$$

where $S + I + R = N$ give the number of susceptible, infected, and recovered individuals, respectively. The parameter β controls the transition rate from being susceptible to infected, and γ controls the transition rate from being infected to recovered. The number of individuals moving from S to I, given by $\triangle N_{SI}$, and the number of people moving from I to R, given by $\triangle N_{IR}$, over a time interval $\triangle t$ are modeled as binomial random variables. The above listed stochastic system has no analytic solution and thus requires numerical simulation methods for recovering parameter values from data. Cast as a parameter estimation task, the challenge is to recover $\boldsymbol{\theta} = \{\beta, \gamma\}$ from three dimensional time-series data $\boldsymbol{x}_{1:T}$ where each $\boldsymbol{x}_t \in \mathbb{N}^3$ is a triple containing the number of susceptible (S), number of infected (I), and recovered (R) individuals at time t.

During training of the networks, we simulate time-series from the stochastic SIR model with varying lengths. The number of time points T is drawn from a uniform distribution $T \sim \mathcal{U}(200, 500)$ at each training iteration. Usually, at lower Ts, the system has not converged to an equilibrium (i.e., not all individuals have transitioned from being I to R). Thus, it is especially interesting to see if BayesFlow can recover the rate parameters, even if the process dynamics are still unfolding over time. Training the networks took approximately two hours with the online learning approach. Inference on 1000 datasets with 2000 posterior samples per parameter took approximately 1.1 seconds.

The results on the SIR model are depicted in Figure 8. In line with the previous examples, we observe very good recovery of the true parameters, with NRMSE at $T = 500$ around 0.03, and R^2s around 0.99. Further, we observe decent performance even at smaller Ts, with better recovery performance as T increases. We also observe that the posterior variance shrinks, as T increases, Finally, the SCB plots indicate that the approximate posteriors are well
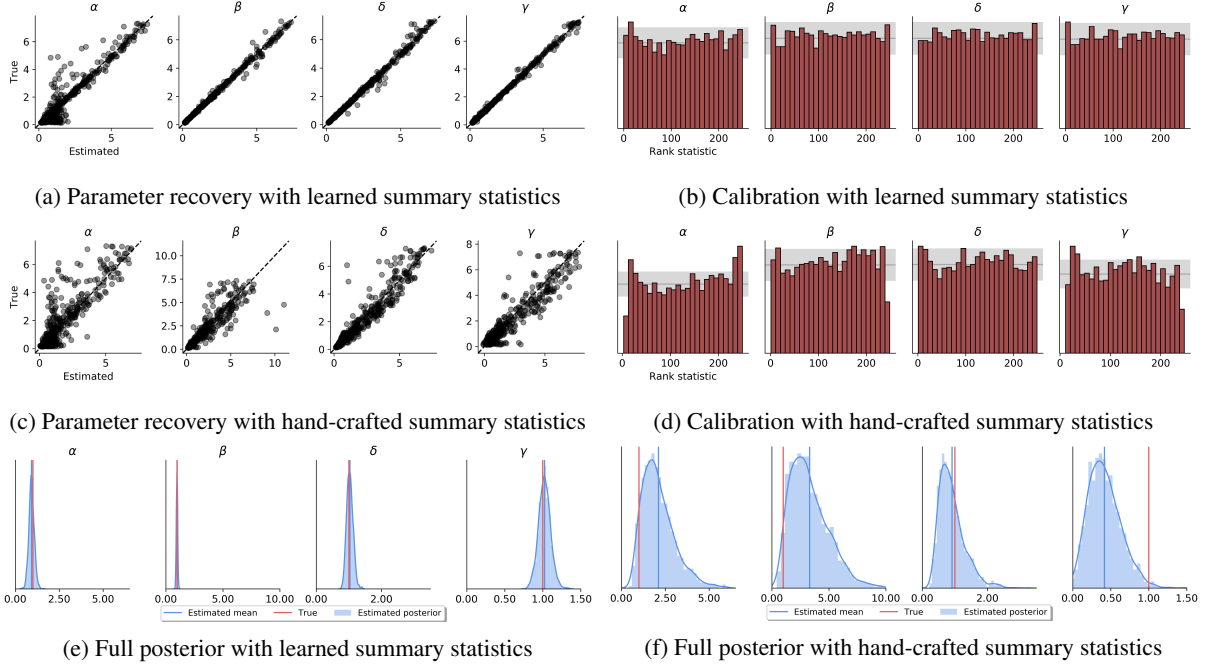
(a) Parameter recovery with learned summary statistics

(b) Calibration with learned summary statistics

(c) Parameter recovery with hand-crafted summary statistics

(d) Calibration with hand-crafted summary statistics

(e) Full posterior with learned summary statistics

(f) Full posterior with hand-crafted summary statistics

Figure 9: Comparison of recovery/calibration on the LV model with learned vs. hand-crafted summary statistics **(a)** Simulation-based calibration (SBC) with learned summary statistics; **(b)** Parameter recovery with learned summary statistics; **(c)** Parameter recovery with hand-crafted summary statistics; **(d)** Simulation-based calibration (SBC) with hand-crafted summary statistics; **(e)** Example full posteriors obtained on a single dataset with ground-truth rate parameters $\boldsymbol{\theta} = (1, 1, 1, 1)$ obtained with learned summaries; **(f)** The posterior obtained from the same dataset using hand-crafted summary statistics.

calibrated, with the approximate posterior mean of β slightly overestimating the true parameter values in the lower range.

3.8 Learned vs. Hand-Crafted Summaries: The Lotka-Volterra Population Model

With this final examples, we want to compare the performance of our method with an LSTM summary network vs. performance obtained with a standard set of hand-crafted summary statistics. For this, we focus on the well-studied Lotka-Volterra (LV) model. The LV model describes the dynamics of biological systems in which a population of predators interacts with a population of prey [57]. It involves a pair of first order, non-linear, differential equations given by:

$$\frac{d}{dt} = \alpha u - \beta u v \tag{49}$$

$$\frac{d}{dt} = -\gamma v + \delta \beta u v \tag{50}$$

where u denotes the number of preys, v denotes the number of predators, and the parameter vector controlling the interaction between the species is $\boldsymbol{\theta} = (\alpha, \beta, \gamma, \delta)$.

During training of the networks, we set the initial conditions as $u_0 = 10$ and $v_0 = 5$ and consider an interval $I_T = 15$ of discrete time units with $T = 500$ time steps (samples) in between. Each sample $\boldsymbol{x}_t$ in each LV time-series $\boldsymbol{x}_{1:T}$ is thus a 2-dimensional vector containing the number of prey and predators in the population at time unit t.

We train two invertible neural networks. The first is trained jointly with an LSTM summary network which outputs a 9-dimensional learned summary statistic $f_{\boldsymbol{\psi}}(\boldsymbol{x}_{1:T})$. The second uses a set of 9 typically used, hand-crafted summary statistics [42, 43], which include: the mean of the time series; the log variance of the time-series; the auto-correlation of each timeseries at lags 0.2 and 0.4 time units; the cross-correlation between the two time series. The same cINN architecture with 5 ACBs is used for both training scenarios. For each scenario, we perform the same number of iterations and epochs. Online learning for each training scenario took approximately 4 hours in total wall-clock time.

The results obtained on the LV model are depicted in Figure 9. We observe notably better recovery of the true parameter estimates when performing inference with the learned summary statistics. The approximate posteriors are also better calibrated when conditioned on the set of 9 learned summary statistics. These results highlight the advantages of using a summary networks when no sufficient summary statistics are available. Finally, Figure 9e and Figure 9f depict the posteriors obtained by the two different INNs on a single dataset with ground-truth parameters $\boldsymbol{\theta} = (1, 1, 1, 1)$. Evidently, learning the summary statistics leads to much sharper posteriors and better point-estimate recovery.

4   Discussion

In the current work, we proposed and explored a novel method which uses invertible neural networks to perform globally amortized approximate Bayesian inference. The method, which we named BayesFlow, requires only simulations from a forward model to learn a reusable probabilistic mapping between data and parameters. We demonstrated the utility of BayesFlow by applying it to models and data from various research domains. Further, we explored an online learning approach with variable number of observations per iteration. We demonstrated that this approach leads to excellent parameter estimation throughout the examples considered in the current work. In theory, BayesFlow is applicable to any mathematical process model which can be implemented as a computer simulation

BayesFlow combines the universal approximation capacity of deep learning methods [19] with the crucial uncertainty quantification assets of Bayesian inference [28, 16]. Besides being capable of performing fully Bayesian inference on intractable mathematical models, BayesFlow provides a general framework for designing reusable Bayesian parameter estimation machines for various research domains. Moreover, it could also prove as a viable alternative in modeling contexts where standard inference methods are available, but inference is nevertheless prohibitively slow. In the following, we highlight the main advantages of BayesFlow.

First, the introduction of separate summary and inference networks renders the method independent of the shape or the size of the observed data. The summary network learns a fixed-size vector representation of the data in an automatic, data-driven manner. Since the summary network is optimized jointly with the inference network, the learned data representation is encouraged to be maximally informative for inferring the parameters' posterior. This is particularly useful in settings where appropriate summary statistics are not known and, as a consequence, relevant information is lost through the choice of sub-optimal summary functions. However, if *sufficient* statistics are available in a given domain, one might omit the summary network altogether and feed these statistics directly to the invertible network.

Second, we showed that BayesFlow generates samples from the correct posterior under perfect convergence without distributional assumptions on the shape of the posterior. This is in contrast to variational methods which optimize a lower-bound on the posterior [29, 30], and oftentimes assume Gaussian approximate posteriors. Additionally, we also showed throughout all examples that the posterior means generated by the BayesFlow method are mostly excellent estimates for the true values. Beyond this, the fact that the BayesFlow method recovers the full posterior does not necessitate the usage of point estimates or summary statistics of the posterior. Further, we observe the desired posterior contraction (posterior variance decreases with increasing number of observations) and better recovery with increasing number of observations. These are indispensable properties of any Bayesian parameter estimation method, as they mirror the decrease in epistemic uncertainty and simultaneous increase in information due to availability of more data.

Third, the largest computational cost of BayesFlow is paid during the training phase. Once trained, the networks can be used and reused to perform inference on large numbers of data sets from a given research domain within seconds, given the same priors. Indeed, there are many instances of research domains where a single model is extensively explored and independently fitted by multiple researches to test scientific hypotheses [56, 60, 47, 11]. These research domains are expected to benefit the most from learning the *model universe* once and then inverting the model multiple times for fast inference on multiple data sets. In this regard, BayesFlow is similar to the recently introduced prepaid method [38] which uses a database of pre-computed summary statistics and nearest-neighbors for inference. Note, however, that BayesFlow does not need to store training data, since the *knowledge* about the relationship between data and parameters is compressed into the networks' weights. This not only makes the global sharing of pre-trained parameter estimation networks across researchers extremely easy, but also makes their local storage very efficient. Finally, all computations involved in the BayesFlow method benefit from a high degree of parallelism and can thus utilize the advantages of modern GPU acceleration.

These advantages notwithstanding, limitations of the BayesFlow method should also be mentioned. Although we could provide an asymptotic theoretical guarantee that BayesFlow recovers the true posteriors under perfect convergence, this might not be achieved in practice. Therefore, is it essential that proper calibration of point estimates and estimated posteriors is performed for each application of the method. Below, we discuss potential challenges and limitations of the method.

A first challenge is the Monte Carlo error introduced by approximating the mathematical expectations in our optimization objective (Eq.19) with a finite sum. The fact that we can loop through the training phase for a potentially unlimited amount of data should mitigate the Monte Carlo error. However, this would only apply to the online learning approach which is preferable when simulations are computationally cheap to perform. Approximation error remains a potential issue for the offline learning approach with a fixed grid or reference table of simulated data. Note, that this is a potential shortcoming of all methods for approximate inference using the reference table approach.

Second, the design of the summary network and invertible networks is a crucial choice for achieving optimal performance of the method. As already mentioned, the summary network should be able to represent the observed data without losing essential information and the invertible network should be powerful enough to encode the relevant data-generating process. Nevertheless, in some real-world scenarios there might be little guidance on how to actually construct suitable summary networks. Recent work on probabilistic symmetry [5] and algorithmic alignment [59] as well as our current experiments do, however, provide some insights about the design of summary networks. For instance, *i.i.d.* data induce a permutation invariant distribution which is well modeled with a deep invariant network [5]. Data with temporal or spatial dependencies are best modeled with recurrent [25], or convolutional [46] networks. When pairwise or multi-way relationships are particularly informative, attention [55] or graph networks [59] appear as reasonable choices. On the other hand, the depth of the invertible network should be tailored to the complexity of the mathematical model of interest. More ACBs will enable the network to encode more complex distributions but will increase training time. Very high-dimensional problems might also require very large networks with millions of parameters, up to a point where estimation becomes practically unfeasible. However, most mathematical models in the life sciences prioritize parsimony and interpretability, so they do not contain hundreds or thousands of latent parameters. In any case, future applications might require novel network architectures and solutions which go beyond our initial recommendations.

A third potential issue is the large number of neural network and optimization hyperparameters that might require fine-tuning by the user for optimal performance on a given task. However, we observe that many of the default hyperparameter values are sufficient to achieve excellent performance, and starting with a relatively large network consisting of 5 to 10 ACBs does not appear to hurt performance or destabilize training, even if the model to be learned is relatively simple. Based on our results, we expect that a single architecture should be able to perform well on models from a given domain. Future research should investigate this question of generality by applying the method to different or even competing models within different research domains. Future research should investigate the impact of modern hyperparameter optimization methods such as Bayesian optimization [15].

Finally, even though modern libraries such as *TensorFlow* [1] or *Torch* [8] allow for rapid and relatively straightforward development of various neural network architectures, the implementational burden associated with the current method is still reasonably high. In order to ease the understanding and independent application of the method, we provide fully functioning code to reproduce and study all of the examples tackled in this paper. In addition, we provide implementation of all metrics and visualization tools for performance validation used throughout the paper. Moreover, we are currently developing a general user-friendly software, which should abstract away most intricacies from the user of the method.

We hope that the new BayesFlow method will enable researchers from a variety of fields to accelerate model-based inference and will further prove its utility beyond the examples considered in this paper.

Acknowledgements

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

[2] Lynton Ardizzone, Jakob Kruse, Sebastian Wirkert, Daniel Rahner, Eric W Pellegrini, Ralf S Klessen, Lena Maier-Hein, Carsten Rother, and Ullrich Köthe. Analyzing inverse problems with invertible neural networks. *Proceedings ICLR 2019*, 2018.

[3] Lynton Ardizzone, Carsten Lüth, Jakob Kruse, Carsten Rother, and Ullrich Köthe. Guided image generation with conditional invertible neural networks. *arXiv preprint arXiv:1907.02392*, 2019.

[4] Daniele Bigoni, Olivier Zahm, Alessio Spantini, and Youssef Marzouk. Greedy inference with layers of lazy maps. *arXiv preprint arXiv:1906.00031*, 2019.

[5] Benjamin Bloem-Reddy and Yee Whye Teh. Probabilistic symmetry and invariant neural networks. *arXiv preprint arXiv:1901.06082*, 2019.

[6] William M Bolstad and James M Curran. *Introduction to Bayesian statistics*. John Wiley & Sons, 2016.

[7] Laming Chen, Guoxin Zhang, and Eric Zhou. Fast greedy map inference for determinantal point process to improve recommendation diversity. In *Advances in Neural Information Processing Systems*, pages 5622–5633, 2018.

[8] Ronan Collobert, Samy Bengio, and Johnny Mariéthoz. Torch: a modular machine learning software library. Technical report, Idiap, 2002.

[9] Kyle Cranmer, Johann Brehmer, and Gilles Louppe. The frontier of simulation-based inference. *arXiv preprint arXiv:1911.01429*, 2019.

[10] Katalin Csilléry, Michael GB Blum, Oscar E Gaggiotti, and Olivier François. Approximate bayesian computation (abc) in practice. *Trends in ecology & evolution*, 25(7):410–418, 2010.

[11] Perry De Valpine and Alan Hastings. Fitting population models incorporating process noise and observation error. *Ecological Monographs*, 72(1):57–76, 2002.

[12] Matthew C Deans. Maximally informative statistics for localization and mapping. In *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292)*, volume 2, pages 1824–1829. IEEE, 2002.

[13] Gianluca Detommaso, Jakob Kruse, Lynton Ardizzone, Carsten Rother, Ullrich Köthe, and Robert Scheichl. Hint: Hierarchical invertible neural transport for general and sequential bayesian inference. *arXiv preprint arXiv:1905.10687*, 2019.

[14] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

[15] Katharina Eggensperger, Matthias Feurer, Frank Hutter, James Bergstra, Jasper Snoek, Holger Hoos, and Kevin Leyton-Brown. Towards an empirical foundation for assessing bayesian optimization of hyperparameters. In *NIPS workshop on Bayesian Optimization in Theory and Practice*, volume 10, page 3, 2013.

[16] Andrew Gelman, Hal S Stern, John B Carlin, David B Dunson, Aki Vehtari, and Donald B Rubin. *Bayesian data analysis*. Chapman and Hall/CRC, 2013.

[17] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. 1999.

[18] Pedro J Gonçalves, Jan-Matthis Lueckmann, Michael Deistler, Marcel Nonnenmacher, Kaan Öcal, Giacomo Bassetto, Chaitanya Chintaluri, William F Podlaski, Sara A Haddad, Tim P Vogels, et al. Training deep neural density estimators to identify mechanistic models of neural dynamics. *bioRxiv*, page 838383, 2019.

[19] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[20] David S Greenberg, Marcel Nonnenmacher, and Jakob H Macke. Automatic posterior transformation for likelihood-free inference. *arXiv preprint arXiv:1905.07488*, 2019.

[21] Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *Journal of Machine Learning Research*, 13(Mar):723–773, 2012.

[22] Aditya Grover, Manik Dhar, and Stefano Ermon. Flow-gan: Combining maximum likelihood and adversarial learning in generative models. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[23] John R Hershey and Peder A Olsen. Approximating the kullback leibler divergence between gaussian mixture models. In *2007 IEEE International Conference on Acoustics, Speech and Signal Processing-ICASSP'07*, volume 4, pages IV–317. IEEE, 2007.

[24] Herbert W Hethcote. The mathematics of infectious diseases. *SIAM review*, 42(4):599–653, 2000.

[25] Seong Jae Hwang, Zirui Tao, Won Hwa Kim, and Vikas Singh. Conditional recurrent flow: Conditional generation of longitudinal samples with applications to neuroimaging. *arXiv preprint arXiv:1811.09897*, 2018.

[26] Bai Jiang, Tung-yu Wu, Charles Zheng, and Wing H Wong. Learning summary statistic for approximate bayesian computation via deep neural network. *Statistica Sinica*, pages 1595–1618, 2017.

[27] Matt J Keeling and Pejman Rohani. *Modeling infectious diseases in humans and animals*. Princeton University Press, 2011.

[28] Alex Kendall and Yarin Gal. What uncertainties do we need in bayesian deep learning for computer vision? In *Advances in neural information processing systems*, pages 5574–5584, 2017.

[29] D Kingma, Tim Salimans, R Josefowicz, Xi Chen, Ilya Sutskever, Max Welling, et al. Improving variational autoencoders with inverse autoregressive flow. 2017.

[30] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *stat*, 1050:1, 2014.

[31] Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems*, pages 10215–10224, 2018.

[32] Emmanuel Klinger, Dennis Rickert, and Jan Hasenauer. pyabc: distributed, likelihood-free inference. *Bioinformatics*, 34(20):3591–3593, 2018.

[33] Hongzhou Lin and Stefanie Jegelka. Resnet with one-neuron hidden layers is a universal approximator. In *Advances in Neural Information Processing Systems*, pages 6169–6178, 2018.

[34] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[35] Jan-Matthis Lueckmann, Pedro J Goncalves, Giacomo Bassetto, Kaan Öcal, Marcel Nonnenmacher, and Jakob H Macke. Flexible statistical inference for mechanistic models of neural dynamics. In *Advances in Neural Information Processing Systems*, pages 1289–1299, 2017.

[36] Ulf Kai Mertens. *Deep learning methods for likelihood-free inference: approximating the posterior distribution with convolutional neural networks*. PhD thesis, 2019.

[37] Ulf Kai Mertens, Andreas Voss, and Stefan Radev. Abrox—a user-friendly python module for approximate bayesian computation with a focus on model comparison. *PloS one*, 13(3):e0193981, 2018.

[38] Merijn Mestdagh, Stijn Verdonck, Kristof Meers, Tim Loossens, and Francis Tuerlinckx. Prepaid parameter estimation without likelihoods. *PLoS computational biology*, 15(9):e1007181, 2019.

[39] Steven Miletić, Brandon M Turner, Birte U Forstmann, and Leendert van Maanen. Parameter recovery for the leaky competing accumulator model. *Journal of Mathematical Psychology*, 76:25–50, 2017.

[40] Ashish Mishra, Shiva Krishna Reddy, Anurag Mittal, and Hema A Murthy. A generative model for zero shot learning using conditional variational autoencoders. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 2188–2196, 2018.

[41] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[42] George Papamakarios and Iain Murray. Fast ε-free inference of simulation models with bayesian conditional density estimation. In *Advances in Neural Information Processing Systems*, pages 1028–1036, 2016.

[43] George Papamakarios, David C Sterratt, and Iain Murray. Sequential neural likelihood: Fast likelihood-free inference with autoregressive flows. *arXiv preprint arXiv:1805.07226*, 2018.

[44] Mijung Park, Wittawat Jitkrittum, and Dino Sejdinovic. K2-abc: Approximate bayesian computation with kernel embeddings. 2016.

[45] Matthew D Parno and Youssef M Marzouk. Transport map accelerated markov chain monte carlo. *SIAM/ASA Journal on Uncertainty Quantification*, 6(2):645–682, 2018.

[46] Stefan T Radev, Ulf K Mertens, Andreas Voss, and Ullrich Köthe. Towards end-to-end likelihood-free inference with convolutional neural networks. *British Journal of Mathematical and Statistical Psychology*, 2019.

[47] Roger Ratcliff and Gail McKoon. The diffusion decision model: theory and data for two-choice decision tasks. *Neural computation*, 20(4):873–922, 2008.

[48] Louis Raynal, Jean-Michel Marin, Pierre Pudlo, Mathieu Ribatet, Christian P Robert, and Arnaud Estoup. Abc random forests for bayesian parameter inference. *Bioinformatics*, 35(10):1720–1728, 2018.

[49] Faryad Darabi Sahneh, Aram Vajdi, Heman Shakeri, Futing Fan, and Caterina Scoglio. Gemfsim: a stochastic simulator for the generalized epidemic modeling framework. *Journal of computational science*, 22:36–44, 2017.

[50] Scott A Sisson and Yanan Fan. *Likelihood-free MCMC*. Chapman & Hall/CRC, New York.[839], 2011.

[51] Mikael Sunnåker, Alberto Giovanni Busetto, Elina Numminen, Jukka Corander, Matthieu Foll, and Christophe Dessimoz. Approximate bayesian computation. *PLoS computational biology*, 9(1):e1002803, 2013.

[52] Sean Talts, Michael Betancourt, Daniel Simpson, Aki Vehtari, and Andrew Gelman. Validating bayesian inference algorithms with simulation-based calibration. *arXiv preprint arXiv:1804.06788*, 2018.

[53] Brandon M Turner and Per B Sederberg. A generalized, likelihood-free method for posterior estimation. *Psychonomic bulletin & review*, 21(2):227–250, 2014.

[54] Marius Usher and James L McClelland. The time course of perceptual choice: the leaky, competing accumulator model. *Psychological review*, 108(3):550, 2001.

[55] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[56] Andreas Voss, Veronika Lerche, Ulf Mertens, and Jochen Voss. Sequential sampling models with variable boundaries and non-normal noise: A comparison of six models. *Psychonomic bulletin & review*, pages 1–20, 2019.

[57] Darren J Wilkinson. *Stochastic modelling for systems biology*. CRC press, 2011.

[58] Simon N Wood. Statistical inference for noisy nonlinear ecological dynamic systems. *Nature*, 466(7310):1102, 2010.

[59] Keyulu Xu, Jingling Li, Mozhi Zhang, Simon S Du, Ken-ichi Kawarabayashi, and Stefanie Jegelka. What can neural networks reason about? *arXiv preprint arXiv:1905.13211*, 2019.

[60] Luke Zappia, Belinda Phipson, and Alicia Oshlack. Splatter: simulation of single-cell rna sequencing data. *Genome biology*, 18(1):174, 2017.

Appendix

A   Computation of Validation Metrics

Normalized Root Mean Squared Error

The normalized root mean squared error (NRMSE) between a sample of true parameters $\{\theta^{(i)}\}_{i=1}^{M}$ and a sample of estimated parameters $\{\hat{\theta}^{(i)}\}_{i=1}^{M}$ is given by:

$$NRMSE = \sqrt{\sum_{i=1}^{M} \frac{\left(\theta^{(i)} - \hat{\theta}^{(i)}\right)^2}{\theta_{max} - \theta_{min}}} \tag{1}$$

Due to the normalization factor $\theta_{max} - \theta_{min}$, the NRMSE is scale-independent, and thus suitable for comparing the recovery across parameters with different numerical ranges. The NRMSE equals zero when the estimates are exactly equal to the true values.

Coefficient of Determination

The coefficient of determination R^2 measures the proportion of variance in a sample of true parameters $\{\theta^{(i)}\}_{i=1}^{M}$ that is *explained* by a sample of estimated parameters $\{\hat{\theta}^{(i)}\}_{i=1}^{M}$. It is computed as:

$$R^2 = 1 - \sum_{i=1}^{M} \frac{\left(\theta^{(i)} - \hat{\theta}^{(i)}\right)^2}{\left(\theta^{(i)} - \bar{\theta}^{(i)}\right)^2} \tag{2}$$

where $\bar{\theta}$ denotes the mean of the true parameter samples. When R^2 equals 1, the estimates are perfect reconstructions of the true parameters.

Re-simulation Error

To compute the re-simulation error Err_{sim}, we first obtain an estimate of the true parameter value given an observed (validations) data set $\boldsymbol{x}_{1:N}^{o}$ by computing the mean of the approximate posterior $\tilde{\boldsymbol{\theta}}$. Then, we run the mathematical model to obtain a simulated dataset $\boldsymbol{x}_{1:N}^{s} = S(\tilde{\boldsymbol{\theta}}, \xi)$. Finally, we compute the maximum mean discrepancy (MMD, [21]) between the observed and the simulated dataset $MMD(\boldsymbol{x}_{1:N}^{o}, \boldsymbol{x}_{1:N}^{s})$. The MMD is a kernel-based metric which estimates the mismatch between two distributions given samples from the distributions by comparing all of their moments. It equals zero when the two distributions are equal almost everywhere [21]). Thus, a low MMD indicates that the distribution of $\boldsymbol{x}_{1:N}^{s}$ is close to the distribution of $\boldsymbol{x}_{1:N}^{o}$. Conversely, a high MMD indicates that the distribution of $\boldsymbol{x}_{1:N}^{s}$ is far from the distribution of $\boldsymbol{x}_{1:N}^{o}$. We report the median MMD computed over all validation data sets.

Calibration Error

The calibration error Err_{cal} quantifies how well the coverage of an approximate posterior matches the coverage of an unknown true posterior. Let α_θ be the fraction of true parameter values lying in a corresponding α-credible interval of the approximate posterior. Thus, for a perfectly calibrated approximate posterior, α_θ should equal α for all $\alpha \in (0, 1)$. We compute the calibration error for each marginal posterior as the median absolute deviation $|\alpha_\theta - \alpha|$ for 100 equally spaced values of $\alpha \in (0, 1)$. Therefore, the calibration error ranges between 0 and 1 with 0 indicating perfect calibration and 1 indicating complete miscalibration of the approximate posterior.

Kullback-Leibler Divergence

The Kullback-Leibler divergence ($\mathbb{KL}$) quantifies the increase in entropy incurred by approximating a target probability distribution P with a distribution Q. Its general form for absolutely continuous distributions is given by

$$\mathbb{KL}(P \,||\, Q) = \int_{-\infty}^{\infty} p(x) \log \frac{p(x)}{q(x)} dx \tag{3}$$

where p and q denote the pdfs of P and Q. In the case where P and Q are both multivariate Gaussian distributions, the KL divergence can be computed in closed form [23]:

$$\mathbb{KL}(P\,||\,Q) = \frac{1}{2}\left[\log\frac{\det\boldsymbol{\Sigma}_q}{\det\boldsymbol{\Sigma}_p} + \mathrm{Tr}(\boldsymbol{\Sigma}_q^{-1}\boldsymbol{\Sigma}_p) - d + (\boldsymbol{\mu}_p - \boldsymbol{\mu}_q)^T\boldsymbol{\Sigma}_q^{-1}(\boldsymbol{\mu}_p - \boldsymbol{\mu}_q)\right] \tag{4}$$

where $\boldsymbol{\Sigma}_p$ and $\boldsymbol{\Sigma}_q$ denote the covariance matrices of p and q, $\boldsymbol{\mu}_p$ and $\boldsymbol{\mu}_q$ the respective mean vectors, and d the number of dimensions of the Gaussian. In the case of diagonal Gaussian distributions, Eq.4 reduces to:

$$\mathbb{KL}(P\,||\,Q) = \sum_{i=1}^{d}\left(\log\frac{\sigma_{q,i}}{\sigma_{p,i}} + \frac{\sigma_{p,i}^2 + (\mu_{q,i} - \mu_{p,i})^2}{2\sigma_{q,i}^2} - \frac{1}{2}\right) \tag{5}$$

Even though the KL divergence is not a proper distance metric, as it is not symmetric in its arguments, it can be used to quantify the error of approximation when a closed-form solution is available.

Simulation-Based Calibration

Simulation-based calibration is a method to detect systematic biases in any Bayesian posterior sampling method [52]. It is based on the *self-consistency* of the Bayesian joint distribution. Given a sample from the prior distribution $\tilde{\theta} \sim p(\theta)$ and a sample from the data-generating process $\tilde{x} \sim p(x\,|\,\tilde{\theta})$, one can integrate $\tilde{\theta}$ and $\tilde{x}$ out of joint distribution and recover back the prior of θ:

$$p(\theta) = \int p(\theta, \tilde{\theta}, \tilde{x})d\tilde{x}d\tilde{\theta} \tag{6}$$

$$= \int p(\theta, \tilde{x}\,|\,\tilde{\theta})p(\tilde{\theta})d\tilde{x}d\tilde{\theta} \tag{7}$$

$$= \int p(\theta\,|\,\tilde{x})p(\tilde{x}\,|\,\tilde{\theta})p(\tilde{\theta})d\tilde{x}d\tilde{\theta} \tag{8}$$

If the Bayesian sampling method produces samples from the exact posterior, the equality implied by Eq.8 should hold regardless of the particular form of the posterior. Thus, any violation of this equality indicates some error incurred by the sampling method. The authors of [52] propose **Algorithm** 2 for visually detecting such violations:

Algorithm 2 Simulation-based calibration (SBC) for a single parameter θ

1: **for** $i = 1, ..., m$ **do**
2:     Sample $\tilde{\theta}^{(i)} \sim p(\theta)$
3:     Simulate a data set $\boldsymbol{x}_{1:N}^{(i)} = S(\tilde{\theta}^{(i)}, \xi)$
4:     Draw posterior samples $\{\theta^{(l)}\}_{l=1}^{L} \sim p_{\boldsymbol{\phi}}(\theta\,|\,\boldsymbol{x}_{1:N}^{(i)})$
5:     Compute rank statistic $r^{(i)} = \sum_{l=1}^{L}\mathbb{1}_{[\theta^{(l)}<\tilde{\theta}^{(i)}]}$
6:     Store $r^{(i)}$
7: **end for**
8: Create a histogram of $\{r^{(i)}\}_{i=1}^{M}$ and inspect it for uniformity

---

Algorithm 2 is correct, since Eq.8 implies that the rank statistic defined in line 5 should be uniformly distributed. Hence, any deviations from uniformity indicate some interpretable error in the approximate posterior [52].

B Model Details

The Ricker Model

Summary Network. We use a bidirectional long short-term memory (LSTM) recurrent neural network [17] for the raw Ricker time-series. The LSTM network architecture is a reasonable choice for this example, as it is able to

capture long-term dependencies in datasets with temporal or spatial autocorrelations. LSTMs can also easily deal with variable-length time-series.

Simulation. We place the following uniform priors over the Ricker model parameters:

$$\rho \sim \mathcal{U}(0, 15) \tag{9}$$
$$r \sim \mathcal{U}(1, 90) \tag{10}$$
$$\sigma \sim \mathcal{U}(0.05, 0.7) \tag{11}$$

These ranges appear to be very broad, as datasets generated by extreme parameter values appear implausible in real-world scenarios. Nevertheless, we stick to broad priors for training, even though parameter recovery might degrade at the extremes.

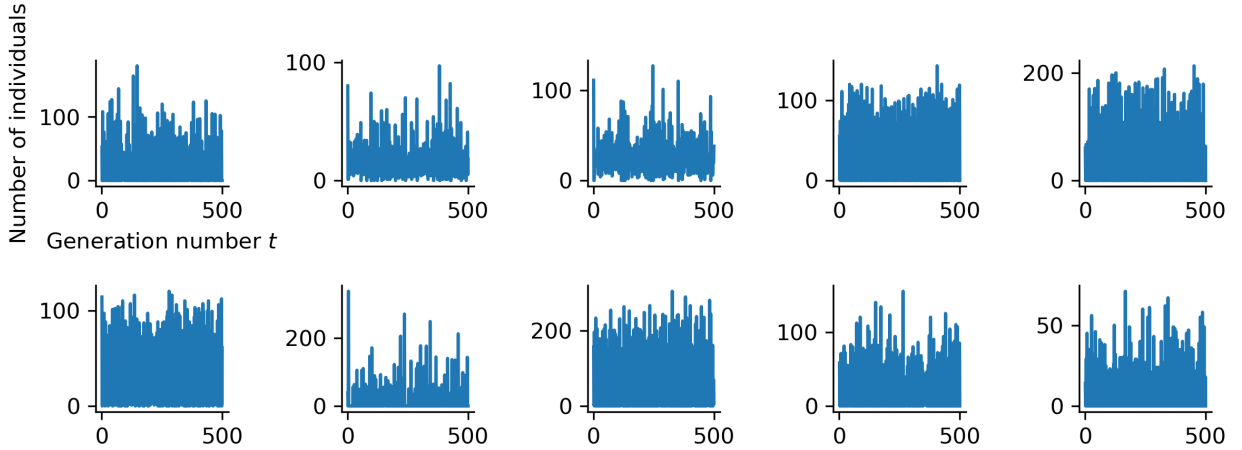Figure 1 depicts different simulated Ricker timeseries generated via draws from the prior.

Figure 1: Example Ricket data sets generated with different parameters.

The Lévy-Flight Model

Summary Network. We use a permutation invariant neural network [5] for the *i.i.d.* reaction times (RT) data. Similarly to the toy Regression example, each response in an RT data set is assumed to be independent of all others, so permutations of the data set must lead to the same parameter estimates.

Simulation. We place the following uniform priors over the LFM parameters, since they are broad enough to cover the range of realistic RT distributions encountered in empirical choice RT scenarios:

$$v_0 \sim \mathcal{U}(0, 6) \tag{12}$$
$$v_1 \sim \mathcal{U}(-6, 0) \tag{13}$$
$$zr \sim \mathcal{U}(0.3, 0.7) \tag{14}$$
$$a \sim \mathcal{U}(0.6, 3) \tag{15}$$
$$t_0 \sim \mathcal{U}(0.3, 1) \tag{16}$$
$$\alpha \sim \mathcal{U}(1, 2) \tag{17}$$

Figure 2 depitcs different simulated RT distributions generated via draws from the prior.
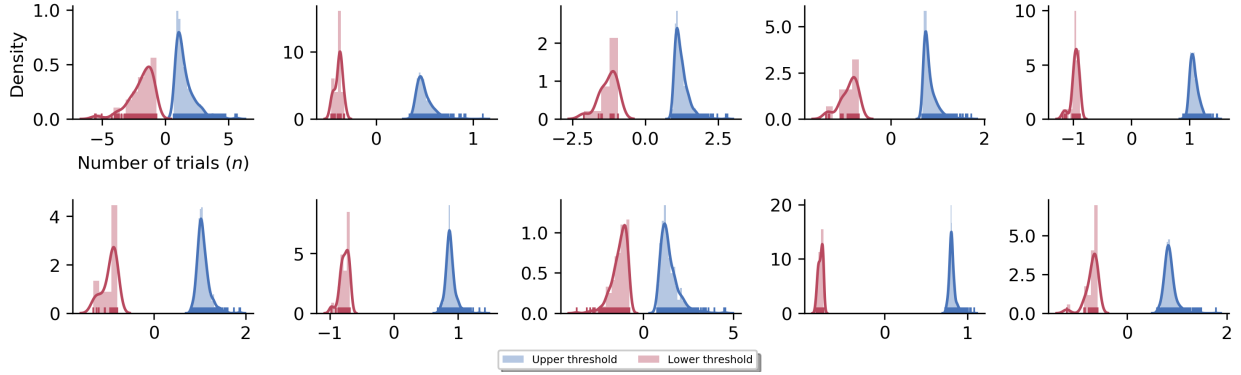
Figure 2: Example RT distributions generated with different parameters.

The Stochastic SIR Model

Summary Network. We use a $1D$ fully convolutional neural network [34] for the raw SIR time-series into fixed-size vectors. Here, we choose a convolutional network architecture over the previously mentioned LSTM, as convolutional networks are more computationally efficient. Further, we wanted to underline the utility of 1D convolutional networks for multidimensional time-series data. Finally, convolutional networks can also deal with variable input sizes.

Simulation. We place the following uniform priors over the two rate parameters of the stochastic SIR model:

$$\beta \sim \mathcal{U}(0.01, 1) \tag{18}$$
$$\gamma \sim \mathcal{U}(0.01, \beta) \tag{19}$$

These ranges were chosen based on empirical plausibility of the generated SIR time-series.

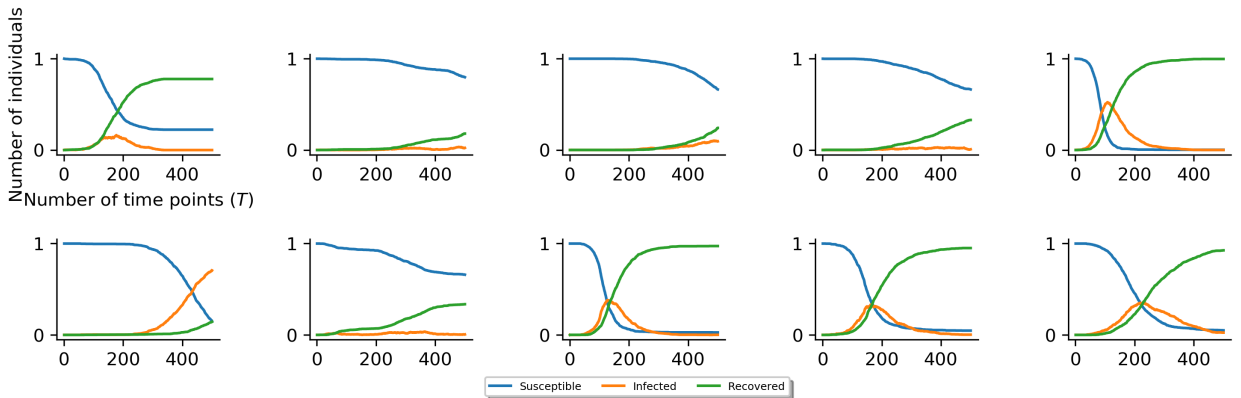Figure 3 depicts different SIR timeseries generated via draws from the prior.

Figure 3: Example SIR timeseries generated with different parameters.

The Lotka-Volterra Model

Summary Network. We use a bidirectional long short-term memory (LSTM) recurrent neural network [17] for the raw LV time-series (as in the Ricker example).

Simulation. We place the following broad uniform priors over the LV parameters. Some of the parameter combinations produced divergent simulations, which we removed during online learning.

4

$$\alpha \sim \mathcal{U}(\exp{(-2)}, \exp{(2)}) \tag{20}$$
$$\beta \sim \mathcal{U}(\exp{(-2)}, \exp{(2)}) \tag{21}$$
$$\gamma \sim \mathcal{U}(\exp{(-2)}, \exp{(2)}) \tag{22}$$
$$\delta \sim \mathcal{U}(\exp{(-2)}, \exp{(2)}) \tag{23}$$