2023 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)
June 28-30, 2023. Seattle, Washington, USA

Solving Stochastic Inverse Problems with Stochastic BayesFlow

1st Yi Zhang
Chair of Mechatronics
University of Augsburg
Augsburg, Germany
yi.zhang@informatik.uni-augsburg.de

2nd Lars Mikelsons
Chair of Mechatronics
University of Augsburg
Augsburg, Germany
lars.mikelsons@informatik.uni-augsburg.de

***Abstract*—Normalizing flows have gained increasing attention in the area of probabilistic modeling. For solving inverse problems, BayesFlow is a state-of-the-art Bayesian inference method based on normalizing flows. However, BayesFlow suffers from overfitting in many real-world scenarios. Therefore, we put forward stochastic BayesFlow, enhancing BayesFlow through stochastic normalizing flows. Apart from being less prone to overfitting, stochastic BayesFlow performs more robustly in parameter identification from noisy observations. Moreover, we develop a stochastic BayesFlow algorithm to solve stochastic inverse problems and validate it using the inverse uncertainty quantification of a simulated vehicle dynamics model.**

***Index Terms*—Neural Networks, Identification and Estimation in Mechatronics, Vehicle Control**

I. Introduction

The goal of solving an inverse problem is to identify the unknown parameters of a deterministic, physics-based model from the observed signals or measurements. When the observations are random processes originating from random parameters or the observations are affected by error, noise, and uncertainty, the problem turns into identifying the probability densities of the parameters, i.e., the stochastic inverse problem (SIP). SIPs can be numerically solved by determining the variation in the parameters from the variation in the model outputs with the help of Bayes' theorem. This approach was formulated as a measure-theoretic computational method [3]–[5]. Recently, machine learning has been utilized to solve SIPs. For example, generative adversarial networks were used to minimize the divergence between the distributions of experimental observations and model outputs [6]. However, this approach requires the prior distributions to be given in advance, which is usually not possible in practice.

Besides, due to intractable noises in observed signals and stochasticity in the physical model, a closed-form posterior distribution of parameters is hardly possible or precise. Therefore, we propose a likelihood-free algorithm for solving SIPs by disintegrating an SIP into a group of deterministic inverse problems, i.e., identifying parameters from each observation, and then reconstructing the estimated data points into probability densities of the random parameters. In order to acquire reliable parameter identification solutions, we apply the globally amortized Bayesian inference method BayesFlow [1], which is based on conditional normalizing flows [2]. In this method, simulation is used to learn a probabilistic mapping from observed data to underlying model parameters. In practice, the method has been proven to be well-performed in end-of-line testing of MEMS (micro-electro-mechanical systems) sensors [7]. However, in this use case, BayesFlow shows its tendency to overfit. Inspired by the novel method of stochastic normalizing flows [9], we improve BayesFlow to "stochastic BayesFlow" by introducing a stochastic element into conditional normalizing flows. The main contributions of our work are as follows:

- We propose stochastic BayesFlow as the extension of the original BayesFlow, contributing to avoiding overfitting to some extent with limited training data.
- We summarize and validate an algorithm for solving SIPs with (stochastic) BayesFlow using the inverse uncertainty quantification of a single-track vehicle model.
- We also show that the stochastic BayesFlow outperforms BayesFlow and BNN in terms of the accuracy and precision of parameter identification, even with noisy observed data.

II. Problem Formulation

We denote $\boldsymbol{\Theta} = [\Theta_1, \dots, \Theta_D]^T$ as a vector of random variables and $\boldsymbol{\mathcal{Y}}(t)$ as corresponding random processes in an SIP, whereas $\boldsymbol{\theta} = [\theta_1, \dots, \theta_D]^T$ as a vector of deterministic parameters and $\boldsymbol{y}(t)$ as the corresponding deterministic model outputs.

In an SIP, $\boldsymbol{\Theta}$ is supposed to be determined from $\boldsymbol{\mathcal{Y}}(t)$, which contains time-varying noise $\boldsymbol{\epsilon}(t)$. $\boldsymbol{\mathcal{Y}}(t)$ includes N observable time series obtained from a physical model, which is formulated as a forward map g [10]:

$$g : (\boldsymbol{\Theta} \in \mathbb{R}^D) \mapsto (\boldsymbol{\mathcal{Y}}(t) \in \mathbb{R}^N)\,; \tag{1a}$$

$$\begin{bmatrix} \Theta_1 \sim p_1(\theta_1) \\ \vdots \\ \Theta_D \sim p_D(\theta_D) \end{bmatrix} \mapsto \begin{bmatrix} \mathcal{Y}_1(t) = g_1(\boldsymbol{\Theta}, t) + \epsilon_1(t) \\ \vdots \\ \mathcal{Y}_N(t) = g_N(\boldsymbol{\Theta}, t) + \epsilon_N(t) \end{bmatrix}. \tag{1b}$$

Thus, the goal of an SIP is to estimate the prior distributions of the model parameters $p_d(\theta_d), d = 1, \dots, D$, given $\boldsymbol{\mathcal{Y}}(t)$ and g.

Considering that all parameters are independent of each other, for the parameter θ_d its probability density $p_d(\theta_d)$ is

$$p_d(\theta_d) = \int p_d(\theta_d|\boldsymbol{\mathcal{Y}}(t))p(\boldsymbol{\mathcal{Y}}(t))d\boldsymbol{\mathcal{Y}}(t) \tag{2a}$$
$$= \mathbb{E}_{\boldsymbol{\mathcal{Y}}(t)}[p_d(\theta_d|\boldsymbol{\mathcal{Y}}(t))] \tag{2b}$$
$$\approx \frac{1}{K}\sum_{k=1}^{K} p(\theta_d^{(k)}|\boldsymbol{y}^{(k)}(t))\,. \tag{2c}$$

At a sufficient K, the expectation of θ_d's posterior distribution over the random processes $\boldsymbol{\mathcal{Y}}(t)$ (2b) can be approximated through Monte-Carlo estimation into the mean of K posterior distributions over deterministic model outputs $\boldsymbol{y}(t)$ (2c). In this way, we can obtain the prior distribution $p_d(\theta_d)$ indirectly by estimating K posterior distributions $p(\theta_d^{(k)}|\boldsymbol{y}^{(k)}(t)), k = 1, \ldots, K$.

III. Methods

We utilize a Bayesian inference method called BayesFlow to learn the posteriors of parameters. BayesFlow is built up with conditional normalizing flows and a summary network III-A. In order to better model periodic time series, we adapt the summary network part of BayesFlow with a long- and short-term time-series network (LSTNet) [11] III-B. Furthermore, the conditional normalizing flows are extended into conditional stochastic normalizing flows III-C, aiming to enhance the generalization ability. We refer to the method as stochastic BayesFlow III-D. To conclude, we summarize the overall algorithm for solving SIPs with (stochastic) BayesFlow in Section III-E.

A. Conditional normalizing flows and BayesFlow

Normalizing flows learn complex distributions by transforming a standard normal distribution of the latent variable $\boldsymbol{z}$ through an invertible mapping [13]–[15]. Under the condition of model observations $\boldsymbol{y}(t)$, the invertible mapping $f_{\boldsymbol{\phi}}$ models the posterior densities $p(\boldsymbol{\theta}|\boldsymbol{y}(t))$ from $p(\boldsymbol{z})$. Accordingly, $f_{\boldsymbol{\phi}}$ is called a conditional normalizing flow (cNF). The architecture of the cNF, also called the conditional invertible neural network (cINN) [12], is built up with a group of affine coupling blocks, ensuring that the network is invertible, bijective, and has an easily calculable Jacobian determinant [16].

Developed on the cNFs, BayesFlow is built up with a summary network and a cINN. The summary network $h_{\boldsymbol{\varphi}}$ extracts statistical information $\widetilde{\boldsymbol{y}}$ from time series $\boldsymbol{y}(t)$. In the forward process, the cINN is trained to map the parameter $\boldsymbol{\theta}$ to $\boldsymbol{z}$-space under the condition of $\widetilde{\boldsymbol{y}}$, whereas in the inverse process, $\boldsymbol{\theta}$ can be inferred from the $\boldsymbol{z}$-space under the same condition. The architecture of BayesFlow is illustrated in Figure 1.

Both the summary network and the cINN are optimized collectively via back-propagation by minimizing the Kullback-Leibler divergence [17] between the true and the model induced posterior of $\boldsymbol{\theta}$. The objective function is

$$\min_{\boldsymbol{\phi},\boldsymbol{\varphi}} \mathbb{E}_{\boldsymbol{y}(t)}[\mathbb{KL}(p(\boldsymbol{\theta}|\boldsymbol{y}(t)) \parallel p_{\boldsymbol{\phi},\boldsymbol{\varphi}}(\boldsymbol{\theta}|\boldsymbol{y}(t)))]. \tag{3}$$

Via the change-of-variables formula of probability, the posterior is reformulated into

$$p_{\boldsymbol{\phi}}(\boldsymbol{\theta}|\widetilde{\boldsymbol{y}}) = p(\boldsymbol{z})|\det \boldsymbol{J}_{f_{\boldsymbol{\phi}}}|,\ \boldsymbol{z} = f_{\boldsymbol{\phi}}(\boldsymbol{\theta};\widetilde{\boldsymbol{y}}), \tag{4}$$

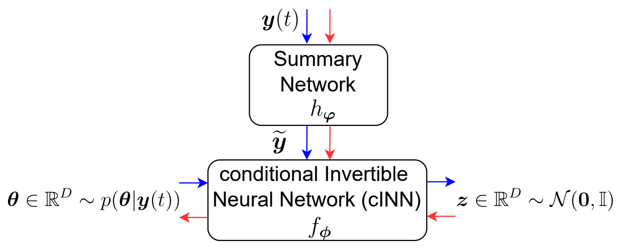

$\boldsymbol{y}(t)$
Summary Network
$h_{\boldsymbol{\varphi}}$
$\widetilde{\boldsymbol{y}}$
conditional Invertible Neural Network (cINN)
$f_{\boldsymbol{\phi}}$
$\boldsymbol{\theta} \in \mathbb{R}^D \sim p(\boldsymbol{\theta}|\boldsymbol{y}(t))$
$\boldsymbol{z} \in \mathbb{R}^D \sim \mathcal{N}(\boldsymbol{0}, \mathbb{I})$

Fig. 1. BayesFlow architecture. $\boldsymbol{\varphi}$, $\boldsymbol{\phi}$ denote the network parameters of the summary network h and the cINN f, respectively. The blue arrows represent the forward training process; the red arrows represent the inverse inference process. In the inverse process, only the cINN part is inverted, and the summary network part remains the same as that in the forward process.

where $\widetilde{\boldsymbol{y}} = h_{\boldsymbol{\varphi}}(\boldsymbol{y}(t))$ and $\boldsymbol{J}_{f_{\boldsymbol{\phi}}}$ is the Jacobian matrix of $f_{\boldsymbol{\phi}}$.

Approximating the expectation with Monte-Carlo estimation with M training samples $(\boldsymbol{y}^{(m)}(t), \boldsymbol{\theta}^{(m)}), m = 1, \ldots, M$ from the dataset, the objective function in Equation (3) becomes

$$\min_{\boldsymbol{\phi},\boldsymbol{\varphi}} \frac{1}{M}\sum_{m=1}^{M}\left(\frac{\left\| f_{\boldsymbol{\phi}}(\boldsymbol{\theta}^{(m)}; h_{\boldsymbol{\varphi}}(\boldsymbol{y}^{(m)}(t)))\right\|^2}{2} - log|\det \boldsymbol{J}_{f_{\boldsymbol{\phi}}}|\right). \tag{5}$$

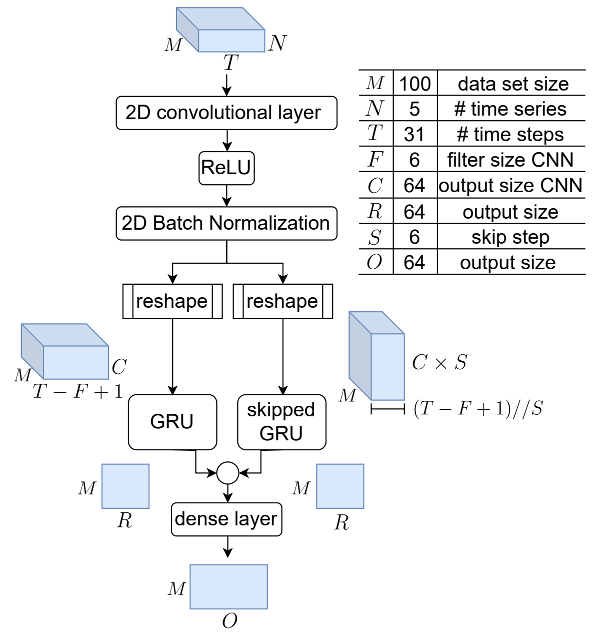
After the BayesFlow network is well trained, in the inverse direction, a single parameter estimate for the test observation $\boldsymbol{y}^{(k)}(t)$ can be derived by sampling the latent variable $\boldsymbol{z}$ for one time with the optimized network parameters $\widehat{\boldsymbol{\phi}}$, $\widehat{\boldsymbol{\varphi}}$. When $\boldsymbol{z}$ is sampled for B times, the posterior distribution $p_{\widehat{\boldsymbol{\phi}},\widehat{\boldsymbol{\varphi}}}(\boldsymbol{\theta}^{(k)}|\boldsymbol{y}^{(k)}(t))$ can be obtained. The estimated parameters are taken as the mean of the implicit posteriors:

$$\widehat{\boldsymbol{\theta}}^{(k)} = \frac{1}{B}\sum_{b=1}^{B} f_{\widehat{\boldsymbol{\phi}}}^{-1}(\boldsymbol{z}^{(b)}; h_{\widehat{\boldsymbol{\varphi}}}(\boldsymbol{y}^{(k)}(t))). \tag{6}$$

B. Summary network with LSTNet

The summary network is supposed to be adapted to the observations. For modeling time series, the summary network is typically realized by the long short-term memory (LSTM) network [18], which is also applied in the original work of BayesFlow [1]. When the observed time series is periodic, however, the LSTM tends to forget the long-term information, resulting in incorrect inferences. In our use case of the vehicle dynamics model, some of the observations are periodic. We conquer this problem by using the LSTNet [11] in the summary network. The architecture is demonstrated in Figure 2.

The first layer of LSTNet is a 2-dimensional convolutional neural layer, aiming to extract short-term patterns in the time dimension as well as local dependencies between variables. We insert a batch normalization layer after the ReLU activation function in order to accelerate the training process [20]. Subsequently, on the one hand, the time series is fed into a regular gated recurrent unit (GRU) network [19]. On the other hand, the time series is split into S slices and stacked into the channel dimension. The S is the pre-defined skip step and can be taken as the period of the time series. In the end, both outputs from GRU and skipped GRU are combined together by a dense layer.



M	100	data set size
N	5	# time series
T	31	# time steps
F	6	filter size CNN
C	64	output size CNN
R	64	output size
S	6	skip step
O	64	output size

Fig. 2. LSTNet architecture.

C. Conditional stochastic normalizing flows

Stochastic normalizing flows (SNFs) combine stochastic sampling methods with deterministic normalizing flows [8]. Theoretically, an SNF with I deterministic invertible blocks is equivalent to a pair of Markov Chains $((U_0, \dots, U_I), (V_I, \dots, V_0))$, as explained and proved in [8], [9]. For a conditional stochastic normalizing flow (cSNF), the conditional joint distributions are

$$P_{(U_0,\dots,U_I)|Y} = P_{U_0|Y} \cdot P_{U_1|U_0,Y} \cdot \ \cdots \ \cdot P_{U_I|U_{I-1},Y}; \quad (7a)$$
$$P_{(V_I,\dots,V_0)|Y} = P_{V_I|Y} \cdot P_{V_{I-1}|V_I,Y} \cdot \ \cdots \ \cdot P_{V_0|V_1,Y}. \quad (7b)$$

Different from deterministic normalizing flows, the loss function for a cSNF is the KL-divergence between the conditional joint distributions of forward and inverse processes:

$$\mathcal{L}_{cSNF} = \mathbb{KL}(P_{(U_0,\dots,U_I)|Y} || P_{(V_0,\dots,V_I)|Y}). \quad (8)$$

In the framework of BayesFlow, $Y \equiv \boldsymbol{y}(t)$, $U_0 \equiv p(\boldsymbol{\theta}_0)$, $V_I \equiv p(\boldsymbol{z})$, $V_0 \equiv p(\boldsymbol{\theta}|\boldsymbol{y}(t))$, and U_I is the forward output. We denote $\boldsymbol{\theta}_i, i = 1, \dots, I$ as the forward output of each deterministic layer. The loss function of cSNF in Equation (8) is expanded into:

$$\widetilde{\mathcal{L}}_{cSNF} = \mathbb{E}_{\boldsymbol{y}(t)}[\mathbb{E}_{(\boldsymbol{\theta}_1,\dots,\boldsymbol{\theta}_I)\sim P_{(U_0,\dots,U_I)|\boldsymbol{y}(t)}}[log(p(\boldsymbol{\theta}_0)) - log(p_Z(\boldsymbol{\theta}_I)) - \sum_{i=1}^{I} log f_i(\boldsymbol{\theta}_{i-1}, \boldsymbol{\theta}_i) \cdot \mathcal{T}_i(\boldsymbol{\theta}_{i-1}, \boldsymbol{\theta}_i)]]; \quad (9a)$$

$$f_i(\cdot, \boldsymbol{\theta}_i) = \frac{dP_{V_{i-1}|V_i=\boldsymbol{\theta}_i}}{dP_{U_{i-1}|U_i=\boldsymbol{\theta}_i}}; \quad (9b)$$

$$\mathcal{T}_i(\boldsymbol{\theta}_{i-1}, \boldsymbol{\theta}_i) = \frac{p_{V_i|\boldsymbol{y}(t)}(\boldsymbol{\theta}_i)}{p_{V_{i-1}|\boldsymbol{y}(t)}(\boldsymbol{\theta}_{i-1})}. \quad (9c)$$

When the layer i is a deterministic cINN, $f_i(\cdot, \boldsymbol{\theta}_i) = 1$ and $\mathcal{T}_i(\boldsymbol{\theta}_{i-1}, \boldsymbol{\theta}_i) = |\det \boldsymbol{J}_{f_{\boldsymbol{\phi}}}|$. Otherwise, the two terms are determined by the stochastic sampling layer. In cases where the cSNF does not contain any stochastic sampling layers, the equation is the same as the loss function of BayesFlow (5).

D. Stochastic BayesFlow

We implement a Markov Chain Monte Carlo (MCMC) [21] sampling layer at the end of the cINN and just before the latent variable $\boldsymbol{z}$. In this way, BayesFlow becomes stochastic.

Assuming U_i' is a random variable as the candidate of U_i. The joint distribution between U_i' and U_{i-1} is

$$P_{U_{i-1},U_i'} = P_{U_{i-1}} \cdot Q_i, \quad (10)$$

where Q_i is the Markov kernel. The kernel probability density function $q_i(\cdot|x)$ is chosen as Gaussian distribution $\mathcal{N}(\cdot|x, \sigma^2\mathbb{I})$. Assuming a uniformly distributed random variable $\eta \sim \mathcal{U}(0, 1)$, the next random state U_i is

$$U_i = U_{i-1} + 1_{[\eta,1]}(a(U_{i-1}, U_i'))\zeta_i, \ \zeta_i \sim \mathcal{N}(\boldsymbol{0}, \sigma^2\mathbb{I}). \quad (11)$$

$1_{[\eta,1]}(\cdot)$ denotes if the acceptance ratio is not smaller than η, then $U_i = U_{i-1} + \zeta_i$; otherwise, $U_i = U_{i-1}$. The term $a_i(x, y)$ is the acceptance ratio,

$$a_i(x, y) := min\left\{1, \frac{p_i(y)q_i(y|x)}{p_i(x)q_i(x|y)}\right\}. \quad (12)$$

In stochastic BayesFlow, the target density $p_i(\cdot)$ is the distribution of latent variable $\boldsymbol{z}$, i.e., $\mathcal{N}(\boldsymbol{0}, \mathbb{I})$. This process is also called Metropolis-Hastings algorithm [21].

E. Algorithm for solving SIPs

The overall algorithm is summarized in Algorithm 1.

Algorithm 1: Algorithm for solving SIPs

```
input :
    • value range of each parameter
      $[\theta_{d,min}, \theta_{d,max}], d = 1, \dots, D$;
    • the forward map $g$;
    • measurements of model $\boldsymbol{\mathcal{Y}}(t)$.
Sample parameters
  $\theta_d \sim \mathcal{U}(\theta_{d,min}, \theta_{d,max}), d = 1, \dots, D$.
Generate training set $(\boldsymbol{y}^{(m)}(t), \boldsymbol{\theta}^{(m)}), m = 1, \dots, M$,
  where $\boldsymbol{y}^{(m)}(t) = g(\boldsymbol{\theta}^{(m)})$.
Train (stochastic) BayesFlow network $h_{\boldsymbol{\varphi}}$, $f_{\boldsymbol{\phi}}$ with the
  training set.
Perform parameter inference with well-trained network
  $h_{\widehat{\boldsymbol{\varphi}}}$, $f_{\widehat{\boldsymbol{\phi}}}$:
for $k$ in $1, \dots, K$ do
    calculate $\widehat{\boldsymbol{\theta}}^{(k)}$ according to Equation (6).
end
Construct the prior distributions of parameters $p(\boldsymbol{\theta})$:
for $d$ in $1, \dots, D$ do
    calculate $p_d(\theta_d)$ according to Equation (2c).
end
output: the prior distributions of parameters $p(\boldsymbol{\theta})$.
```

Given the possible value ranges of parameters and the underlying physical model, for example, the ordinary differential equations, the prior distributions of parameters can be obtained from the measurements of model outputs. At first, a training set should be generated with the forward map by sampling the parameters randomly in their value ranges. Because the prior distributions are unknown, the sampled data points are supposed to follow uniform distributions. Next, the (stochastic) BayesFlow network is trained to learn the posterior distributions of parameters from corresponding model outputs. After the network is well trained, the posteriors of the parameters can be estimated through the measurements. At last, the prior distribution is reconstructed by the estimated parameters, which are the means of the posteriors.

IV. Experiments and Results

We conduct the experiments with a nonlinear single-track (ST) vehicle model, which is referred to the CommonRoad vehicle models [22]. The ST model is a reasonable simplification of the real vehicle dynamics and can be fast solved by a regular ordinary differential equation solver. The model is called "single-track" because the front and rear wheel pairs are each considered one wheel. Besides, the drifting and slipping effects of wheels are also calculated in both lateral and longitudinal directions. The random parameters of interest are the initial lateral position of center-of-gravity y_0, the friction coefficient μ, and the distance between vehicle center-of-gravity and front axle l_f. We sample the parameters for the training set following the uniform distributions $y_0 \sim \mathcal{U}(-1,4)$, $\mu \sim \mathcal{U}(0,2)$, $l_f \sim \mathcal{U}(0.5,2.5)$ respectively. For test set, the target distributions are $y_0 \sim \mathcal{U}(-0.5,3)$, $\mu \sim \mathcal{N}(1,0.1^2)$, $l_f \sim \mathcal{U}(1.5,0.15^2)$. Correspondingly, five model outputs are observed: the steering angle of the font wheel δ; yaw angle ψ, yaw rate $\dot{\psi}$, slide slip angle β and the lateral position s_y of center-of-gravity. The deterministic input signals $u_1(t), u_2(t)$ are:

$$u_1(t) := v_\delta(t) = 0.2sin(2\pi t), \tag{13a}$$

$$u_2(t) := a_x(t) = cos(\int (0.1 + (3.0-0.1)t/3)dt), \tag{13b}$$

where t is from $0\,s$ to $3\,s$. The model's inputs and outputs are sampled with a step size of $0.1\,s$. Note that the differential equation is solved with a variable time step that is possibly smaller. $v_\delta(t)$ is the steering velocity, and $a_x(t)$ is the longitudinal acceleration.

Firstly, we investigate the choices of the summary network in BayesFlow in Section IV-A, i.e., LSTM and LSTNet. Secondly, in Section IV-B, we compare BayesFlow and stochastic BayesFlow with the Bayesian Neural Network (BNN), which is another state-of-the-art probabilistic modeling method [23]. Thirdly, we determine the sufficient and necessary training set size in Section IV-C. Fourthly, we challenge the models by increasing the noise factor in the observed time series. In order to avoid overfitting, in the following experiments, we implement dropout in the cINN of BayesFlow with a dropout rate of 0.1. Finally, in Section IV-E, we perform inverse uncertainty quantification for stochastic BayesFlow. The code for the experiments is available at https://github.com/yiyi1zhang/stochastic_bayesflow.

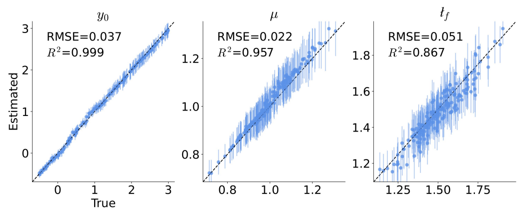
We evaluate the models from the following perspectives:

- quality of parameter inference: We sample the latent space of (stochastic) BayesFlow for 100 times ($B = 100$), and obtain the posterior distributions. We consider the following metrics:
  - for accuracy of parameter estimate: NRMSE (normalized root mean squared error),
  - for precision of parameter estimate: NMCIW (normalized mean confidence interval width),
  - for examination the actual frequency that the confidence interval contains the ground truth: CICP (confidence interval coverage probability);

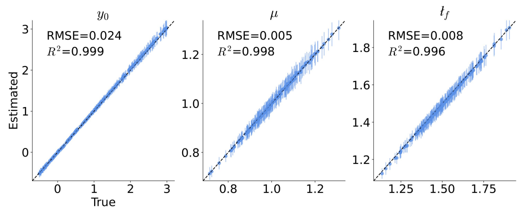
  The details of above mentioned three metrics can be found in [7].
- quality of likelihood reconstruction: We sample the prior density to be estimated for 200 times ($K = 200$) and identify all of the parameters. After that, we reconstruct the prior density with the estimated parameters, including the confidence intervals. To compare the target and reconstructed likelihoods, we compute the Bhattacharyya distance [24] between the kernel distribution estimates [25] of the estimated and true parameters.
- examination of overfitting: Except for the Bayesian inference results, we monitor the development of training and validation loss as well.

A. LSTM vs. LSTNet as summary network

We train 400 samples with LSTM or LSTNet in the summary network of BayesFlow. Figure 3 shows that the parameter inference results are obviously more precise, when the LSTNet is applied.



(a) LSTM in the summary network of BayesFlow



(b) LSTNet in the summary network of BayesFlow

Fig. 3. Parameter inference results with different summary networks of BayesFlow. The data points represent the mean values of the parameter posteriors, and the vertical lines represent the corresponding confidence intervals. The black dashed lines are the reference lines. When a parameter is correctly estimated, the corresponding blue point should be located on the reference line. $RMSE$ means root mean squared error, and R^2 is the coefficient of determination. The accuracy of estimates of three parameters in terms of RMSE is improved by 35.1 %, 77.3 % and 84.3 %, when LSTNet is used in the summary network.

B. BNN, BayesFlow vs. stochastic BayesFlow

Using the same 400 training samples, we train a BNN with the same summary network as (stochastic) BayesFlow and replace the cINN with two linear BNN layers. All three approaches achieve 100 % CICP. Table I shows the comparison in terms of NRMSE, NMCIW, and D_{BH} of the three parameters y_0, μ, and l_f. Both BayesFlow and stochastic BayesFlow estimate the posteriors much more precisely than the BNN, as indicated by the much smaller NRMSE. In the meantime, the more narrow NMCIW under the condition of 100 % CICP suggests that the estimated posteriors by BayesFlow and stochastic BayesFlow are reliable as well.

TABLE I
BNN, BayesFlow, vs. stochastic BayesFlow

y_0,μ,l_f	NRMSE			NMCIW			D_{BH}		
BNN	0.012	0.037	0.038	0.043	0.260	0.206	0.096	0.017	0.017
BayesFlow	0.007	0.008	**0.011**	0.051	0.109	0.099	0.098	0.015	0.015
s. BayesFlow	**0.001**	**0.005**	0.012	0.017	0.059	0.140	0.102	0.014	0.016

With regard to D_{BH}, the three methods does not distinguish themselves significantly. The more detailed parameter distributions are illustrated in Figure 4.

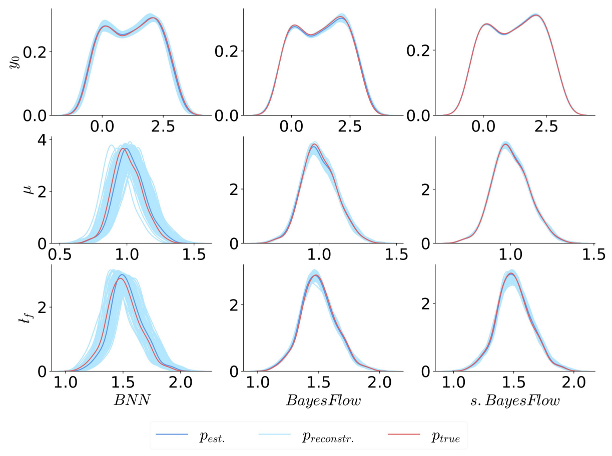


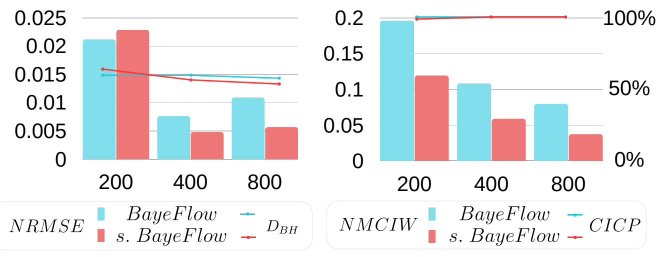
Fig. 4. Comparison of kernel density estimates of the three parameters y_0, μ, l_f. In each diagram, the red line represents the ground truth distribution, the blue line represents the estimated value distribution, and the light blue bands represent the reconstructed distributions obtained by running BNN or sampling the latent space of (stochastic) BayesFlow for B times, i.e., $\widehat{\theta}_d^{(k,b)}, b = 1, \dots, B, k = 1, \dots, K$.

In conclusion, BayesFlow and stochastic BayesFlow are able to reconstruct all three distributions almost perfectly, whereas the BNN might not be adequately precise and reliable.

C. Varying size of training set

We train BayesFlow and stochastic BayesFlow with 200, 400, and 800 training samples, respectively. Figure 5 compares the metrics NRMSE, D_{BH}, NMCIW, and CICP for the parameter μ.

When the training set size is doubled from 200 to 400, both the NRMSE and NMCIW of BayesFlow and stochastic BayesFlow drop sharply. The NRMSE of BayesFlow at 400 is about 1/3 of that at 200, while the NRMSE of stochastic BayesFlow at 400 is about 1/5 of that at 200. However, when



(a) NRMSE & D_{BH} (b) NMCIW & CICP

Fig. 5. Comparison of inference results for μ between BayesFlow and stochastic BayesFlow with different sizes of training set size.

the training set size is increased to 800, the inference precision of BayesFlow and stochastic BayesFlow does not improve significantly. At the same time, the difference between the reconstructed and the true prior distributions remains at the same level. Thus, the training set with 400 samples is sufficient and necessary to acquire precise and reliable probability density estimates of parameters.

Compared with BayesFlow, stochastic BayesFlow achieves lower NRMSE and NMCIW while keeping CICP at 100 %. In addition, stochastic BayesFlow has more advantages than BayesFlow in respect of avoiding overfitting. The training and validation losses of the two networks are monitored, as shown in Figure 6. A very high validation loss can occur in some epochs of BayesFlow. Stochastic BayesFlow, on the other hand, requires more training time to converge due to the stochastic sampling layer. Besides, the number of epochs is related to the stopping criterion of training.

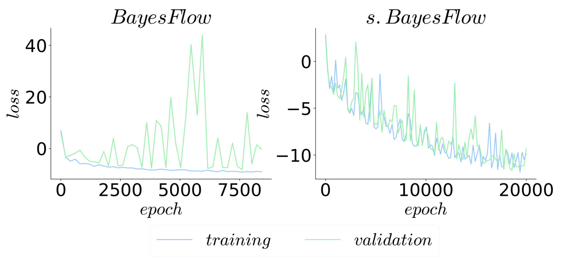


Fig. 6. Comparison of training and validation loss development between BayesFlow and stochastic BayesFlow with training set size 400.

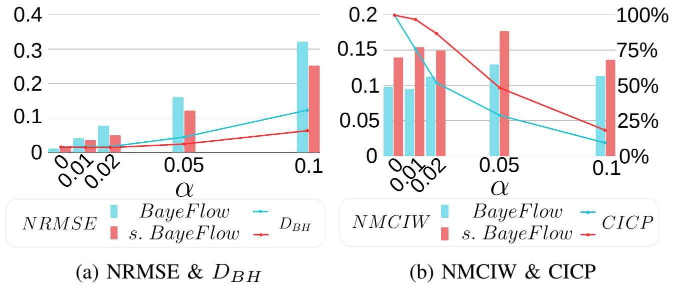
D. Influence of noise

For a better simulation of the stochastic inverse problem, we add additive white Gaussian noise to the observations as follows:

$$\mathcal{Y}_n(t) = g_n(\boldsymbol{\Theta}, t) + \epsilon_n(t), n = 1, \dots, N, \tag{14}$$

$$\epsilon_n(t) = a \frac{\max\limits_{1 \le k \le K}(\mathcal{Y}_n^{(k)}) - \min\limits_{1 \le k \le K}(\mathcal{Y}_n^{(k)})}{2} \mathcal{N}(0, 1), \tag{15}$$

where α is the noise factor multiplied with the overall amplitude. The inference results of BayesFlow and stochastic BayesFlow with different noise factors for l_f are shown in Figure 7.



(a) NRMSE & D_{BH} (b) NMCIW & CICP

Fig. 7. Comparison of inference results for l_f between BayesFlow and stochastic BayesFlow with noisy observations.

With $\alpha = 0.01$, the two models estimate parameters as precisely and reliably as they would without noisy observations. Start with $\alpha = 0.02$, the NRMSE rises and the CICP falls as the noise factor increases. Besides, stochastic BayesFlow reaches a lower NRMSE (-35 % at $\alpha = 0.02$), a higher NMCIW (+33 % at $\alpha = 0.02$), and a higher CICP (+67 % at $\alpha = 0.02$) than BayesFlow, implying that stochastic BayesFlow is more robust against noise in observations. The complete reconstructed distributions with different noise factors are shown in Figure 8.

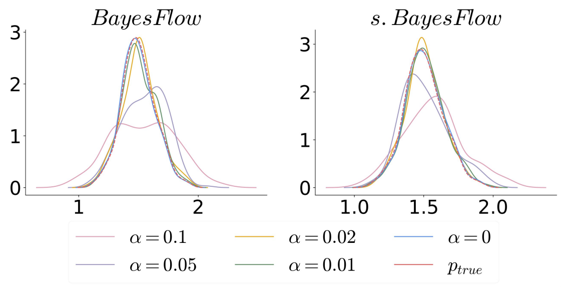


Fig. 8. Comparison of distribution reconstruction for l_f between BayesFlow and stochastic BayesFlow with noisy observations.

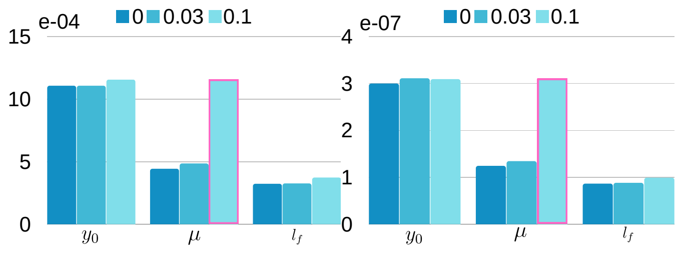
For $\alpha < 0.02$, stochastic BayesFlow reconstructs the distribution flawlessly. For $\alpha \geqslant 0.05$, although both models fail to estimate the densities exactly, stochastic BayesFlow can still rebuild a distribution that characteristically follows the ground truth.

E. Uncertainty quantification

An estimated distribution's total variance can be divided into aleatoric σ_a^2 and epistemic σ_e^2 uncertainty. The aleatoric uncertainty arises from the randomness of the latent variable $\boldsymbol{z}$ and the error in observations $\boldsymbol{\epsilon}$, whereas the epistemic uncertainty arises from the randomness in the model, i.e., the MCMC layer in the stochastic BayesFlow. The aleatoric and epistemic uncertainty for a single posterior distribution are calculated by

$$\sigma_a^2 = \mathbb{E}_{f_\phi}[\mathbb{V}_{\boldsymbol{z}}[f_\phi(h_\varphi(\boldsymbol{z}))]],\ \sigma_e^2 = \mathbb{V}_{f_\phi}[\mathbb{E}_{\boldsymbol{z}}[f_\phi(\boldsymbol{z})]]. \tag{16}$$

We train stochastic BayesFlow using observations with evenly mixed noise factors $\alpha = 0, 0.01, 0.02, 0.05$. Then, we quantify the inverse uncertainty using test parameters following different distributions but still within their value ranges, i.e., $y_0 \sim \mathcal{U}(-0.1, 3)$, $\mu \sim \mathcal{N}(0.8, 0.2^2)$, $l_f \sim \mathcal{U}(0.5, 0.15^2)$, and correspondingly clean and noisy observations with $\alpha = 0, 0.03, 0.1$. Afterwards, we execute the well-trained model 100 times in the forward direction, and at each time we sample the latent variable B times. The three groups of uncertainty decomposition are compared in Figure 9.



(a) σ_a^2 (b) σ_e^2

Fig. 9. Uncertainty decomposition of the estimated distributions of y_0, μ, and l_f identified from observations with $\alpha = 0, 0.03, 1$. As for the value of aleatoric, epistemic uncertainty, we take the mean value over the K samples.

The dominant uncertainty is aleatoric, suggesting the noise in observations is not well modeled by stochastic BayesFlow and cannot be reduced by collecting more data. Regarding the uncertainty from clean observations as the reference, when the uncertainty stays at a comparable level, like at $\alpha = 0.03$, the parameter distributions are well estimated. In contrast, if one of the uncertainties is extremely higher than the reference, such as μ at $\alpha = 0.1$, the estimated parameter is unreliable. From Figure 7a, we can infer that the prior distributions can still be estimated within the tolerance of NRMSE < 0.1 for $\alpha < 0.04$. Otherwise, the observed signals should be pre-processed by denoising methods.

The estimated distributions identified from noisy observations with $\alpha = 0.03, 0.1$ are shown in Figure 10.

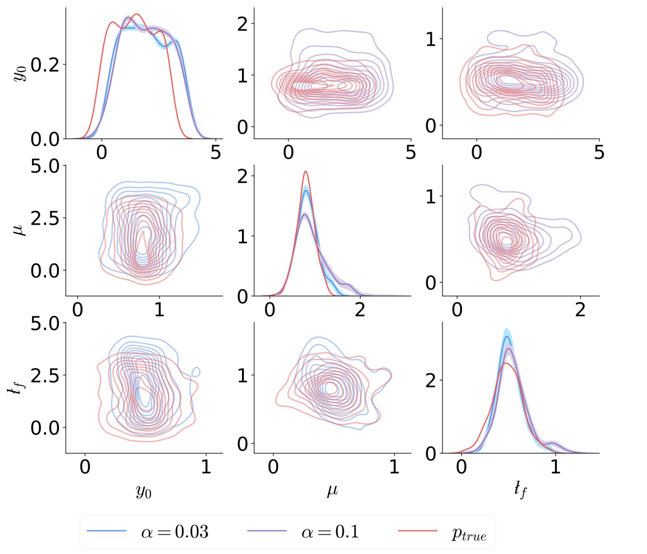


Fig. 10. Estimated distributions of y_0, μ, and l_f as well as joint distributions identified from noisy observations with $\alpha = 0.03, 0.1$. The light blue lines for $\alpha = 0.03$ and the light purple lines for $\alpha = 0.1$ demonstrate the total variances of the reconstructed prior distributions.

At $\alpha = 0.03$, the three estimated probability densities differ slightly from the reference. At $\alpha = 0.1$, the variance disagrees with the reference to some extent, although the mean of $p(\mu)$ is roughly accurately estimated. This is consistent with the uncertainty quantification. In addition, the overlapped joint probability densities of parameters prove that the relationship between parameters can also be maintained.

V. Conclusion

The aim of this study is to determine the prior probability densities of parameters from stochastic observations of physics-based models. Our proposed algorithm with stochastic BayesFlow has the following advantages: Firstly, the form of the distributions to be estimated is unlimited because stochastic BayesFlow infers indirectly the posteriors of parameters, and the prior distributions are reconstructed from the estimated posteriors. Secondly, the stochastic sampling element in BayesFlow improves its ability for generalization and robustness against noisy observations. Thirdly, the model-inherited possibility of uncertainty quantification provides an indicator of the trustworthiness of the estimated parameters and distributions.

Nevertheless, the major limitation of this algorithm is that the value ranges of parameters for test samples must be within those of the training set. The out-of-distribution problem might be detected by the model misspecification method [26], or potentially solved by test-time training [27] in future work. In addition, another realization of stochastic BayesFlow can be attempted by implementing a stochastic layer after each deterministic layer. Then, a forward mapping from the parameter to the summary network output is required for calculating the target density of stochastic layers. A possible forward mapping is the physics-enhanced latent space variational autoencoder (PELS-VAE), which is explained in [28].

Acknowledgment

We would like to genuinely thank Dr. Stefan T. Radev (University of Heidelberg) for generously sharing his experience and expertise with us. This research was supported by the PHyMoS Project (Proper Hybrid Models for Smarter Vehicles), which is funded by the German Federal Ministry for Economic Affairs and Climate Action (BMWK) under grant number 19I20022H.

References

[1] S. T. Radev, U. K. Mertens, A. Voss, L. Ardizzone, and U. Kothe, “BayesFlow: Learning Complex Stochastic Models With Invertible Neural Networks,” IEEE Transactions on Neural Networks and Learning Systems, vol. 33, no. 4, pp. 1452–1466, Apr. 2022.

[2] C. Winkler, D. Worrall, E. Hoogeboom, and M. Welling, ”Learning likelihoods with conditional normalizing flows,” arXiv [cs.LG], Preprint, 2019.

[3] J. Breidt, T. Butler, and D. Estep, “A Measure-Theoretic Computational Method for Inverse Sensitivity Problems I: Method and Analysis,” SIAM Journal on Numerical Analysis, vol. 49, no. 5, pp. 1836–1859, Jan. 2011.

[4] T. Butler, D. Estep, and J. Sandelin, “A Computational Measure Theoretic Approach to Inverse Sensitivity Problems II: A Posteriori Error Analysis,” SIAM Journal on Numerical Analysis, vol. 50, no. 1, pp. 22–45, Jan. 2012.

[5] T. Butler, D. Estep, S. Tavener, C. Dawson, and J. J. Westerink, “A Measure-Theoretic Computational Method for Inverse Sensitivity Problems III: Multiple Quantities of Interest,” SIAM/ASA Journal on Uncertainty Quantification, vol. 2, no. 1, pp. 174–202, Jan. 2014.

[6] J. Parikh, J. Kozloski, and V. Gurev, ”Integration of AI and mechanistic modeling in generative adversarial networks for stochastic inverse problems,” arXiv [stat.ML], 2020.

[7] Heringhaus, Monika E., Yi Zhang, André Zimmermann, and Lars Mikelsons, ”Towards Reliable Parameter Extraction in MEMS Final Module Testing Using Bayesian Inference,” Sensors 22, no. 14: 5408, 2022.

[8] Wu, Hao, Jonas Köhler, and Frank Noé. ”Stochastic normalizing flows,” Advances in Neural Information Processing Systems 33: 5933-5944, 2020.

[9] P. Hagemann, J. Hertrich, and G. Steidl, “Stochastic Normalizing Flows for Inverse Problems: A Markov Chains Viewpoint,” SIAM/ASA Journal on Uncertainty Quantification, vol. 10, no. 3, pp. 1162–1190, Sep. 2022.

[10] Marcy, Peter W., and Rebecca E. Morrison, ”” Stochastic Inverse Problems” and Changes-of-Variables,” arXiv [stat.ME], Preprint, 2022.

[11] G. Lai, W.-C. Chang, Y. Yang, and H. Liu, ”Modeling long-and short-term temporal patterns with deep neural networks,” in The 41st international ACM SIGIR conference on research & development in information retrieval, pp. 95–104, 2018.

[12] Ardizzone, L., Lüth, C., Kruse, J., Rother, C. and Köthe, U., ”Guided image generation with conditional invertible neural networks,” arXiv [cs.CV], Preprint, 2019.

[13] I. Kobyzev, S. J. D. Prince, and M. A. Brubaker, “Normalizing Flows: An Introduction and Review of Current Methods,” IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 1–1, 2020.

[14] G. Papamakarios, E. T. Nalisnick, D. J. Rezende, S. Mohamed, and B. Lakshminarayanan, ”Normalizing Flows for Probabilistic Modeling and Inference,” J. Mach. Learn. Res., vol. 22, no. 57, pp. 1–64, 2021.

[15] D. Rezende and S. Mohamed, ”Variational inference with normalizing flows,” in International conference on machine learning, pp. 1530–1538, 2015.

[16] Ardizzone, Lynton, et al., ”Analyzing inverse problems with invertible neural networks,” arXiv [cs.LG], Preprint, 2018.

[17] J. R. Hershey and P. A. Olsen, ”Approximating the Kullback Leibler divergence between Gaussian mixture models,” in 2007 IEEE International Conference on Acoustics, Speech and Signal Processing-ICASSP’07, vol. 4, p. IV–317, 2007.

[18] S. Hochreiter and J. Schmidhuber, ”Long short-term memory,” Neural computation, vol. 9, no. 8, pp. 1735–1780, 1997.

[19] K. Cho et al., ”Learning phrase representations using RNN encoder-decoder for statistical machine translation,” arXiv [cs.CL], Preprint, 2014.

[20] S. Ioffe and C. Szegedy, ”Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in International conference on machine learning, pp. 448–456, 2015.

[21] G. O. Roberts and J. S. Rosenthal, “General state space Markov chains and MCMC algorithms,” Probability Surveys, vol. 1, no. none, Jan. 2004.

[22] “tum-cps / commonroad-vehicle-models · GitLab,” GitLab. https://gitlab.lrz.de/tum-cps/commonroad-vehicle-models (accessed Jan. 25, 2023).

[23] L. V. Jospin, H. Laga, F. Boussaid, W. Buntine, and M. Bennamoun, “Hands-On Bayesian Neural Networks—A Tutorial for Deep Learning Users,” IEEE Computational Intelligence Magazine, vol. 17, no. 2, pp. 29–48, May 2022.

[24] S. Bi, M. Broggi, and M. Beer, “The role of the Bhattacharyya distance in stochastic model updating,” Mechanical Systems and Signal Processing, vol. 117, pp. 437–452, Feb. 2019.

[25] S. Seabold and J. Perktold, ”statsmodels: Econometric and statistical modeling with python,” in 9th Python in Science Conference, 2010.

[26] M. Schmitt, P.-C. Bürkner, U. Köthe, and S. T. Radev, ”Detecting Model Misspecification in Amortized Bayesian Inference with Neural Networks,” arXiv [stat.ME], Preprint, 2021.

[27] Osowiechi, D., Hakim, G. A. V., Noori, M., Cheraghalikhani, M., Ben Ayed, I., and Desrosiers, C., ”TTTFlow: Unsupervised Test-Time Training with Normalizing Flow,” In Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, pp. 2126-2134, 2023.

[28] Y. Zhang, L. Mikelsons, ”Sensitivity-Guided Iterative Parameter Identification and Data Generation with BayesFlow and PELS-VAE for Model Calibration,” Research Square, Preprint, 2022.