JANA: Jointly Amortized Neural Approximation of Complex Bayesian Models (Supplementary Material)

Stefan T. Radev[1] **Marvin Schmitt**[2] **Valentin Pratz**[3] **Umberto Picchini**[4]
Ullrich Köthe[*3] **Paul-Christian Bürkner**[*2]

[1]Cluster of Excellence STRUCTURES, Heidelberg University
[2]Cluster of Excellence SimTech, University of Stuttgart
[3]Visual Learning Lab, Heidelberg University
[4]Department of Mathematical Sciences, Chalmers University of Technology & University of Gothenburg

A FREQUENTLY ASKED QUESTIONS (FAQ)

Q: How can I reproduce the results?

Code to reproduce all results is available in the repository at https://github.com/bayesflow-org/JANA-Paper.

Q: How can I apply JANA to my own Bayesian models?

Simulation-based algorithms for jointly amortized inference are implemented in the `BayesFlow` library. Take a look at the code and tutorials, available at: https://github.com/stefanradev93/BayesFlow.

Q: When should I use amortized inference instead of sequential methods?

Whenever you want to follow a principled Bayesian workflow and you have lots of data sets on which a Bayesian model needs to be applied independently.

Q: Does amortization come at the cost of wasteful simulations?

Some previous papers assume that this is generally the case. On the contrary, we believe that wasteful simulations are primarily the consequence of poorly chosen priors, whereas modern neural networks actually profit from broader simulation scopes, as long as the priors are informative. Moreover, amortization makes a principled Bayesian workflow much easier than case-based inference. Still, specifying sensible joint priors is not always easy.

Q: Can you somehow combine the three networks and utilize weight sharing?

Finding a suitable weight sharing approach which is applicable to various model structures—such as exchangeable or Markovian—proves challenging. Since JANA is an attempt at a universal method, we refrain from customizing the overall architecture to suit a particular model structure. Instead of weight sharing, we exploit the probabilistic symmetries of joint Bayesian learning, which is universal across all model structures (see Figure 1 of the main paper). Although it remains a possible area for further investigation, we are uncertain whether weight sharing in our context is even desirable.

Q: Can I use a different type of generative network for the posterior or likelihood networks?

JANA can operate with arbitrary conditional density approximators. However, it is important that these approximators are able to efficiently compute *normalized densities* for the purpose of marginal likelihood and posterior predictive estimation.

Q: Why do you need a summary network?

Because most real world data comes in various sizes and shapes. Thus, we need an interface between the Bayesian model and the posterior network which renders the latter applicable to various sizes and shapes.

Q: Can you also use a summary network for the likelihood network?

It is possible and can be helpful if the parameter space of the reference Bayesian model requires some form of compression.

Indeed, in the second iteration of the paper, we included a Bayesian denoising experiment (**Experiment 5**) which equips the surrogate likelihood with a convolutional summary network.

Q: Is it necessary to have normalized likelihood estimates or would a standard feedforward neural network suffice?

A normalized likelihood is necessary to estimate the expected log predictive density (ELPD) for approximating out-of-sample predictive performance via cross validation or log marginal likelihoods (LMLs) for approximating Bayes factors. Normalization is also needed to compare likelihoods obtained from different models (which might otherwise report unnormalized likelihoods at different scales). If none of these (log) likelihood metrics is needed for a particular analysis, normalization of the likelihood network is not strictly required.

B CODE

The code and instructions for running and reproducing all experiments are available at the project's repository https://github.com/bayesflow-org/JANA-Paper. We use fixed seeds for the random number generators of test (held-out) sets. Training uses no seeds, as we believe the methods to be stable enough to converge on any run.

C METHOD DETAILS

C.1 PSEUDOCODE

Algorithm 1 Jointly amortized neural approximation: offline training using a pre-simulated training set

```
Input: Bayesian model $p(\boldsymbol{\theta}, \boldsymbol{x})$; summary network $\mathcal{H}_{\boldsymbol{\psi}}$; posterior network $\mathcal{P}_{\boldsymbol{\phi}}$; likelihood network $\mathcal{L}_{\boldsymbol{\eta}}$; number of simulations $N$ (budget); batch size $B$
 1: Initialize $\mathcal{D} = \{\}$.
 2: for $n = 1, \ldots, N$ do
 3:     Sample from prior: $\boldsymbol{\theta}_n \sim p(\boldsymbol{\theta})$
 4:     Sample from (implicit) likelihood: $\boldsymbol{x}_n \sim p(\boldsymbol{x} \mid \boldsymbol{\theta}_n)$
 5:     Add simulations to training data: $\mathcal{D} := \mathcal{D} \cup \{(\boldsymbol{\theta}_n, \boldsymbol{x}_n)\}$
 6: end for
 7: while not converged do
 8:     Sample batch from training data: $\{(\boldsymbol{\theta}_b, \boldsymbol{x}_b)\}_{b=1}^{B} \sim \mathcal{D}$
 9:     Compute Monte Carlo estimate of loss function over batch (Eq. 12).
10:     Update neural network parameters $(\boldsymbol{\psi}, \boldsymbol{\phi}, \boldsymbol{\eta})$ via backpropagation.
11: end while
12: return trained networks $l_{\boldsymbol{\eta}}(\boldsymbol{x} \mid \boldsymbol{\theta}), p_{\boldsymbol{\phi}}(\boldsymbol{\theta} \mid \mathcal{H}_{\boldsymbol{\psi}}(\boldsymbol{x})), \mathcal{H}_{\boldsymbol{\psi}}$
```

C.2 LIKELIHOOD NETWORKS FOR EXCHANGEABLE DATA

Exchangeable models generate IID data, that is, each run $G(\boldsymbol{\theta}, \boldsymbol{\xi})$ with a fixed configuration $\boldsymbol{\theta}$ is independent of all other runs. Thus, for N runs of such a (memoryless or stateless) model, the likelihood decomposes into the product of point-wise likelihoods:

$$p(\boldsymbol{x} \mid \boldsymbol{\theta}) = \prod_{n=1}^{N} p(\boldsymbol{x}_n \mid \boldsymbol{\theta}) \tag{1}$$

Accordingly, we can represent such data as unordered sets and simply apply the likelihood network exchangeably by concatenating each $\boldsymbol{x}_n$ with $\boldsymbol{\theta}$ in each coupling layer. The forward pass for a single conditional affine coupling layer (Ardizzone et al., 2019; Radev et al., 2020) of an exchangeable likelihood network is given by:

$$\begin{aligned}
\boldsymbol{z}_n^{\mathcal{A}} &= \boldsymbol{x}_n^{\mathcal{A}} \odot \exp(S_1(\boldsymbol{x}_n^{\mathcal{B}}; \boldsymbol{\theta})) + T_1(\boldsymbol{x}_n^{\mathcal{B}}; \boldsymbol{\theta}) \\
\boldsymbol{z}_n^{\mathcal{B}} &= \boldsymbol{x}_n^{\mathcal{B}} \odot \exp(S_2(\boldsymbol{z}_n^{\mathcal{A}}; \boldsymbol{\theta})) + T_2(\boldsymbol{z}_n^{\mathcal{A}}; \boldsymbol{\theta}),
\end{aligned}$$

where $\boldsymbol{x}_n = (\boldsymbol{x}_n^{\mathcal{A}}, \boldsymbol{x}_n^{\mathcal{B}})$ is a disjoint partition of the input data at position n, $\boldsymbol{z}_n = (\boldsymbol{z}_n^{\mathcal{A}}, \boldsymbol{z}_n^{\mathcal{B}})$ is the corresponding latent partition, and the functions S_1, S_2, T_1, T_2 are implemented as multi-headed fully connected (FC) neural networks (with trainable parameters suppressed for clarity). The forward pass for neural spline flows (Durkan et al., 2019) is modified accordingly, such that the spline parameters are generated exchangeably, conditioned on the parameter vector $\boldsymbol{\theta}$.

C.3 LIKELIHOOD NETWORKS FOR MARKOVIAN DATA

The widely used family of Markovian models factorize in a way that the probability of each data point depends on previous data points:

$$p(\boldsymbol{x} \mid \boldsymbol{\theta}) = \prod_{n=1}^{N} p(\boldsymbol{x}_n \mid \boldsymbol{\theta}, \boldsymbol{x}_{1:n-1}) \tag{2}$$

Such models require a slightly different coupling layer design which respects their non-IID outputs. To this end, we augment standard coupling layers with a conditional recurrent (GRU) memory $\boldsymbol{h}_n = M(\boldsymbol{\theta}, \boldsymbol{x}_n; \boldsymbol{h}_{n-1})$ which encodes temporal dependencies into a hidden state vector $\boldsymbol{h}$.

For instance, the forward pass for a single conditional affine coupling layer of the non-exchangeable likelihood network is then given by:

$$\begin{aligned}
\boldsymbol{h}_n &= M(\boldsymbol{\theta}, \boldsymbol{x}_n; \boldsymbol{h}_{n-1}) \\
\boldsymbol{z}_n^{\mathcal{A}} &= \boldsymbol{x}_n^{\mathcal{A}} \odot \exp(S_1(\boldsymbol{x}_n^{\mathcal{B}}; \boldsymbol{\theta}, \boldsymbol{h}_{n-1})) + T_1(\boldsymbol{x}_n^{\mathcal{B}}; \boldsymbol{\theta}, \boldsymbol{h}_{n-1}) \\
\boldsymbol{z}_n^{\mathcal{B}} &= \boldsymbol{x}_n^{\mathcal{B}} \odot \exp(S_2(\boldsymbol{z}_n^{\mathcal{A}}; \boldsymbol{\theta}, \boldsymbol{h}_{n-1})) + T_2(\boldsymbol{z}_n^{\mathcal{A}}; \boldsymbol{\theta}, \boldsymbol{h}_{n-1}),
\end{aligned}$$

where now each latent representation $\boldsymbol{z}_n$ at position n depends on the preceding data points, as encoded by $\boldsymbol{h}_{n-1}$, and the functions S_1, S_2, T_1, T_2 are implemented as multi-headed fully connected (FC) neural networks. The forward pass for neural spline flows (Durkan et al., 2019) is modified accordingly, such that the spline parameters are generated using a recurrent memory, conditioned on the parameter vector $\boldsymbol{\theta}$.

C.4 CORRECTNESS OF JOINT SIMULATION-BASED TRAINING

To show that our jointly optimized criterion yields correct posterior, likelihood, and marginal likelihood inference, consider first the joint optimization of the posterior and the summary network:

$$\begin{aligned}
(\boldsymbol{\phi}^*, \boldsymbol{\psi}^*) &= \operatorname*{arg\,min}_{\boldsymbol{\phi}, \boldsymbol{\psi}} \mathbb{E}_{p^*(\boldsymbol{x})} \left[\mathbb{KL}(p(\boldsymbol{\theta} \,|\, \boldsymbol{x}) \,||\, p_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \mathcal{H}_{\boldsymbol{\psi}}(\boldsymbol{x})))\right] && (3) \\
&= \operatorname*{arg\,min}_{\boldsymbol{\phi}, \boldsymbol{\psi}} \mathbb{E}_{p^*(\boldsymbol{x})} \left[\mathbb{E}_{p(\boldsymbol{\theta} \,|\, \boldsymbol{x})} \left[\log p(\boldsymbol{\theta} \,|\, \boldsymbol{x}) - \log p_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \mathcal{H}_{\boldsymbol{\psi}}(\boldsymbol{x}))\right]\right] && (4) \\
&= \operatorname*{arg\,min}_{\boldsymbol{\phi}, \boldsymbol{\psi}} \mathbb{E}_{p^*(\boldsymbol{x})} \left[\mathbb{E}_{p(\boldsymbol{\theta} \,|\, \boldsymbol{x})} \left[- \log p_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \mathcal{H}_{\boldsymbol{\psi}}(\boldsymbol{x}))\right]\right] && (5)
\end{aligned}$$

The above criterion (Eq. 3) states that, in order to achieve proper amortized inference, we want to minimize the Kullback-Leibler (KL) divergence between the analytic and the approximate posterior density in expectation over all possible observations from the true data-generating distribution p^*. This reduces to the expected negative log posterior (Eq. 5), since the negative entropy of the analytic posterior $-\mathbb{H}\left[p(\boldsymbol{\theta} \,|\, \boldsymbol{x})\right] = \mathbb{E}_{p(\boldsymbol{\theta} \,|\, \boldsymbol{x})} \left[\log p(\boldsymbol{\theta} \,|\, \boldsymbol{x})\right]$ does not depend on the neural network parameters $(\boldsymbol{\phi}, \boldsymbol{\psi})$.

In order to make amortized posterior inference tractable under Eq. 3, we need to assume that the true data-generating distribution p^* and the model-implied (i.e., prior predictive) distribution $p(\boldsymbol{x}) = \mathbb{E}_{p(\boldsymbol{\theta})} \left[p(\boldsymbol{x} \,|\, \boldsymbol{\theta})\right]$ match, that is, $p^*(\boldsymbol{x}) = p(\boldsymbol{x})$ for any $\boldsymbol{x}$. In other words, we invoke the so-called *closed-world assumption*, which states the Bayesian model is a correct representation of the true data-generating distribution. In that case, we can simply replace $p^*(\boldsymbol{x})$ with $p(\boldsymbol{x})$ and write our criterion as:

$$\begin{aligned}
(\boldsymbol{\phi}^*, \boldsymbol{\psi}^*) &= \operatorname*{arg\,min}_{\boldsymbol{\phi}, \boldsymbol{\psi}} \mathbb{E}_{p(\boldsymbol{x})} \left[\mathbb{E}_{p(\boldsymbol{\theta} \,|\, \boldsymbol{x})} \left[- \log p_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \mathcal{H}_{\boldsymbol{\psi}}(\boldsymbol{x}))\right]\right] && (6) \\
&= \operatorname*{arg\,min}_{\boldsymbol{\phi}, \boldsymbol{\psi}} \mathbb{E}_{p(\boldsymbol{\theta}, \boldsymbol{x})} \left[- \log p_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \mathcal{H}_{\boldsymbol{\psi}}(\boldsymbol{x}))\right] && (7)
\end{aligned}$$

We can now readily approximate the expectation with its empirical mean over a data set of simulations $(\boldsymbol{\theta}, \boldsymbol{x}) \sim \mathcal{D}$ generated from the Bayesian joint model $p(\boldsymbol{\theta}, \boldsymbol{x})$. Thereby, we leverage the fact that we can directly evaluate $- \log p_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \mathcal{H}_{\boldsymbol{\psi}}(\boldsymbol{x}))$ (and not a lower bound) due to the use of a normalizing flow (NF) for the approximate posterior. Moreover, as shown by Radev et al. (2020), perfect convergence under Eq. 7 ensures that the summary network learns maximally informative (ideally sufficient) summary statistics and the posterior network samples from the analytic posterior. Note, however, that if the key assumption of $p^*(\boldsymbol{x}) = p(\boldsymbol{x})$ is violated for some $\boldsymbol{x}$, then the approximate posterior may no longer be a faithful representation of the analytic posterior in general. This situation motives the introduction of the summary space distribution $p(\mathcal{H}_{\boldsymbol{\psi}}(\boldsymbol{x}))$ (to be discussed shortly).

As for the likelihood network, we aim to minimize the KL divergence between the analytic and the approximate posterior density in expectation over all possible parameter configurations from the prior:

$$\boldsymbol{\eta}^* = \operatorname*{arg\,min}_{\boldsymbol{\eta}} \mathbb{E}_{p(\boldsymbol{\theta})} \left[\mathbb{KL}(p(\boldsymbol{x} \,|\, \boldsymbol{\theta}) \,||\, l_{\boldsymbol{\eta}}(\boldsymbol{x} \,|\, \boldsymbol{\theta}))\right] \tag{8}$$

Following the same reasoning as for the posterior KL and leveraging the fact that the expectation runs over a model-implied quantity (i.e., the prior), the above criterion directly reduces to:

$$\boldsymbol{\eta}^* = \operatorname*{arg\,min}_{\boldsymbol{\eta}} \mathbb{E}_{p(\boldsymbol{\theta}, \boldsymbol{x})} \left[- \log l_{\boldsymbol{\eta}}(\boldsymbol{x} \,|\, \boldsymbol{\theta})\right] \tag{9}$$

Observing that both optimization criteria (Eq. 7 and Eq. 9) include an expectation over the Bayesian joint $p(\boldsymbol{\theta}, \boldsymbol{x})$, we arrive at our combined loss function:

$$\mathcal{L}_{\text{JANA}} := -\mathbb{E}_{p(\boldsymbol{\theta},\boldsymbol{x})}\big[\log l_{\boldsymbol{\eta}}(\boldsymbol{x} \,|\, \boldsymbol{\theta}) + \log p_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \mathcal{H}_{\boldsymbol{\psi}}(\boldsymbol{x}))\big] \tag{10}$$

Thus, under the closed-world assumption, proper minimization of this loss ensures correct posterior and likelihood approximation. However, in practice, we want to obtain some measure of the mismatch between $p^*(\boldsymbol{x})$ and $p(\boldsymbol{x})$. Moreover, since $\boldsymbol{x}$ is typically a high dimensional data set (e.g., a data set of multivariate IID observations) and the posterior network only "sees" $\boldsymbol{x}$ through the lens of the summary network, it makes sense to measure the potential mismatch in the reduced summary space given by $\mathcal{H}_{\boldsymbol{\psi}}(\boldsymbol{x})$. To make the detection task even easier, we want to re-structure the unrestricted $p(\mathcal{H}_{\boldsymbol{\psi}}(\boldsymbol{x}))$ into a simple distribution (e.g., Gaussian) with a well-defined notion of an outlier. Accordingly, we utilize the Maximum Mean Discrepancy (MMD; Gretton et al., 2012):

$$\mathbb{MMD}^2\big[p^*(\boldsymbol{x}) \,||\, p(\boldsymbol{x})\big] = \mathbb{E}_{p^*(\boldsymbol{x})}\big[\kappa(\boldsymbol{x}, \boldsymbol{x}')\big] + \mathbb{E}_{p(\boldsymbol{x})}\big[\kappa(\boldsymbol{x}, \boldsymbol{x}')\big] - 2\mathbb{E}_{\boldsymbol{x}\sim p^*(\boldsymbol{x}),\boldsymbol{x}'\sim p(\boldsymbol{x})}\big[\kappa(\boldsymbol{x}, \boldsymbol{x}')\big], \tag{11}$$

where $\kappa(\cdot, \cdot)$ is any reproducing kernel and we simply replace $\boldsymbol{x}$ with $\mathcal{H}_{\boldsymbol{\eta}}(\boldsymbol{x})$. The MMD is a suitable alternative to the KL whenever we want to measure the distance between two distributions from which we can obtain samples but cannot evaluate explicitly. Our augmented loss function then becomes:

$$\mathcal{L}_{\text{JANA-MMD}} := -\mathbb{E}_{p(\boldsymbol{\theta},\boldsymbol{x})}\big[\log l_{\boldsymbol{\eta}}(\boldsymbol{x} \,|\, \boldsymbol{\theta}) + \log p_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \mathcal{H}_{\boldsymbol{\psi}}(\boldsymbol{x}))\big] + \lambda \cdot \mathbb{MMD}^2\big[p(\mathcal{H}_{\boldsymbol{\psi}}(\boldsymbol{x})) \,||\, \mathcal{N}(\mathbf{0}, \mathbb{I})\big], \tag{12}$$

where $\mathcal{N}(\mathbf{0}, \mathbb{I})$ denotes a spherical multivariate Gaussian distribution. Note, that, in theory, proper minimization of the MMD term does not trade off the performance of the posterior network, but simply implies a reparameterization $\boldsymbol{\phi} \to \boldsymbol{\phi}', \boldsymbol{\psi} \to \boldsymbol{\psi}'$, such that:

$$p(\boldsymbol{\theta}) = \int p_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \mathcal{H}_{\boldsymbol{\psi}}(\boldsymbol{x}))\, p(\boldsymbol{x})\, \mathrm{d}\boldsymbol{x} = \int p_{\boldsymbol{\phi}'}(\boldsymbol{\theta} \,|\, \mathcal{H}_{\boldsymbol{\psi}'}(\boldsymbol{x}))\, \mathcal{N}(\mathcal{H}_{\boldsymbol{\psi}'}(\boldsymbol{x}) \mid \mathbf{0}, \mathbb{I})\, \mathrm{d}\boldsymbol{x} \tag{13}$$

In a particular empirical setting, neural network parameters $(\boldsymbol{\phi}, \boldsymbol{\psi})$ may be more easily reachable by a given optimizer than corresponding parameters $(\boldsymbol{\phi}', \boldsymbol{\psi}')$, resulting in a practical trade-off. However, Schmitt et al. (2021) did not observe a diminished performance of amortized posterior approximators trained with a structured summary space, warranting promising results and further investigation into latent summary spaces.

Finally, the correctness of the posterior and likelihood networks trivially implies a correct marginal likelihood (i.e., model evidence) due to the probabilistic change-of-variable resulting from Bayes' rule:

$$p(\boldsymbol{\theta} \,|\, \boldsymbol{x}) = \frac{p(\boldsymbol{x} \,|\, \boldsymbol{\theta})\, p(\boldsymbol{\theta})}{p(\boldsymbol{x})} \iff p(\boldsymbol{x}) = p(\boldsymbol{x} \,|\, \boldsymbol{\theta})\, \frac{p(\boldsymbol{\theta})}{p(\boldsymbol{\theta} \,|\, \boldsymbol{x})} \tag{14}$$

Thus, assuming perfect convergence of the posterior and the likelihood network under either $\mathcal{L}_{\text{JANA}}$ (Eq. 10) or $\mathcal{L}_{\text{JANA-MMD}}$ (Eq. 12), we can compute the log marginal likelihood (LML) of $\boldsymbol{x}$ by using any single $\boldsymbol{\theta} \sim p(\boldsymbol{\theta})$ as:

$$\log p(\boldsymbol{x}) = \log l_{\boldsymbol{\eta}}(\boldsymbol{x} \,|\, \boldsymbol{\theta}) + \log p(\boldsymbol{\theta}) - \log p_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \mathcal{H}_{\boldsymbol{\psi}}(\boldsymbol{x})) \tag{15}$$

Moreover, it follows, that we can use any violation of Eq. 15 to diagnose non-convergence and measure the joint approximation error incurred by the networks.

D IMPLEMENTATION DETAILS AND ADDITIONAL RESULTS

All experiments are implemented using the BayesFlow library https://github.com/stefanradev93/BayesFlow built on top of TensorFlow (Abadi et al., 2016). Throughout, we use an Adam optimizer (Kingma & Ba, 2014) with an initial learning rate between 0.0005 and 0.001, default hyperparameters, and a cosine learning rate decay schedule. All networks are trained on a single machine equipped with an NVIDIA® T4 graphics accelerator with 16GB of GPU memory.

D.1 EXPERIMENT 1: TEN BENCHMARKS

We follow the model specifications from Lueckmann et al. (2021). Model implementations are directly imported from the BayesFlow library under MIT license because this implementation has no dependencies on a particular deep learning framework. For inspecting the software code for the benchmark implementations, we kindly refer the reader to the BayesFlow repository https://github.com/stefanradev93/BayesFlow/tree/master/bayesflow/benchmarks. Table 1 contains an overview of the benchmarks and core network settings. The full network configurations can be inspected in the code section of the **Appendix**.

Table 1: Overview of the model and training configurations for **Experiment 1**.

#	Benchmark name	# Dimensions[1]	Epochs	Batch size	LR	# Coupling[2]	Results
1	Gaussian Linear	(10, 10)	50	64	0.001	(5, 5)	Figure 1
2	Gaussian Linear Uniform	(10, 10)	50	64	0.001	(5, 5)	Figure 2
3	SLCP[3]	(8, 5)	100	32	0.0005	(4, 6)	Figure 3
4	SLCP[3] with Distractors	(100, 5)	60	32	0.001	(6, 8)	Figure 4
5	Bernoulli GLM	(10, 10)	50	32	0.0001	(5, 8)	Figure 5
6	Bernoulli GLM Raw	(100, 10)	50	32	0.0001	(8, 8)	Figure 6
7	Gaussian Mixture	(2, 2)	150	64	0.0005	(6, 6)	Figure 7
8	Two Moons	(2, 2)	50	32	0.0005	(6,6)	Figure 8
9	SIR	(10, 2)	250	32	0.0001	(6,6)	Figure 9
10	Lotka-Volterra	(20, 4)	150	128	0.001	(8,6)	Figure 10

[1] Dimensionality of the Bayesian model, denoted as a tuple for $\boldsymbol{x}$ and $\boldsymbol{\theta}$, respectively.

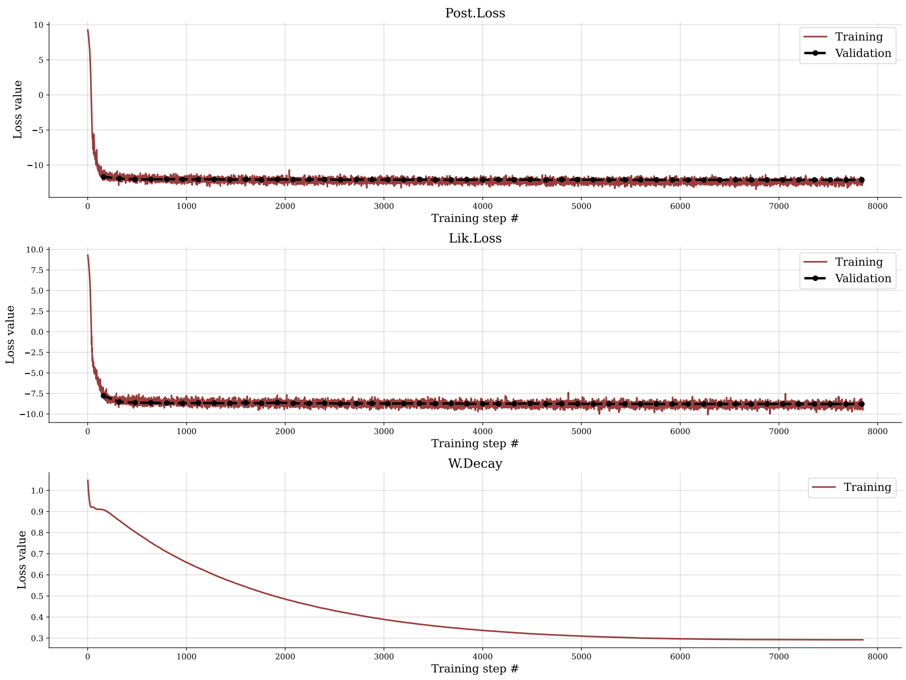
[1] Number of coupling layers, denoted as a tuple for the likelihood and posterior network, respectively.

[2] Simple Likelihood, Complex Posterior.

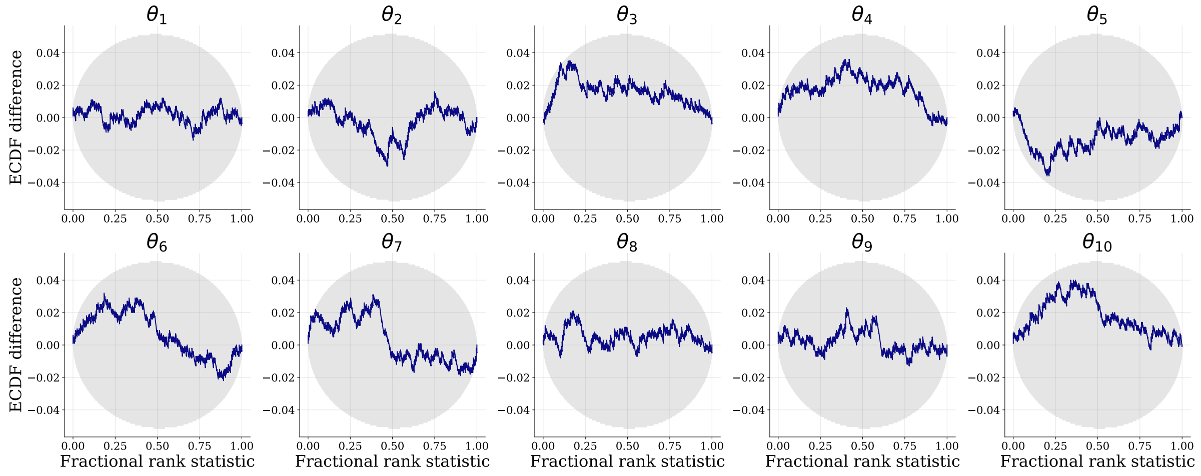
The following figures show the loss history (training and validation) as well as detailed calibration diagnostics for the posterior and joint learning tasks. Note that the simulation budget is fixed at $10\,000$ simulations. However, depending on the benchmark, the *number of training steps* may vary (i.e., Gaussian Linear is trivial to learn and requires a few epochs, in contrast to a more challenging benchmark, such as Lotka-Volterra).

Further, note that most of these models are *not* meaningful for joint or likelihood estimation in their original formulation. Still, we apply JANA to all benchmarks for the sake of completeness, as these experiments serve as a proof-of-concept for more advanced applications.

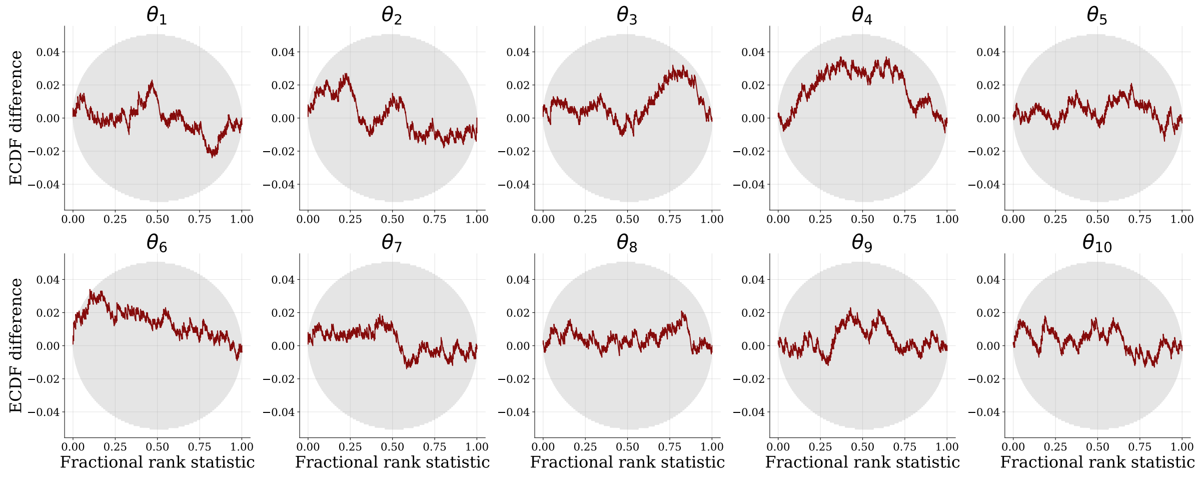
Special care is needed for the Bernoulli GLM Raw model, as its likelihood yields N IID binary data points. These should neither be directly modeled as N-dimensional vectors (as this completely ignores the permutation-invariance of the data), nor as exchangeable inputs for coupling-based invertible networks (as the latter assumes at least two-dimensional continuous outputs). In order to tackle the likelihood of this model, we augment each binary data point x_n with an independent random variate $u_n \sim \mathcal{N}(0, 1)$ and use a SoftFlow architecture (Kim et al., 2020) for dequantization of the binary data.



(a) Training and validation loss history.

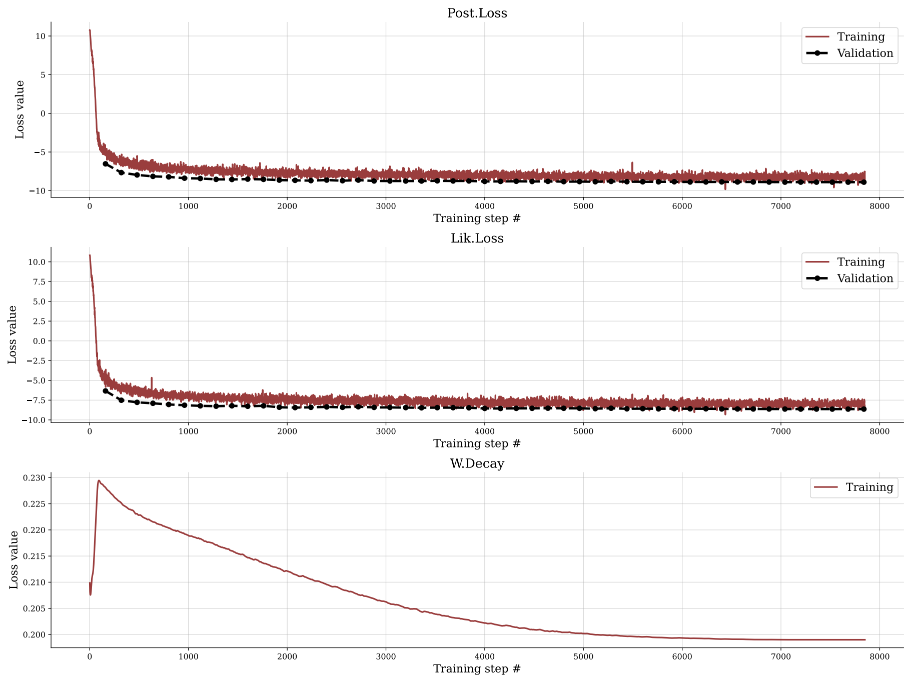


(b) Posterior calibration.

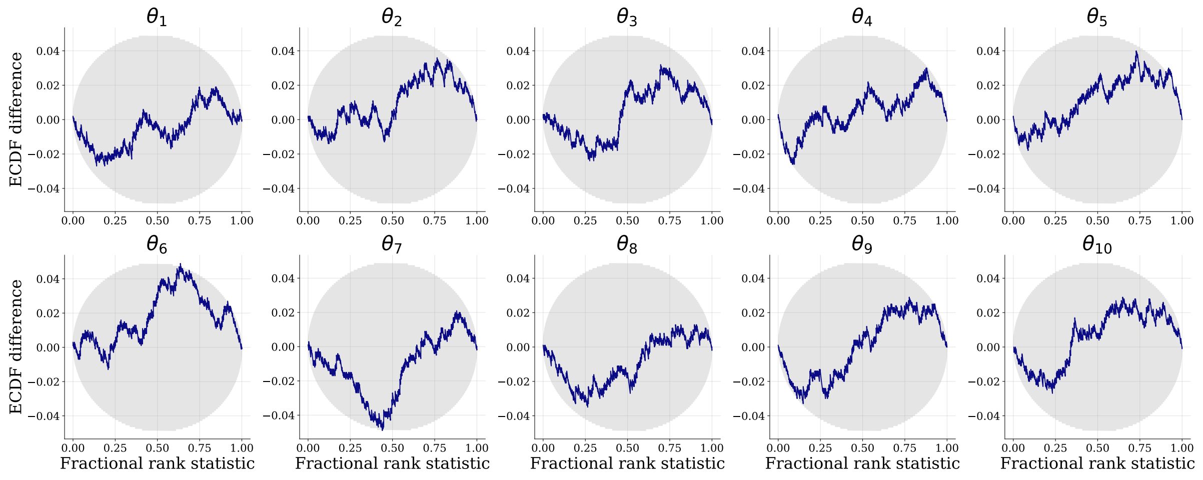


(c) Joint calibration.

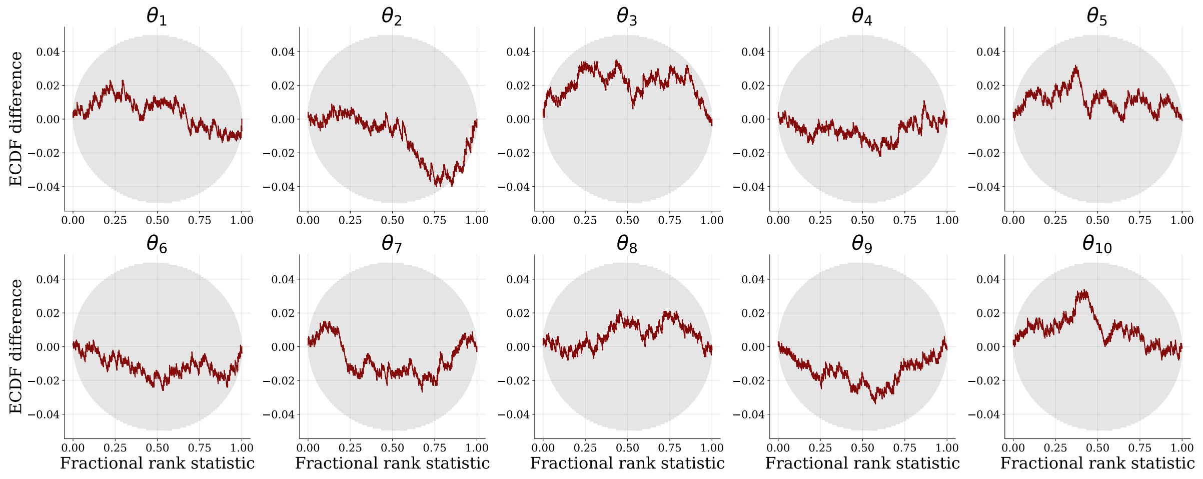
Figure 1: **Benchmark 1, Gaussian Linear.** Loss history, posterior calibration, and joint calibration.



(a) Training and validation loss history.

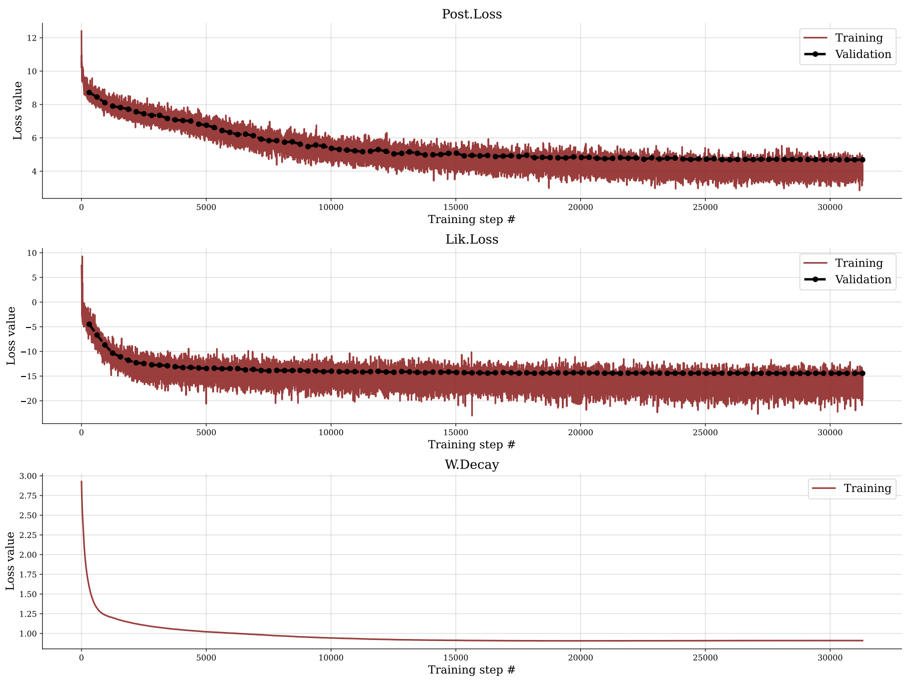


(b) Posterior calibration.

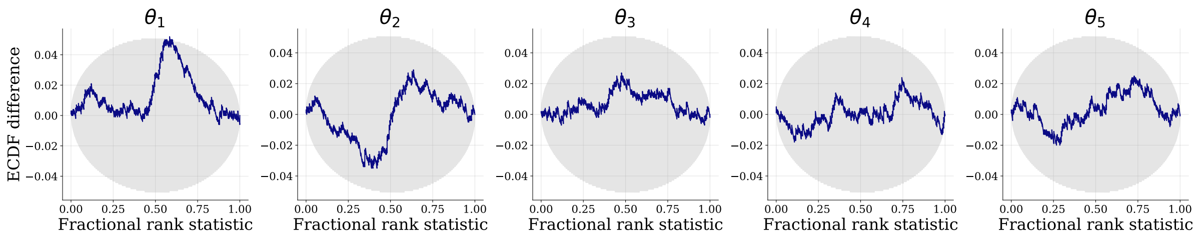


(c) Joint calibration.

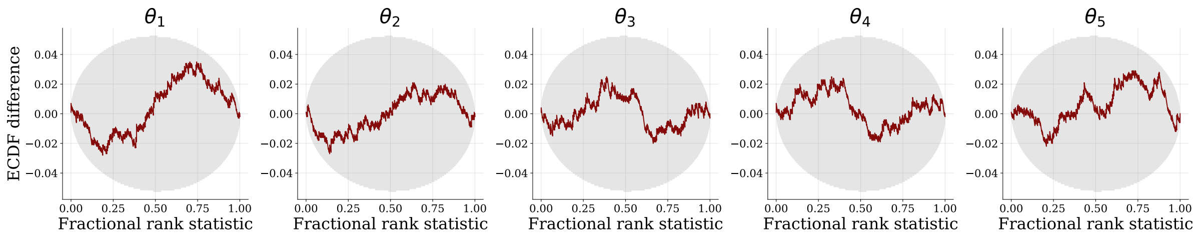
Figure 2: **Benchmark 2, Gaussian Linear Uniform.** Loss history, posterior calibration, and joint calibration.



(a) Training and validation loss history.

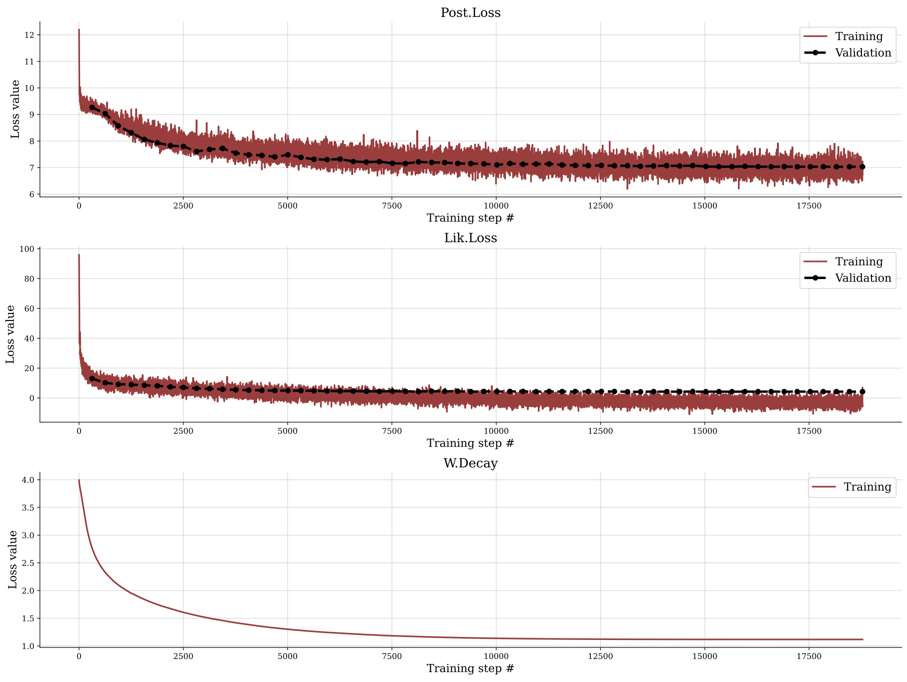


(b) Posterior calibration.

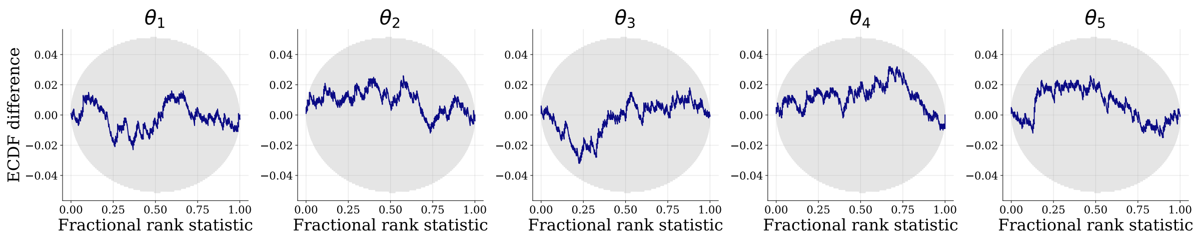


(c) Joint calibration.

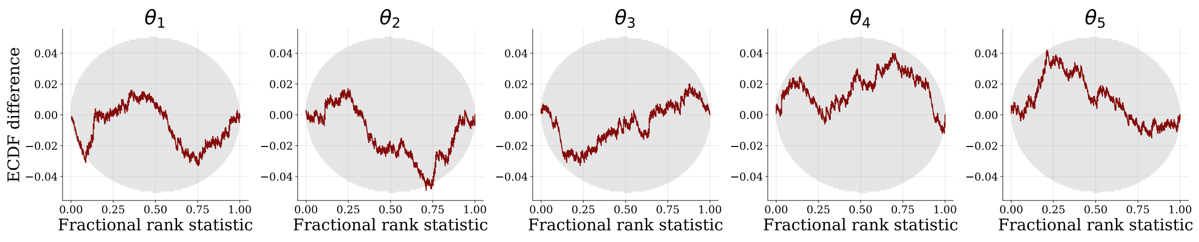
Figure 3: **Benchmark 3, Simple Likelihood Complex Posterior.** Loss history, posterior calibration, and joint calibration.



(a) Training and validation loss history.

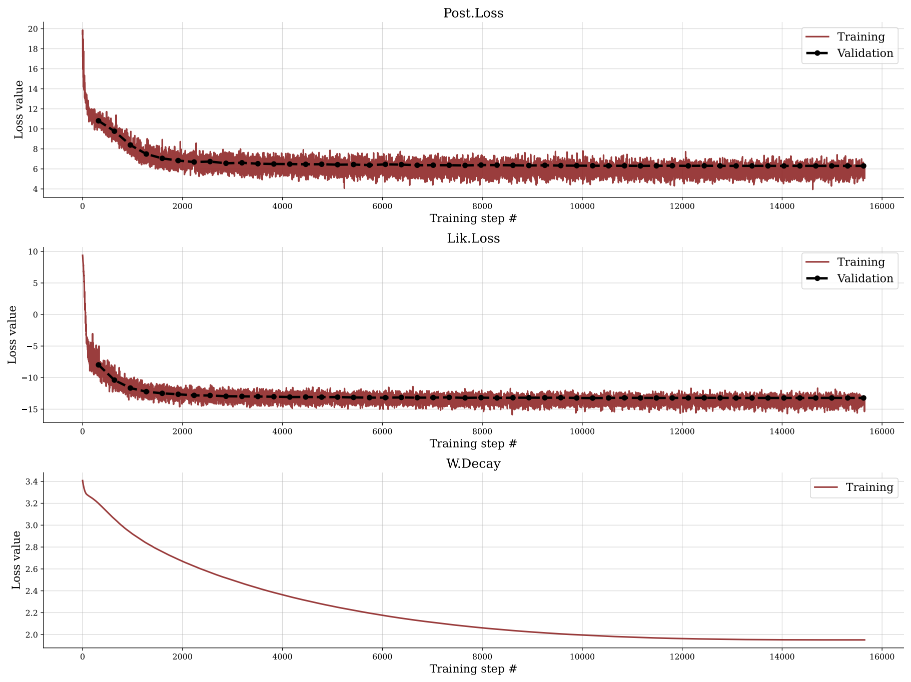


(b) Posterior calibration.

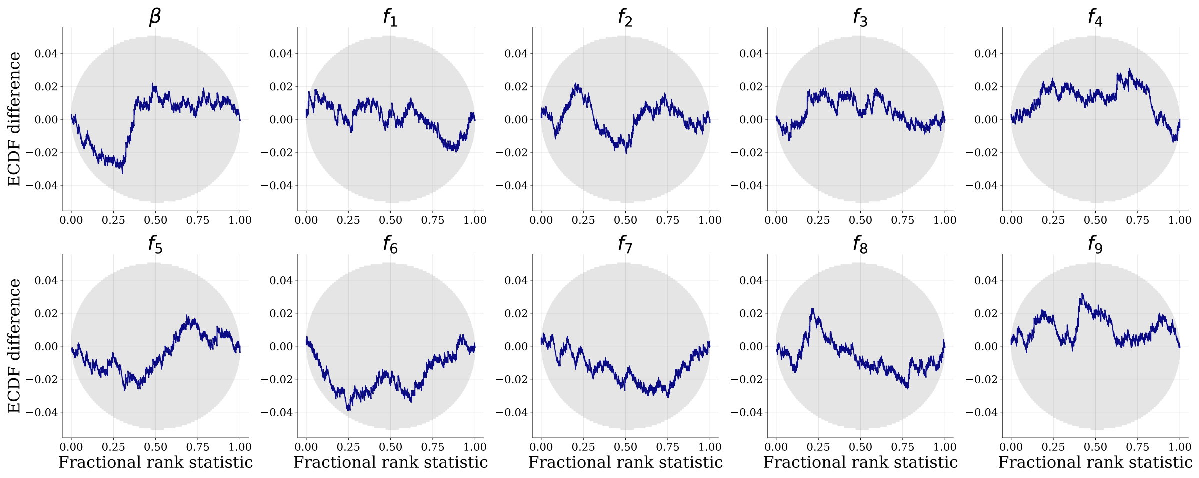


(c) Joint calibration.

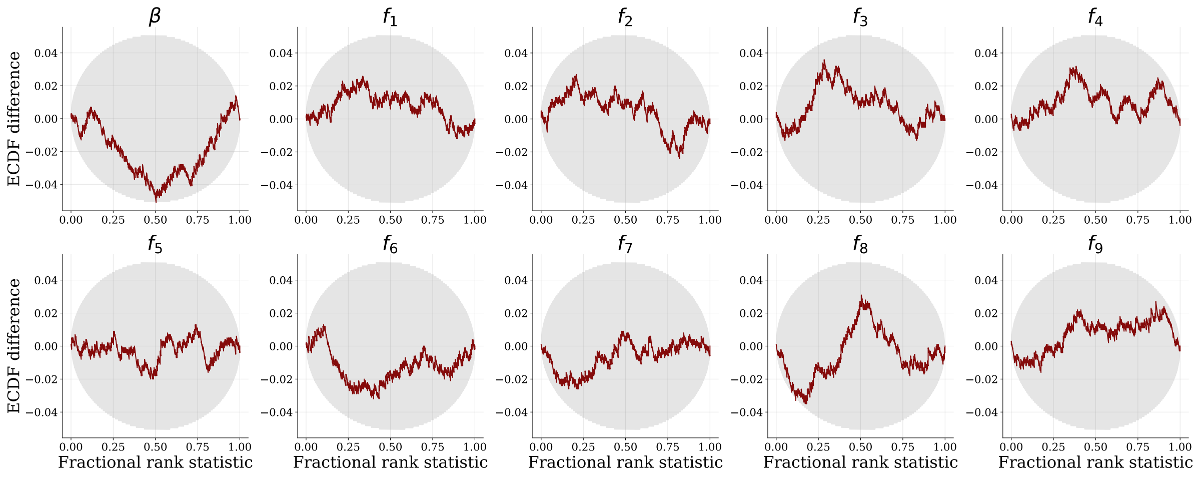
Figure 4: **Benchmark 4, Simple Likelihood Complex Posterior with Distractors.** Loss history, posterior calibration, and joint calibration.



(a) Training and validation loss history.

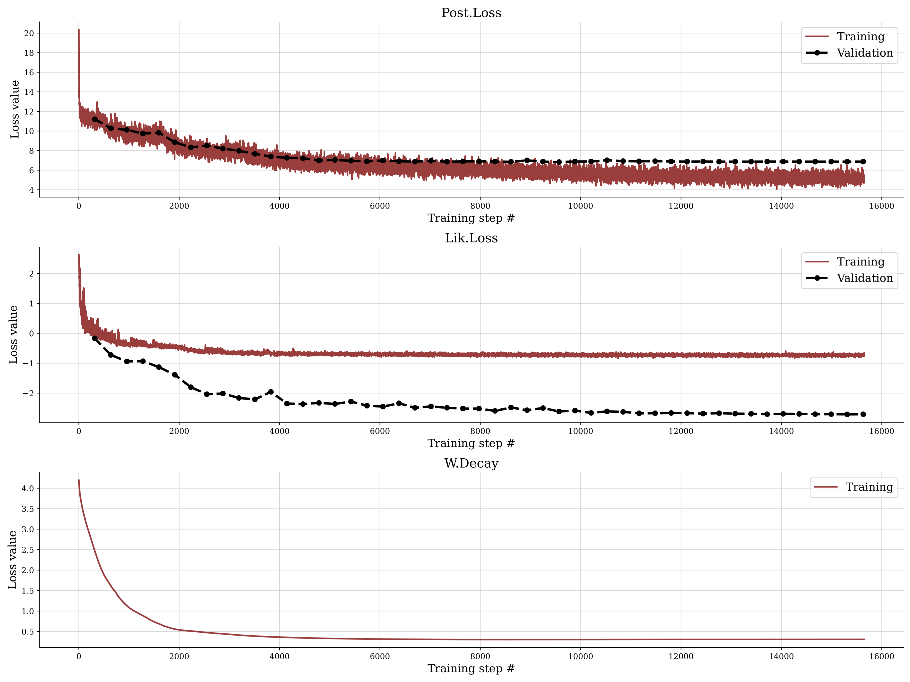


(b) Posterior calibration.

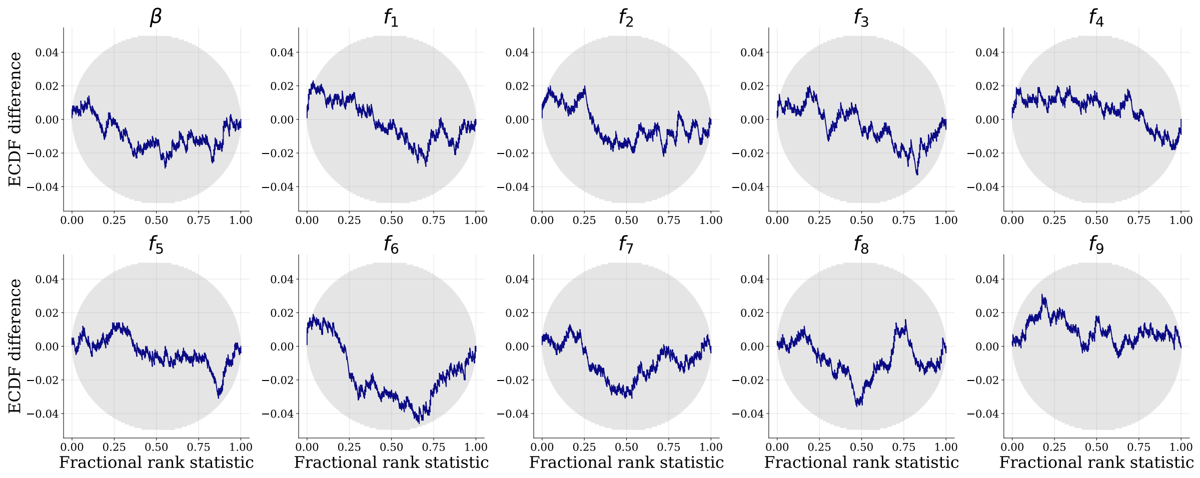


(c) Joint calibration.

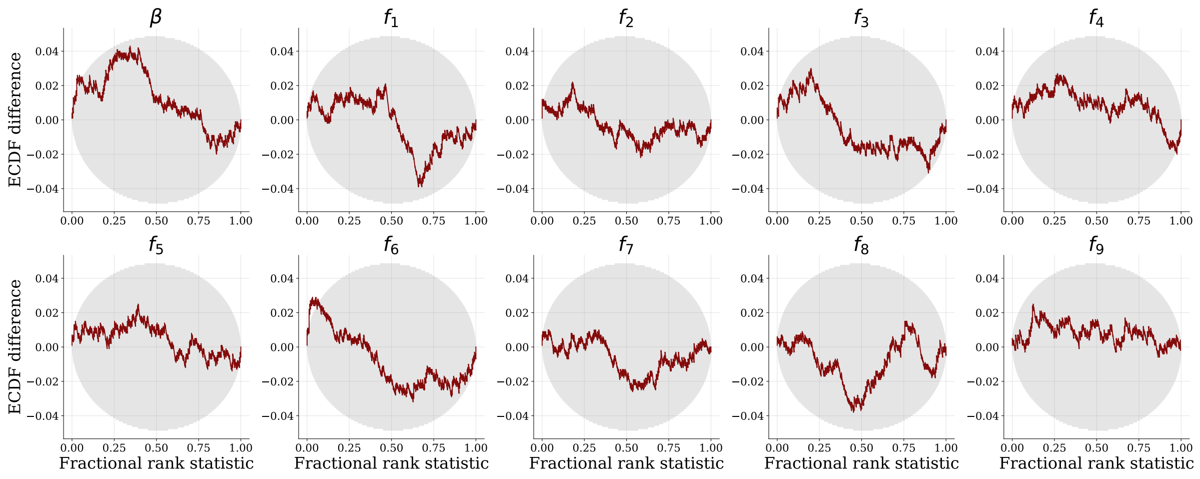
Figure 5: **Benchmark 5, Bernoulli GLM.** Loss history, posterior calibration, and joint calibration.



(a) Training and validation loss history.

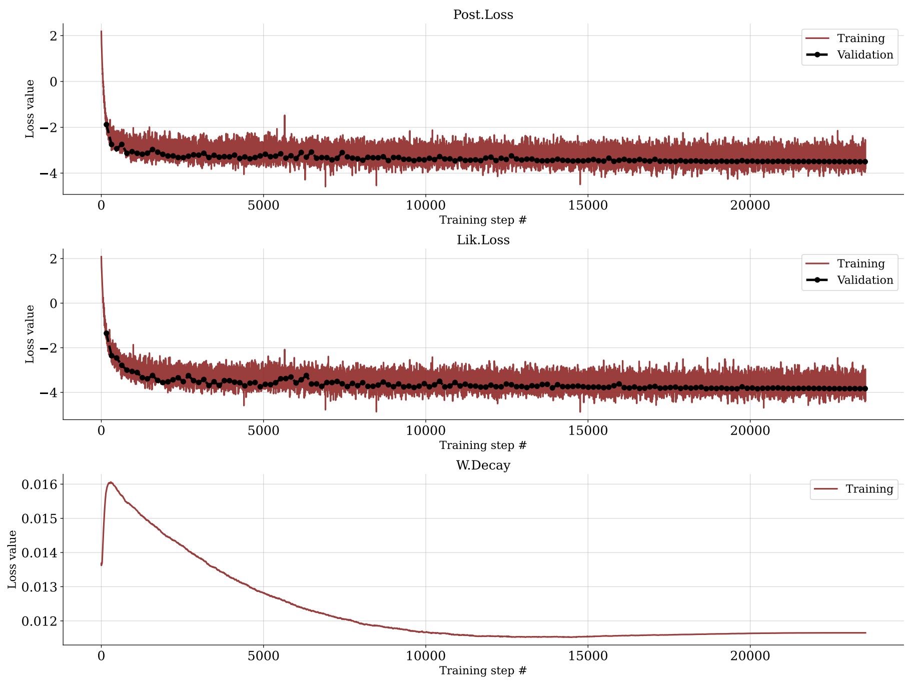


(b) Posterior calibration.

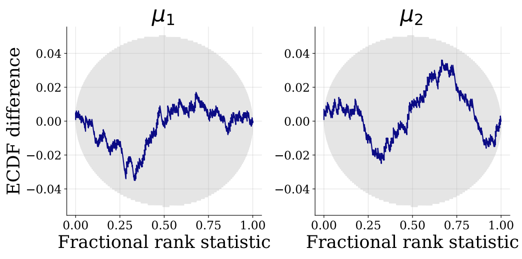


(c) Joint calibration.

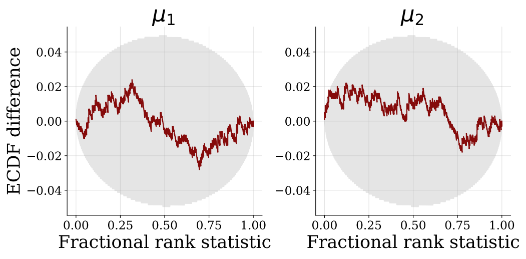
Figure 6: **Benchmark 6, Bernoulli GLM raw.** Loss history, posterior calibration, and joint calibration.



(a) Training and validation loss history.

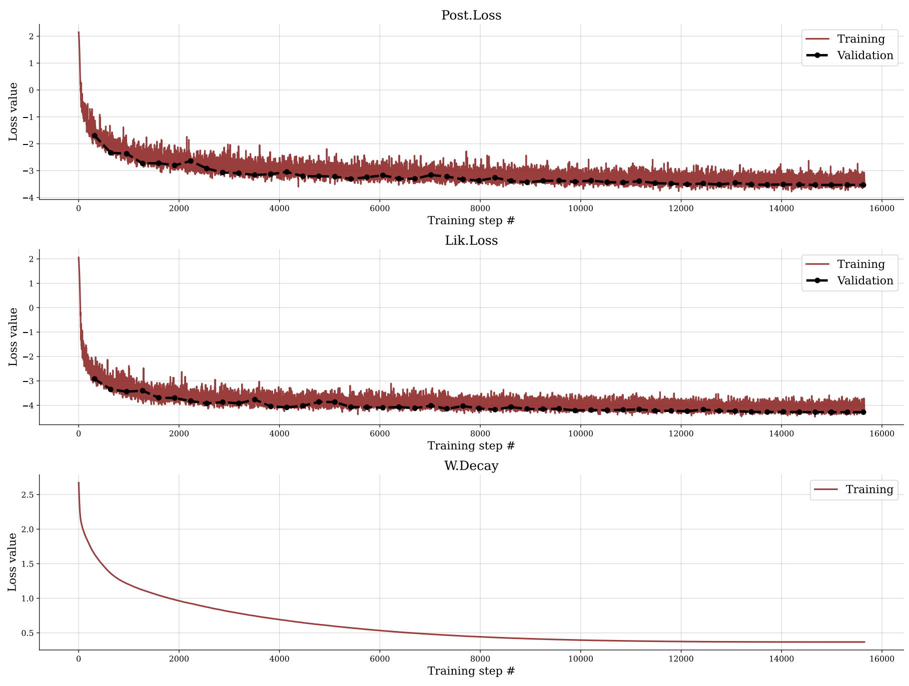


(b) Posterior calibration.

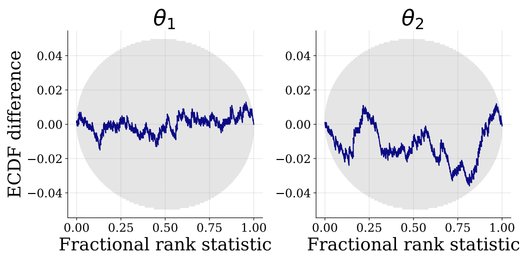


(c) Joint calibration.

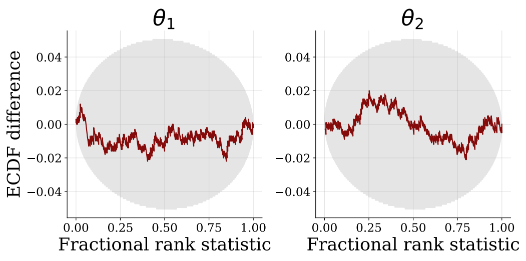
Figure 7: **Benchmark 7, Gaussian Mixture.** Loss history, posterior calibration, and joint calibration.



(a) Training and validation loss history.

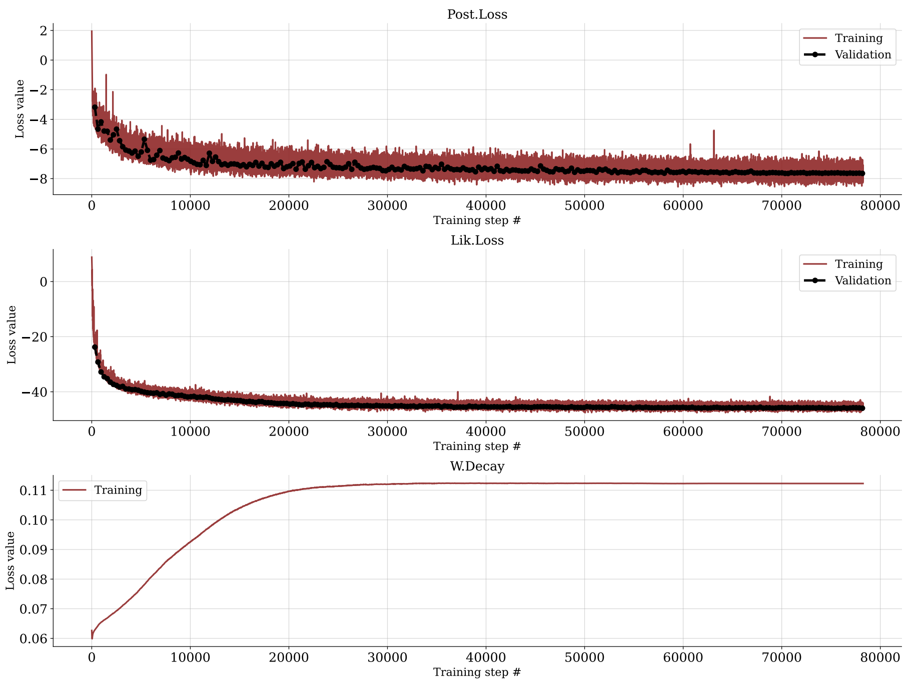


(b) Posterior calibration.

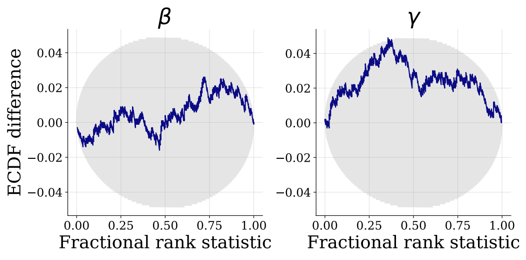


(c) Joint calibration.

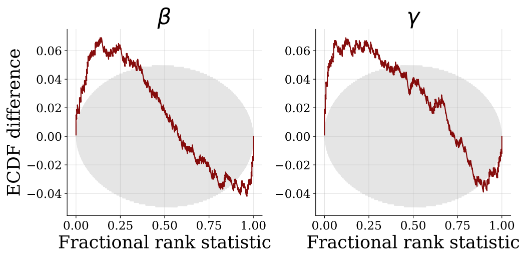
Figure 8: **Benchmark 8, Two Moons.** Loss history, posterior calibration, and joint calibration.



(a) Training and validation loss history.

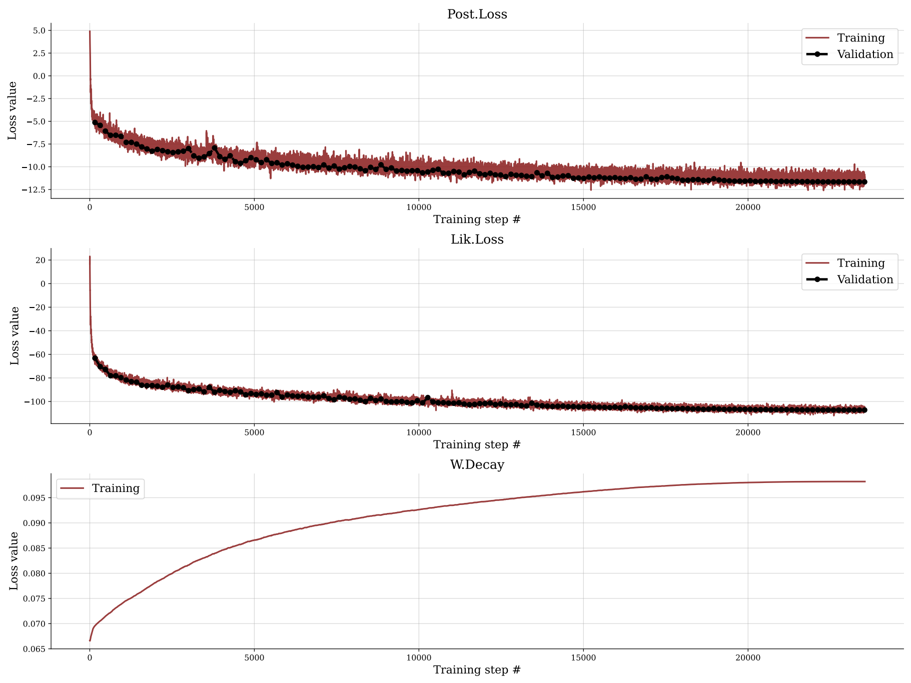


(b) Posterior calibration.

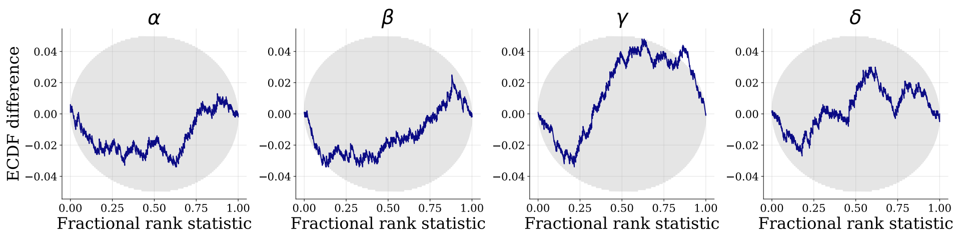


(c) Joint calibration.

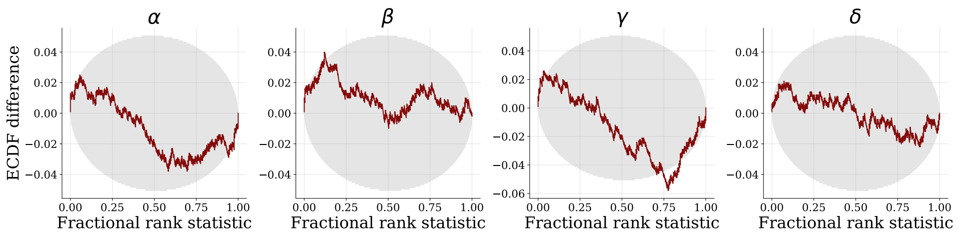
Figure 9: **Benchmark 9, SIR time series.** Loss history, posterior calibration, and joint calibration.



(a) Training and validation loss history.



(b) Posterior calibration.



(c) Joint calibration.

Figure 10: **Benchmark 10, Lotka-Volterra.** Loss history, posterior calibration, and joint calibration.

D.2 EXPERIMENT 2: TWO MOONS

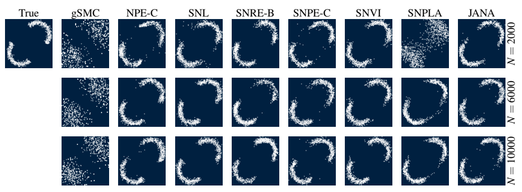
Model details This experiment utilizes the two moons simulator from Greenberg et al. (2019) – not to be confused with the standard two moons data set used for unconditional estimation – with the same experimental setup as described in Wiqvist et al. (2021).

Network and training details The posterior network is a neural spline flow with 4 coupling layers and a Gaussian latent space. The likelihood network uses an interleaved coupling architecture with 5 coupling layers. We train the networks in an offline fashion on the respective simulation budget ($2\,000$, $6\,000$ and $10\,000$ simulations) for 64 epochs with a batch size of 32 and a learning rate of 0.0005.

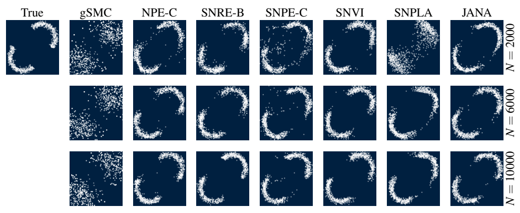
The wall-clock times on a consumer-grade CPU are listed in Table 2. While the JANA implementation in the BayesFlow framework would certainly benefit from GPU acceleration, the available implementations of SNPLA and SNVI do not come with GPU support out-of-the-box due to their APIs to dependent packages (i.e., issues with Pyro for SNVI and issues with PyTorch for SNPLA). We repeat the training phase of each method 10 times to further investigate the reliability of the methods. We only conducted one repetition with SNL due to the prohibitively slow run time (see Table 2).

	NPE-C	SNPE-C	SNRE-B	SNL	SNVI	SNPLA	JANA
Training (seconds)	229	1151	5533	17492	198	496	435
Posterior Inference (seconds)	0.02	0.02	592.63	1890	0.60	0.01	0.13
Posterior Predictive Inference (seconds)	—	—	—	1872	0.61	0.03	0.27

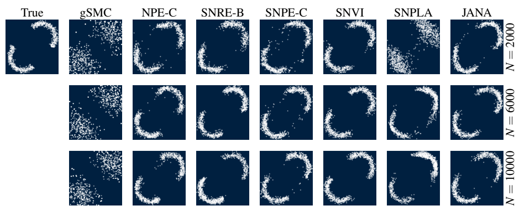
Table 2: Average wall-clock times (seconds) on a consumer-grade CPU for different neural methods. Training time is based on offline learning with $10\,000$ simulations. Posterior inference indicates wall-clock time for obtaining $1\,000$ samples from the approximate posterior on a single observation. Posterior predictive inference indicates wall-clock time for obtaining $1\,000$ samples from the approximate posterior and evaluating the approximate likelihood of each sample. Note, that NPE-C and JANA are *amortized*, so no further training is needed for applications on new observations.



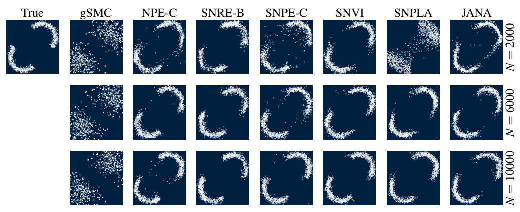
(a) Repetition #1 (main paper)



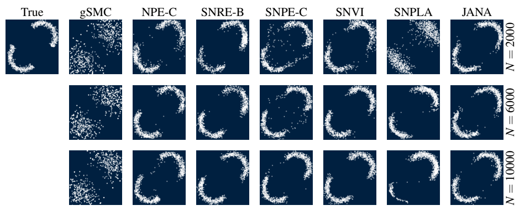
(b) Repetition #2



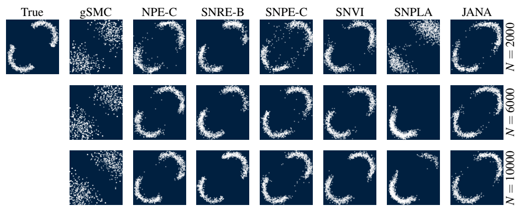
(c) Repetition #3



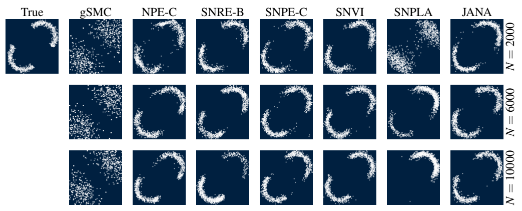
(d) Repetition #4



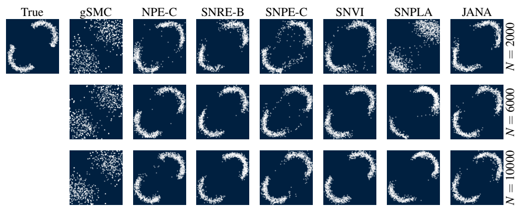
(e) Repetition #5



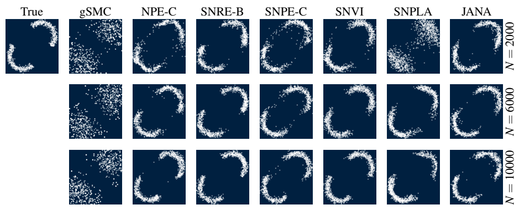
(f) Repetition #6



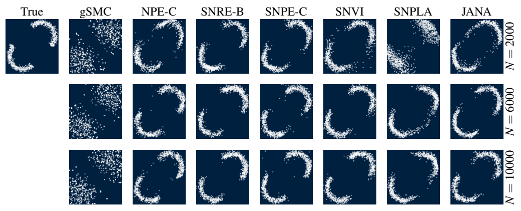
(g) Repetition #7



(h) Repetition #8



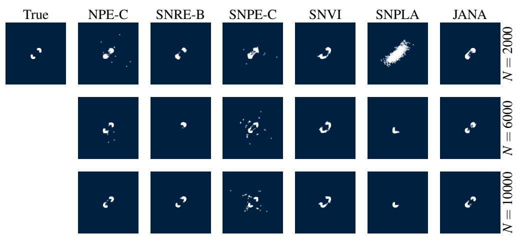
(i) Repetition #9



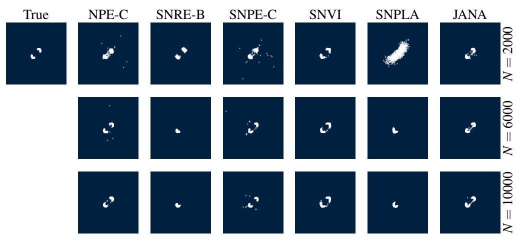
(j) Repetition #10

Figure 11: **Experiment 2.** 1 000 Posterior draws for all methods and repetitions of the experiment. The main paper shows repetition #1, which is in line with the other runs. The axis limits represent the support of the uniform prior distribution.

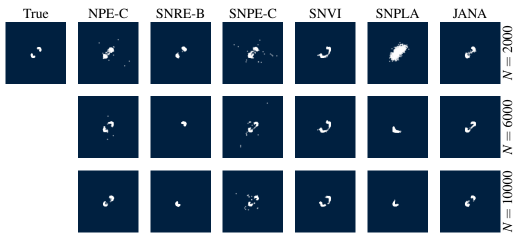
Different Simulator We repeat the experiment with the simulator from Lueckmann et al. (2021), which produces smaller moons with larger relative distance. The only difference to Lueckmann et al. (2021) is that we use a broader uniform prior with bounds $[-2, 2]$ (instead of $[-1, 1]$) to further increase the difficulty of the task. The results are largely equivalent to the ones reported in the main text.



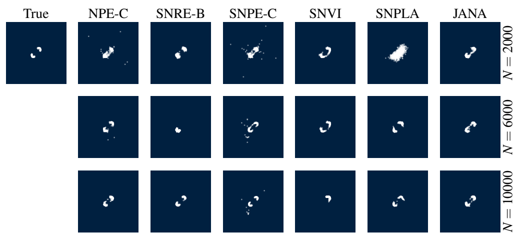
(a) Repetition #1 (main paper)



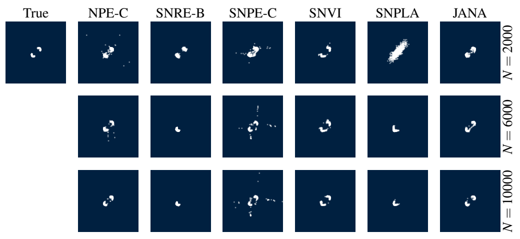
(b) Repetition #2



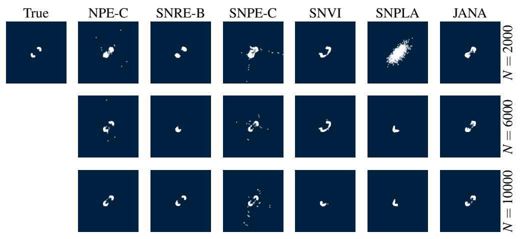
(c) Repetition #3



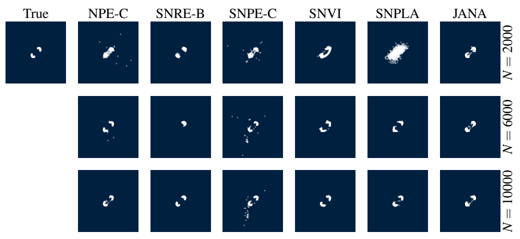
(d) Repetition #4



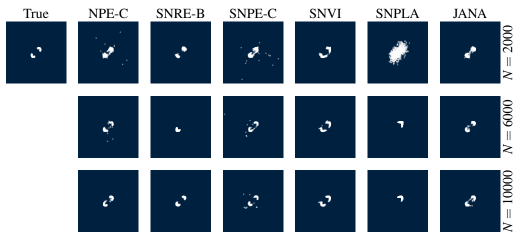
(e) Repetition #5



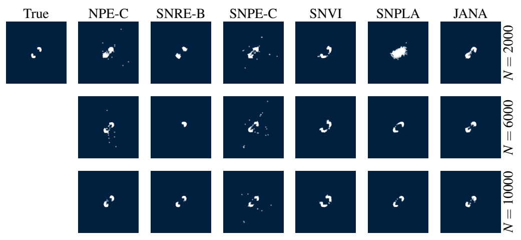
(f) Repetition #6



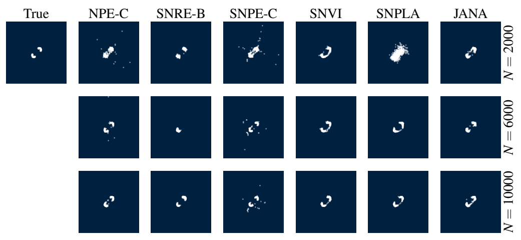
(g) Repetition #7



(h) Repetition #8



(i) Repetition #9



(j) Repetition #10

Figure 12: **Experiment 2.** 1 000 Posterior draws for all methods and repetitions of the experiment with a more challenging simulator. The axis limits represent the support of the uniform prior distribution.

D.3 EXPERIMENT 3: EXCHANGEABLE DIFFUSION MODEL

Model details We focus on the drift diffusion model (DDM)—a cognitive model describing reaction times (RTs) in binary decision tasks (Ratcliff & McKoon, 2008). The DDM assumes that perceptual information for a choice alternative accumulates continuously according to a Wiener diffusion process. The change in information accumulation $\mathrm{d}x$ follows a random walk with drift and Gaussian noise:

$$\mathrm{d}x = v\mathrm{d}t + \xi\sqrt{\mathrm{d}t} \quad \text{with} \quad \xi \sim \mathcal{N}(0, 1). \tag{16}$$

The model consists of four parameters: drift-rate v, boundary separation a, non-decision time t_0 and bias (relative starting point) w. The model has the particularity of being very sensible to early outliers, as all reaction times smaller than the non-decision time are considered impossible (i.e., have a likelihood of zero). We employ the simple DDM, as its likelihood function is tractable (Voss & Voss, 2007), and place truncated normal priors over the parameters $\boldsymbol{\theta} = (v, a, t_0, w)$,

$$v \sim \mathcal{TN}_{[-5,5]}(0, 10), \quad a \sim \mathcal{TN}_{[0.5,3]}(1, 1), \quad t_0 \sim \mathcal{TN}_{[0.2,1]}(0.4, 0.2), \quad w \sim \mathcal{TN}_{[0.3,0.7]}(0.5, 0.1), \tag{17}$$

where $\mathcal{TN}_{[a,b]}(\mu, \sigma)$ denotes the truncated normal distribution with location μ and standard deviation σ truncated within the interval $[a, b]$. The summary network is a permutation-invariant network which reduces simulated IID RT data sets to $S = 10$ summary statistics (Radev et al., 2020).

Network and training details The summary network is a deep permutation-invariant network with 2 equivariant modules followed by an invariant module (Bloem-Reddy & Teh, 2020; Radev et al., 2020). The summary network reduces the IID RT data sets into 10-dimensional learned summary statistics. The posterior network is a conditional invertible neural network (cINN) with 5 conditional coupling layers and a Student-t latent space ($df = 50$). The internal networks of the coupling layers are fully connected (FC) networks with 2 hidden layers featuring 128 units and `tanh` activation function.

The likelihood network is a cINN with 12 conditional coupling layers, with smaller internal FC networks of 2 hidden layers having 32 units each, a `tanh` activation function, and a Student-t latent space. We train the networks in an offline fashion. The likelihood network is trained for 20 epochs with a batch size of 64 and a learning rate of 0.001. The posterior network is trained for 100 epochs with a batch size of 64 and a learning rate of 0.002.

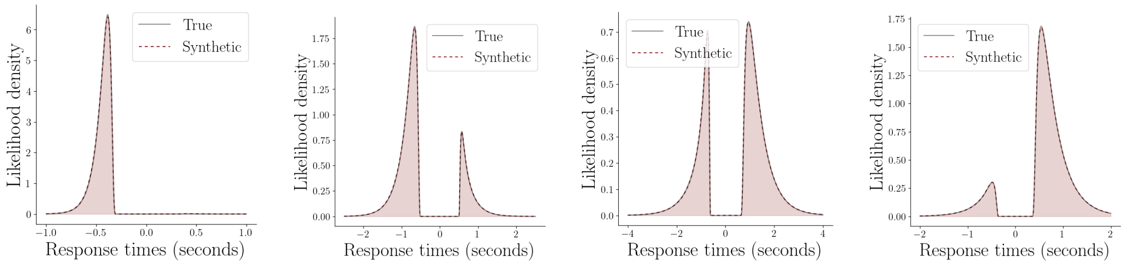


Figure 13: **Experiment 3.** JANA exhibits essentially perfect likelihood emulation for various parameter configurations.

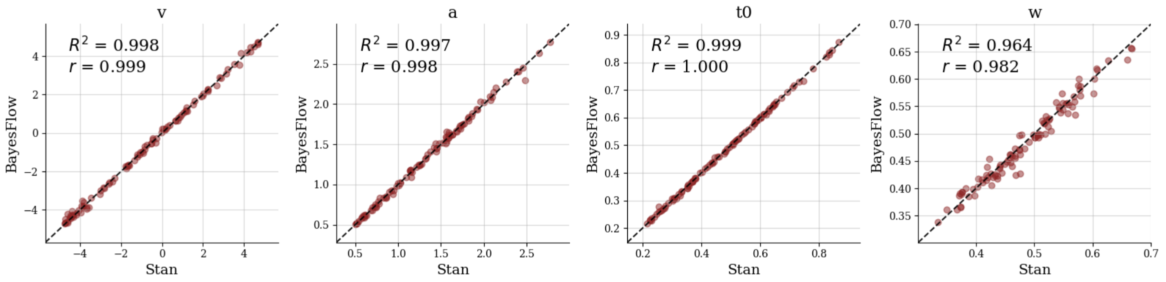


Figure 14: **Experiment 3.** The parameter recovery of JANA is largely identical to the estimates obtained via the gold-standard HMC-MCMC implementation in Stan.

D.4 EXPERIMENT 4: MARKOVIAN COMPARTMENTAL MODEL

Model details We use the model formulation from Radev, D'Alessandro, et al. (2021), which consists of three components: 1) a latent sub-model, 2), an intervention sub-model; 3) and an observation sub-model.

First, the latent sub-model is a SIR-type system of non-linear ordinary differential equations (ODEs) with six population compartments representing the interactions between susceptible (S), exposed (E), infected (I), carrier (C), recovered (R), and dead (D) individuals. The interaction dynamics are governed by:

$$\frac{dS}{dt} = -\lambda(t) \left(\frac{C + \beta\, I}{N}\right) S \tag{18}$$

$$\frac{dE}{dt} = \lambda(t) \left(\frac{C + \beta\, I}{N}\right) S - \gamma\, E \tag{19}$$

$$\frac{dC}{dt} = \gamma\, E - (1-\alpha)\, \eta\, C - \alpha\, \theta\, C \tag{20}$$

$$\frac{dI}{dt} = (1-\alpha)\, \eta\, C - (1-\delta)\, \mu\, I - \delta\, d\, I \tag{21}$$

$$\frac{dR}{dt} = \alpha\, \theta\, C + (1-\delta)\, \mu\, I \tag{22}$$

$$\frac{dD}{dt} = \delta\, d\, I \tag{23}$$

For simulating the system, we use $dt = 1$ which corresponds to a time scale of days.

Second, an *intervention sub-model* accounts for changes in the transmission rate $\lambda(t)$ due to non-pharmaceutical policies. It defines three change points for $\lambda(t)$ encoding an assumed transmission rate reduction in response to intervention measures imposed by the German authorities in 2020. Each change point is a piece-wise linear function with three parameters: the effect strength and the boundaries defining the time interval for the effect to take place (Radev, Graw, et al., 2021).

The observation sub-model assumes that only compartments I, R, and D are potentially observable. Moreover, it accounts for the fact that officially reported cases might not represent the true latent numbers of an outbreak:

$$I_t^{(obs)} = I_{t-1}^{(obs)} + (1 - f_I(t))\, (1-\alpha)\, \eta\, C_{t-L_I} + \sqrt{I_{t-1}^{(obs)}}\, \sigma_I\, \xi_t \tag{24}$$

$$R_t^{(obs)} = R_{t-1}^{(obs)} + (1 - f_R(t))\, (1-\delta)\, \mu\, I_{t-L_R} + \sqrt{R_{t-1}^{(obs)}}\, \sigma_R\, \xi_t \tag{25}$$

$$D_t^{(obs)} = D_{t-1}^{(obs)} + (1 - f_D(t))\, \delta\, d\, I_{t-L_D} + \sqrt{D_{t-1}^{(obs)}}\, \sigma_D\, \xi_t \tag{26}$$

In the above equations, L_I, L_R, and L_D denote the reporting delays (lags), and denote σ_I, σ_R, and σ_D the scales of multiplicative reporting noise for the respective compartments. The noise variables ξ_t follow a Student-*t* distribution with 4 degrees of freedom. The weekly modulation of reporting coverage $f_\mathcal{C}(t)$ for each of the compartments $\mathcal{C} \in \{I, R, D\}$ is computed as follows:

$$f_\mathcal{C}(t) = (1 - A_\mathcal{C}) \left(1 - \left|\sin\left(\frac{\pi}{7} t - 0.5\, \Phi_\mathcal{C}\right)\right|\right) \tag{27}$$

This yields three additional unknown parameters for the weekly modulation amplitudes A_I, A_R, A_D, and phases Φ_I, Φ_R, Φ_D, each.

Network and training details The summary network is a combination of 1D convolutional and LSTM layers, which reduce the multivariate time series into a vector of 192 learned summary statistics (Radev, Graw, et al., 2021). The posterior network is a conditional invertible neural network (cINN) with 6 conditional affine coupling layers. The internal networks of the coupling layers are fully connected (FC) networks with 2 hidden layers of 128 units and a `swish` activation function. The likelihood network is a recurrent cINN with 8 conditional coupling layers with the same structure as the coupling layers of the posterior network. We use a gated recurrent unit (GRU) with 256 hidden units for the internal recurrent memory. We train the networks in an online fashion (i.e., on-the-fly simulations) for 100 epochs with a batch size of 32 and a learning rate of 0.0005. This initial learning rate is reduced throughout the training phase following a cosine decay schedule with a minimum learning rate of 0.

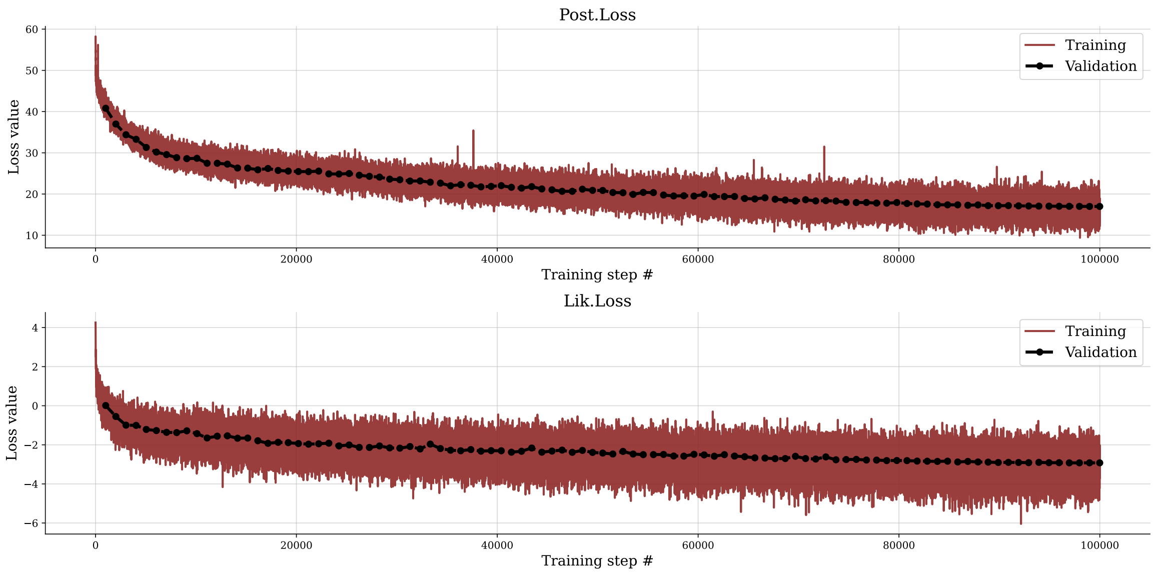
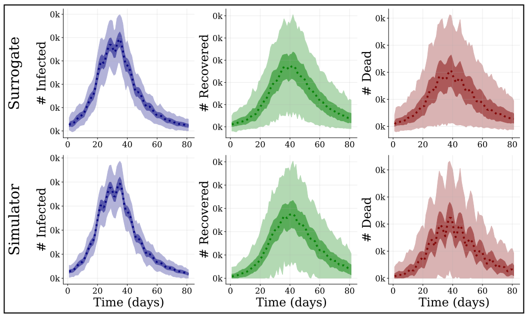
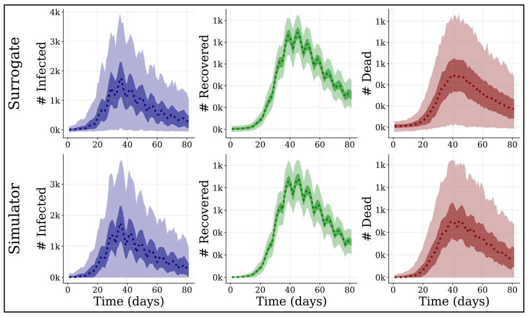


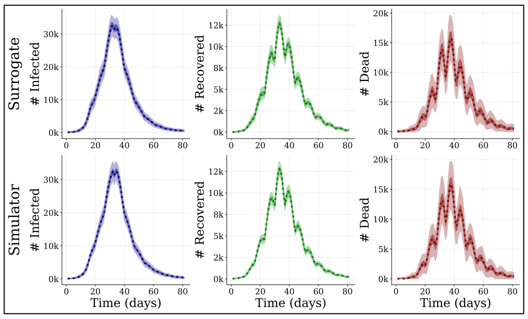
Figure 15: **Experiment 4.** Loss history



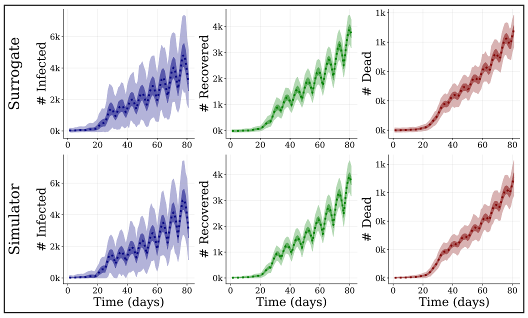
(a) Scenario I



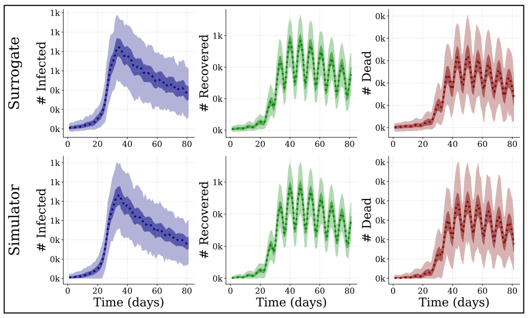
(b) Scenario II



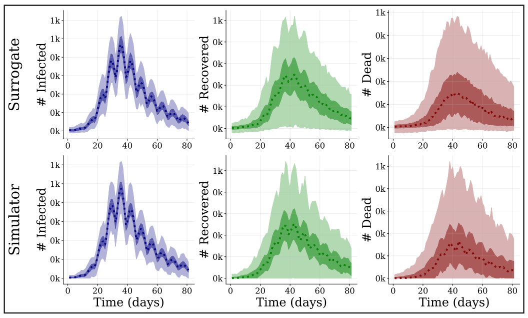
(c) Scenario III



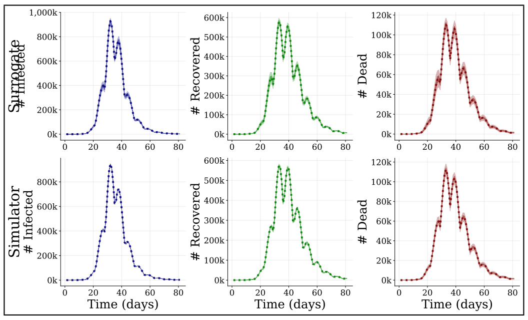
(d) Scenario IV



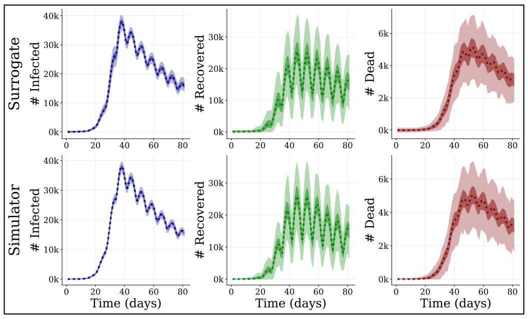
(e) Scenario V



(f) Scenario VI

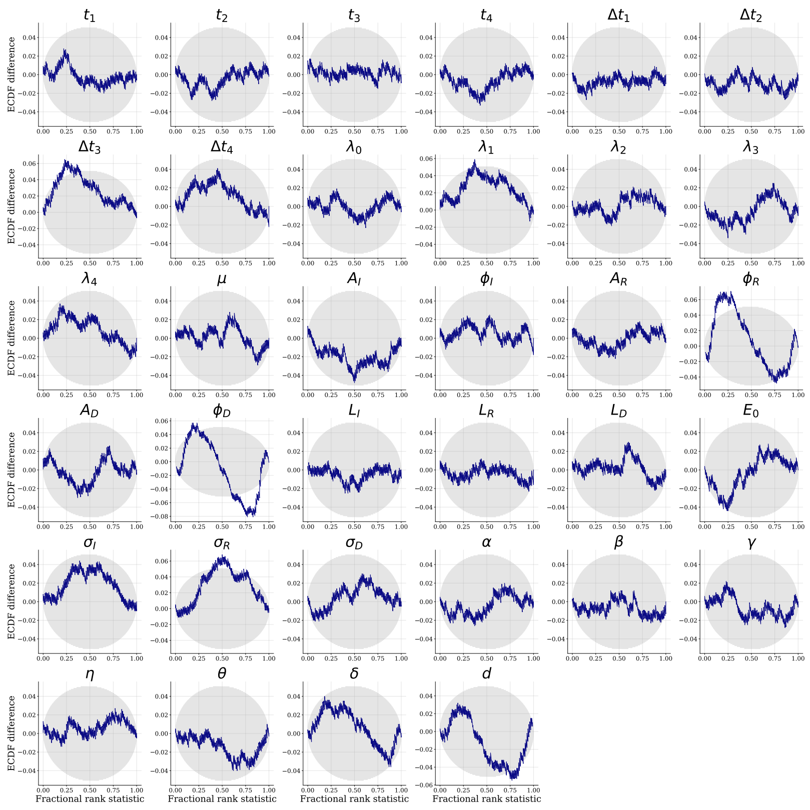


(g) Scenario VII



(h) Scenario VIII

Figure 16: **Experiment 4.** The likelihood network can emulate the simulator and its aleatoric uncertainty strikingly well. Each sub-panel depicts 1000 runs from the original and the surrogate neural simulator given the same parameter configuration, each leading to a qualitatively different outbreak scenario.


(a) Posterior calibration

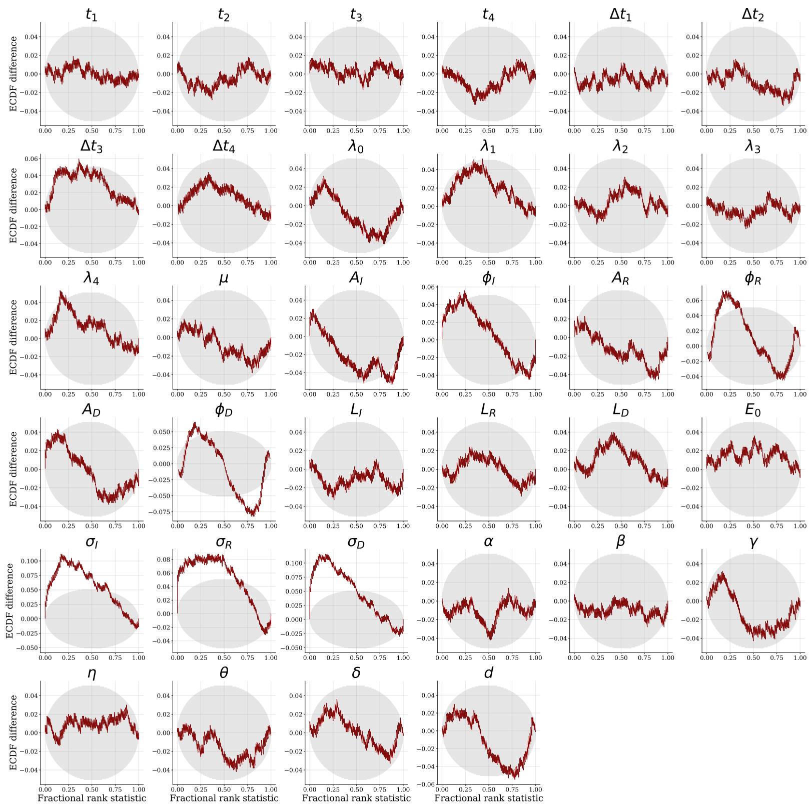

(b) Joint calibration

Figure 17: **Experiment 4.** Posterior and joint calibration results

D.5 EXPERIMENT 5: HIGH-DIMENSIONAL BAYESIAN DENOISING

Model Details This experiment follows the problem formulation from Pacchiardi and Dutta (2022) and Ramesh et al. (2022). However, we choose the Fashion MNIST data set because of its richer and more interesting structure. In this Bayesian denoising setup, a simulated noisy camera applies a multidimensional Gaussian filter (i.e., a blur) to each Fashion MNIST image. Thus, the original image represents the "parameters" $\boldsymbol{\theta} \in \mathbb{R}^{784}$ and its blurry version $\mathbb{R}^{784}$ represents the "observation". In order to make the problem more challenging, we do not use the class label as an additional conditioning input for the networks. We also do not process the image data optimally (e.g., by applying a Haar wavelet downsampling or using convolutional couplings, as in Ardizzone et al., 2019; Kingma & Dhariwal, 2018), as our goal is not to perform high-quality image reconstruction, but simply to illustrate the applicability of JANA for analyzing potentially high-dimensional Bayesian models.

Network and training details Since both "data" and "parameters" are images with a (theoretically) lower intrinsic dimensionality than the total number of pixels, both the likelihood and the posterior network utilize a separate summary network with identical architecture. For each, we use a 4-layer fully convolutional network with a final global average pooling layer yielding a 128-dimensional summary representation of the original and blurry image, respective. The posterior network is a conditional invertible neural network (cINN) comprising 12 conditional affine coupling layers. The internal networks of the coupling layers are fully connected (FC) networks with a single hidden layer of 512 units and a `ReLU` non-linearity. The likelihood network uses the same architecture as the posterior network. Finally, we use a multivariate Student-T latent space (Alexanderson & Henter, 2020), as it allows us to perform a much more stable maximum likelihood training with higher learning rates.

We train the networks on the official training set of $60\,000$ Fashion MNIST images for 120 epochs with a batch size of 32 and a learning rate of 0.001. This initial learning rate is reduced throughout the training phase following a cosine decay schedule with a minimum learning rate of 0. For each batch, we add a small amount of Gaussian noise with a scale of 0.001 as a form of dequantization (Ardizzone et al., 2019). We use 500 images from the test set as a validation set to estimate the generalization error during training. We utilize the remaining 9500 images from the test set for evaluating the approximation quality and calibration of the networks.

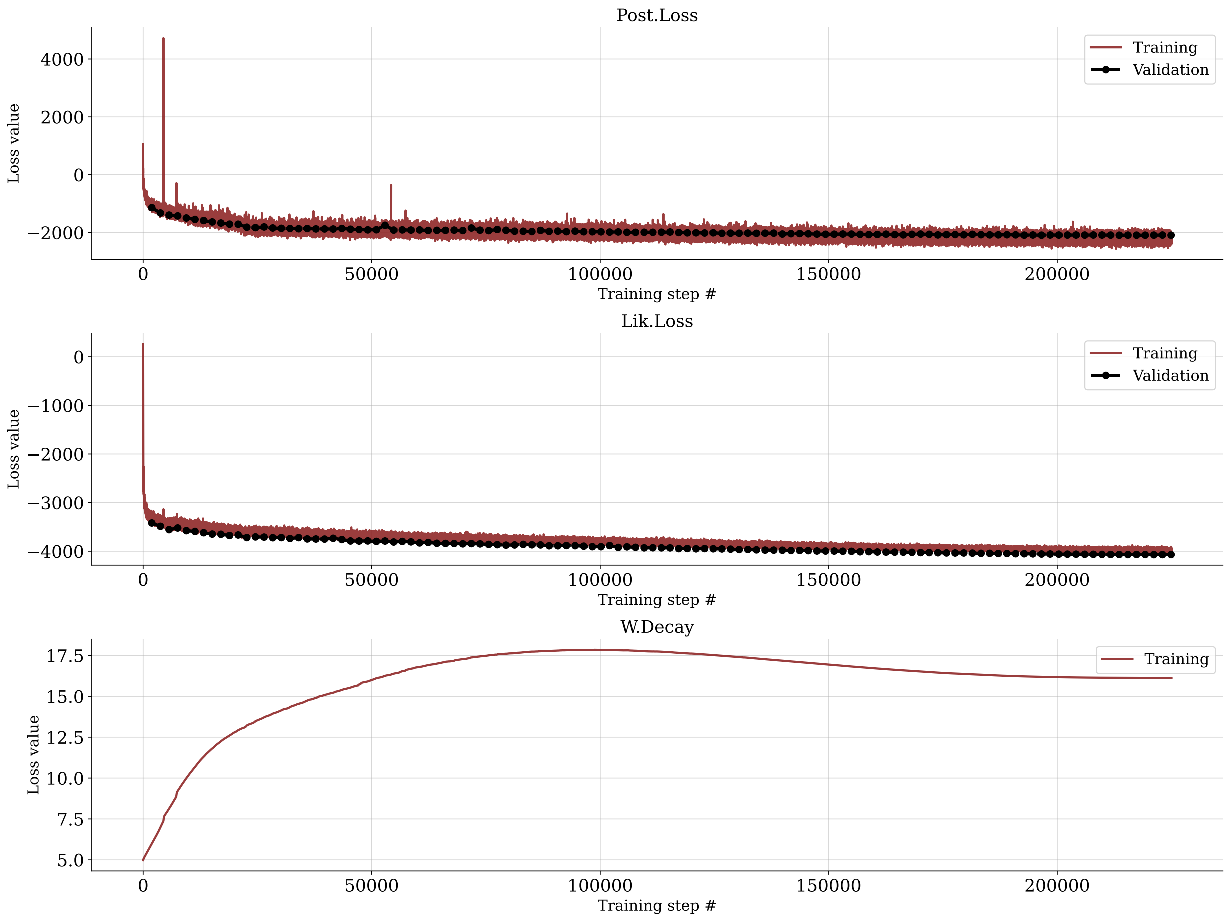
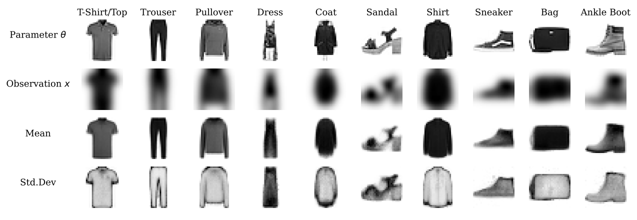
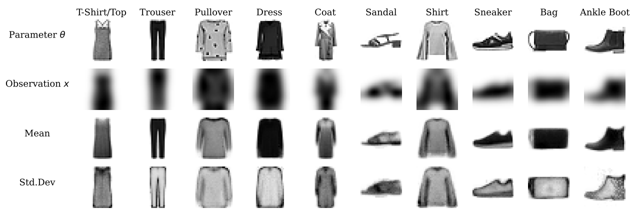


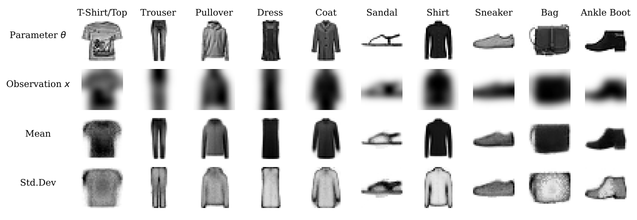
Figure 18: **Experiment 5.** Loss history



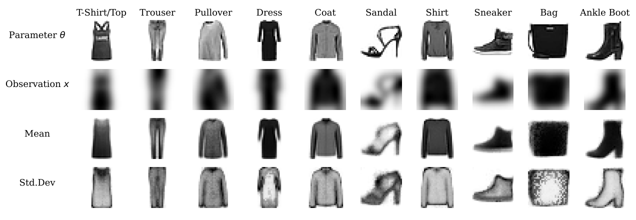
(a) Posterior estimation #1



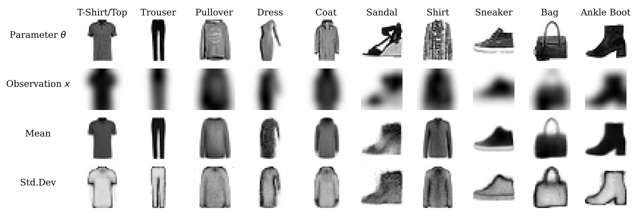
(b) Posterior estimation #2



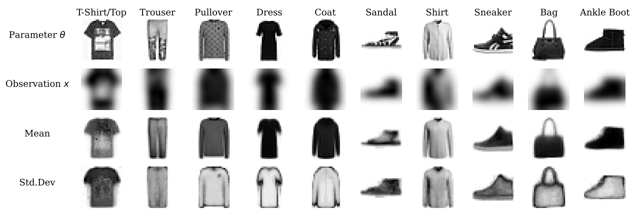
(c) Posterior estimation #3



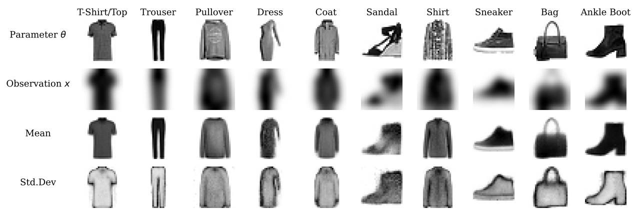
(d) Posterior estimation #4



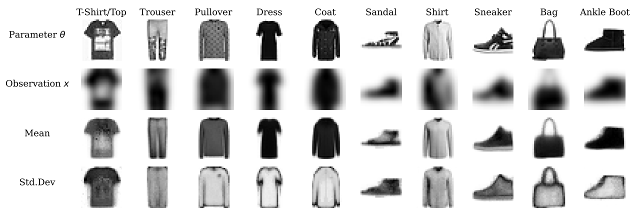
(e) Posterior estimation #5



(f) Posterior estimation #6



(g) Posterior estimation #7



(h) Posterior estimation #8

Figure 19: **Experiment 5.** Posterior (denoising) results on 8 randomly selected sets of images from the official Fashion MNIST test set.

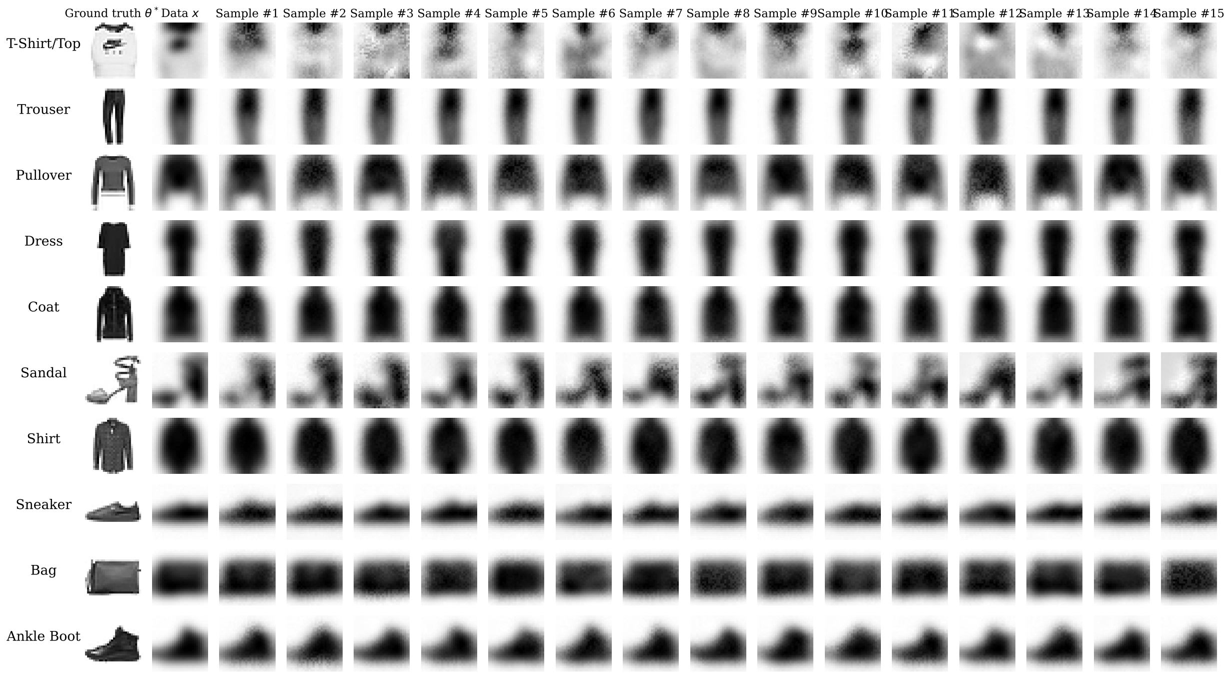


Figure 20: **Experiment 5.** Samples from the surrogate camera (i.e., the "likelihood" in the Bayesian denoising setup) given ten randomly selected clean images (i.e., "parameters") from each class.

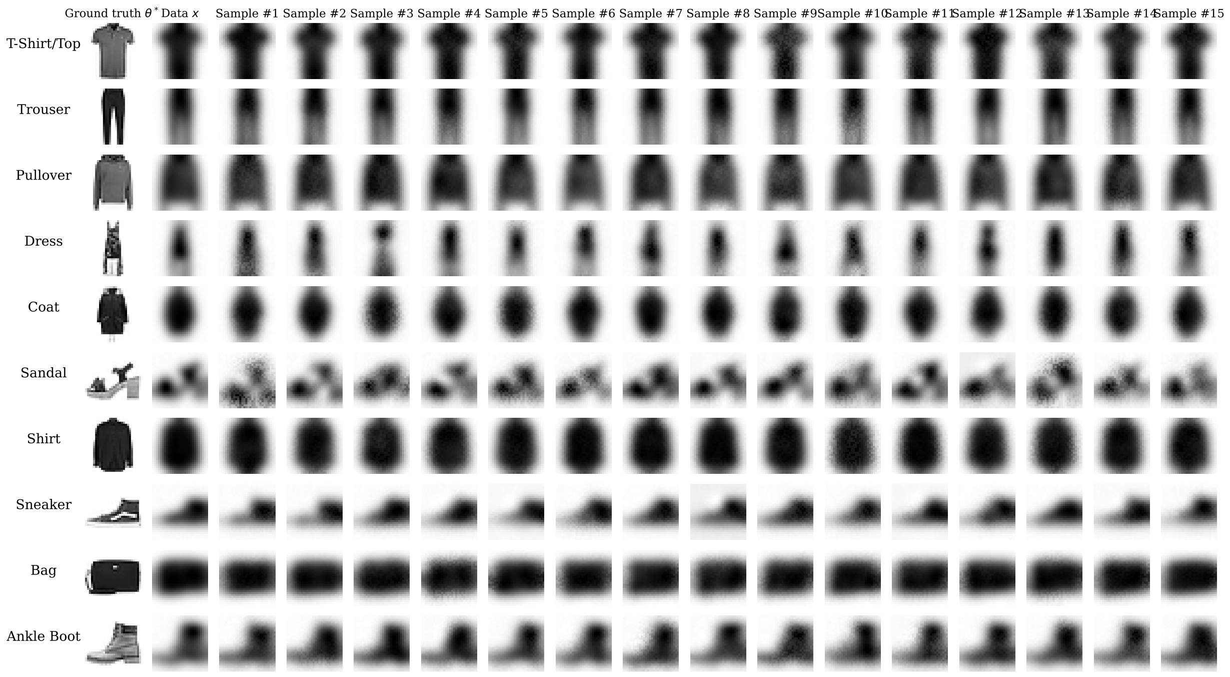


Figure 21: **Experiment 5.** Further samples from the surrogate camera.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., et al. (2016). Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*.

Alexanderson, S., & Henter, G. E. (2020). Robust model training and generalisation with studentising flows. *arXiv preprint arXiv:2006.06599*.

Ardizzone, L., Lüth, C., Kruse, J., Rother, C., & Köthe, U. (2019). Guided image generation with conditional invertible neural networks. *arXiv preprint*.

Bloem-Reddy, B., & Teh, Y. W. (2020). Probabilistic symmetries and invariant neural networks. *The Journal of Machine Learning Research*, *21*(1), 3535–3595.

Durkan, C., Bekasov, A., Murray, I., & Papamakarios, G. (2019). Neural spline flows. *Advances in neural information processing systems*, *32*.

Greenberg, D., Nonnenmacher, M., & Macke, J. (2019). Automatic posterior transformation for likelihood-free inference. *International Conference on Machine Learning*.

Gretton, A., Borgwardt, K., Rasch, M., Schölkopf, B., & Smola, A. (2012). A Kernel Two-Sample Test. *The Journal of Machine Learning Research*, *13*, 723–773.

Kim, H., Lee, H., Kang, W. H., Lee, J. Y., & Kim, N. S. (2020). Softflow: Probabilistic framework for normalizing flow on manifolds. *Advances in Neural Information Processing Systems*, *33*, 16388–16397.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kingma, D. P., & Dhariwal, P. (2018). Glow: Generative flow with invertible 1x1 convolutions. *Advances in neural information processing systems*, *31*.

Lueckmann, J.-M., Boelts, J., Greenberg, D. S., Gonçalves, P. J., & Macke, J. H. (2021). Benchmarking simulation-based inference. *arXiv preprint*.

Pacchiardi, L., & Dutta, R. (2022). Score matched neural exponential families for likelihood-free inference. *J. Mach. Learn. Res.*, *23*, 38–1.

Radev, S. T., D'Alessandro, M., Mertens, U. K., Voss, A., Köthe, U., & Bürkner, P.-C. (2021). Amortized bayesian model comparison with evidential deep learning. *IEEE Transactions on Neural Networks and Learning Systems*.

Radev, S. T., Graw, F., Chen, S., Mutters, N. T., Eichel, V. M., Bärnighausen, T., & Köthe, U. (2021). Outbreakflow: Model-based bayesian inference of disease outbreak dynamics with invertible neural networks and its application to the covid-19 pandemics in germany. *PLoS computational biology*.

Radev, S. T., Mertens, U. K., Voss, A., Ardizzone, L., & Köthe, U. (2020). Bayesflow: Learning complex stochastic models with invertible neural networks. *IEEE transactions on neural networks and learning systems*.

Ramesh, P., Lueckmann, J.-M., Boelts, J., Tejero-Cantero, Á., Greenberg, D. S., Gonçalves, P. J., & Macke, J. H. (2022). Gatsbi: Generative adversarial training for simulation-based inference. *arXiv preprint arXiv:2203.06481*.

Ratcliff, R., & McKoon, G. (2008). The diffusion decision model: Theory and data for two-choice decision tasks. *Neural Computation*, *20*(4), 873–922.

Schmitt, M., Bürkner, P.-C., Köthe, U., & Radev, S. T. (2021). Detecting model misspecification in amortized bayesian inference with neural networks. *arXiv preprint*.

Voss, A., & Voss, J. (2007). Fast-dm: A free program for efficient diffusion model analysis. *Behavior research methods*, *39*(4), 767–775.

Wiqvist, S., Frellsen, J., & Picchini, U. (2021). Sequential neural posterior and likelihood approximation. *arXiv preprint*.