Scoring Rule Training for Simulation-Based Inference

Anonymous authors
Paper under double-blind review

Abstract

Bayesian Simulation-Based Inference yields posterior approximations for simulator models with intractable likelihood. Recently, normalizing flows were used to approximate either the likelihood or the posterior directly. Normalizing flows are invertible neural networks which transform samples from a latent distribution; the probability density of the transformed samples is accessible and the normalizing flow can be trained via maximum likelihood on simulated parameter-observation pairs. In contrast, Ramesh et al. (2022) approximated the posterior with generative networks, which drop invertibility and are thus more flexible, hence scaling to high-dimensional and structured data. As generative networks merely allow sampling from the parametrized distribution, Ramesh et al. (2022) exploited adversarial training, where the generative network plays a game against a "discriminator" network. This procedure is unstable and can lead to a learned distribution underestimating the uncertainty. Here, we apply Scoring Rule minimization, an adversarial-free training approach for generative networks, to approximate the posterior for simulator models. We term the resulting method ScoRuTSBI (Scoring Rule Training for Simulation-Based Inference). On high-dimensional examples, we found ScoRuTSBI to perform better with shorter training time than the adversarial framework, which was previously found to outperform traditional methods (such as Approximate Bayesian Computation) and normalizing-flow-based inference methods.

1 Introduction

Simulator models are statistical models for which it is impossible to evaluate the likelihood $p(\mathbf{y}|\boldsymbol{\theta})$ for an observation $\mathbf{y}$, but from which it is easy to simulate for any parameter value $\boldsymbol{\theta}$. Given $\mathbf{y}$ and a prior $\pi(\boldsymbol{\theta})$, the Bayesian posterior is $\pi(\boldsymbol{\theta}|\mathbf{y}) \propto \pi(\boldsymbol{\theta})p(\mathbf{y}|\boldsymbol{\theta})$. However, obtaining that explicitly or sampling from it with Markov Chain Monte Carlo (MCMC) is impossible without having access to the likelihood.

Bayesian Simulation-Based Inference (SBI) techniques exploit model simulations to approximate the exact posterior distribution when the likelihood is unavailable. A prototypical method is Approximate Bayesian Computation (ABC) (Lintusaari et al., 2017; Bernton et al., 2019), which builds an implicit approximation of the posterior by drawing parameter values from the prior and weighting them according to the distance between observations and simulations obtained from those parameter values. Variants of ABC relying on MCMC (Marjoram et al., 2003) and sequential Monte Carlo (Beaumont et al., 2009) also exist.

A recent strand of literature has performed SBI using neural networks representing probability densities, such as generative networks trained adversarially (Goodfellow et al., 2014), normalizing flows (Papamakarios et al., 2021). Normalizing flows represent a distribution with an invertible neural network transforming samples from a simple base measure, thus allowing evaluating the density of the distribution via the change-of-variables formula enabled by invertibility. Using the latter, normalizing flows can be trained via maximum likelihood estimation on parameter-simulation pairs. In SBI, they have been used to represent either the likelihood (Papamakarios et al., 2019; Lueckmann et al., 2019) or the posterior (Papamakarios & Murray, 2016; Lueckmann et al., 2017; Greenberg et al., 2019; Radev et al., 2020) or both (Wiqvist et al., 2021).

Despite being present on the deep learning scene for longer than normalizing flows or diffusion models, generative networks have only been used for SBI in GATSBI (Ramesh et al., 2022), which adapted the

Generative Adversarial Network, or GAN, framework of Goodfellow et al. (2014) to train a generative network to learn the posterior. Generative networks can be thought of as normalizing flows dropping the invertibility requirement: by doing so, they gain in expressiveness and ability to scale to larger input and output sizes, but forgo density evaluation (from a generative network, you can only obtain draws from the parametrized probability distribution). As such, GAN recurs to training the generative network in a min-max game against an additional *discriminator* network aiming at distinguishing between training samples and simulations from the generative network. Empirically, Ramesh et al. (2022) showed how generative networks outperform normalizing flows on high-dimensional and structured data, but lead to generally poor calibration of the learnt distribution, which is a well-known consequence of unstable adversarial training (Arora et al., 2017; Bellemare et al., 2017; Arora et al., 2018; Richardson & Weiss, 2018).

Here, we show how the performance of generative networks for SBI can be improved by training them by Scoring Rule (SR, Gneiting & Raftery, 2007) minimization. Despite the name similarity, this is different from score-based diffusion networks (Song & Ermon, 2019; Song et al., 2020). SR minimization has been sparsely used before for training generative networks (Bouchacourt et al., 2016; Gritsenko et al., 2020; Harakeh & Waslander, 2021; Pacchiardi et al., 2022) but its application to Bayesian SBI is novel. We term this method Scoring Rule Training for Simulation-Based Inference (ScoRuTSBI). SR training is non-adversarial, thus leading to smoother (and faster) training and better calibration of the learnt distribution.

The rest of the paper is organized as follows. Section 2 discusses how to use a generative network to represent an approximate posterior. Section 3 introduces Scoring Rule Training for SBI (ScoRuTSBI) and discusses related approaches, including GATSBI (Ramesh et al., 2022) and normalizing flow methods. Section 4 reports simulation results and Section 5 gives concluding remarks.

Notation We will denote respectively by $\mathcal{Y} \subseteq \mathbb{R}^d$ and $\Theta \subseteq \mathbb{R}^p$ the data and parameter space. We will use $P(\cdot|\boldsymbol{\theta})$ and $p(\cdot|\boldsymbol{\theta})$ to denote the distribution and likelihood (with respect to Lebesgue measure) of the considered simulation-based model. Π and π will denote prior distribution and prior density on Θ, and $\Pi(\cdot|\mathbf{y})$ and $\pi(\cdot|\mathbf{y})$ will denote corresponding posterior quantities for observation $\mathbf{y}$. In general, we will use P or Q to denote distributions, while S will denote a generic Scoring Rule. Other upper-case letters ($\mathbf{X}, \mathbf{Y}$ and $\mathbf{Z}$) will denote random variables while lower-case ones will denote observed (fixed) values. We will denote by $\mathbf{Y}$ or $\mathbf{y}$ the observations (correspondingly random variables and realizations). Bold symbols denote vectors, and subscripts to bold symbols denote sample index (for instance, $\mathbf{y}_i$). Instead, subscripts to normal symbols denote component indices (for instance, y_j is the j-th component of $\mathbf{y}$, and $y_{i,j}$ is the j-th component of $\mathbf{y}_i$). Finally, $\perp\!\!\!\perp$ will denote independence between random variables, while $\mathbf{Y} \sim P$ indicates a random variable distributed according to P and $\mathbf{y} \sim P$ a sample from such random variable.

2 Approximate posterior via generative network

We use a generative network to represent an approximate posterior distribution $Q_\phi(\cdot|\mathbf{y})$ on the parameter space Θ given an observation $\mathbf{y} \in \mathcal{Y}$. The density of $Q_\phi(\cdot|\mathbf{y})$ (with respect to the Lebesgue measure) will be denoted by $q_\phi(\cdot|\mathbf{y})$. The generative network is defined via a neural network $g_\phi : \mathcal{Z} \times \mathcal{Y} \to \Theta$ transforming samples from a probability distribution $P_\mathbf{z}$ over the space $\mathcal{Z}$ conditionally on an observation $\mathbf{y} \in \mathcal{Y}$; ϕ represents neural network weights. Samples from $Q_\phi(\cdot|\mathbf{y})$ are therefore obtained by sampling $\mathbf{z} \sim P_\mathbf{z}$ and computing $\tilde{\boldsymbol{\theta}} = g_\phi(\mathbf{z}, \mathbf{y}) \sim Q_\phi(\cdot|\mathbf{y})$[1]

In the following, as it is standard in the SBI setup, we assume to have access to parameter-simulations pairs $(\boldsymbol{\theta}_i, \mathbf{y}_i)_{i=1}^n$ generated from the prior $\boldsymbol{\theta}_i \sim \Pi$ and the model $\mathbf{y}_i \sim P(\cdot|\boldsymbol{\theta}_i)$; critically, these can also be considered as samples from the data marginal $\mathbf{y}_i \sim P$ and the posterior $\boldsymbol{\theta}_i \sim \Pi(\cdot|\mathbf{y}_i)$. Using these samples, we want to tune ϕ such that $Q_\phi(\cdot|\mathbf{y}) \approx \Pi(\cdot|\mathbf{y})$ for all values of $\mathbf{y}$; this is therefore an *amortized* setting (Radev et al., 2020), namely the resulting posterior approximation is valid for multiple observations.

In the amortized setting, a single neural network has to map the observation into a posterior for all possible observations; intuitively, we expect this to work well for those cases where such inversion process is in some

---

[1] Formally, $Q_\phi(\cdot|\mathbf{y})$ is the push-forward of $P_\mathbf{z}$ through the map $g_\phi(\cdot, \mathbf{y})$: $Q_\phi(\cdot|\mathbf{y}) = g_\phi(\cdot, \mathbf{y})\sharp P_\mathbf{z}$, which means that, for any set A belonging to the Borel σ-algebra $\sigma(\Theta)$, $Q_\phi(A|\mathbf{y}) = P_\mathbf{z}\left(\{\mathbf{z} \in \mathbf{Z} : g_\phi(\mathbf{z}, \mathbf{y}) \in A\}\right)$.

sense “generic”. In contrast, the amortized approach will perform poorly when the posterior distribution depends on the data in a non-linear way. Additionally, the amortized approach may be wasteful in terms of model simulations when inference for a single observation is needed, as the simulations from the simulation-based model are drawn independently from it, so that many will be uninformative. In Appendix A, we discuss strategies for tailoring simulations to a specific observation.

3 Posterior inference via Scoring Rule training

We first review Scoring Rules and give examples OF Scoring Rules that we’ll use in our framework (Sec. 3.1). We then discuss in detail our proposed training method (Sec. 3.2) and related approaches (Sec. 3.2).

3.1 Scoring Rules

We first introduce Scoring Rules for a distribution P related to a generic random variable $\mathbf{X}$. A Scoring Rule (SR, Gneiting & Raftery, 2007) $S(P, \mathbf{x})$ is a function of P and of an observation $\mathbf{x}$ of the random variable $\mathbf{X}$. If $\mathbf{X}$ is distributed according to Q, the expected Scoring Rule is defined as:

$$S(P, Q) := \mathbb{E}_{\mathbf{X}\sim Q} S(P, \mathbf{X}),$$

The Scoring Rule S is *proper* relative to a set of distributions $\mathcal{P}$ over $\mathcal{X}$ if

$$S(Q, Q) \leq S(P, Q) \ \forall \ P, Q \in \mathcal{P},$$

i.e., if the expected Scoring Rule is minimized in P when $P = Q$. Moreover, S is *strictly proper* relative to $\mathcal{P}$ if $P = Q$ is the unique minimum:

$$S(Q, Q) < S(P, Q) \ \forall \ P, Q \in \mathcal{P} \text{ s.t. } P \neq Q.$$

3.1.1 Examples of Scoring Rules

The following are strictly proper Scoring Rules that we will use in our experiments.

Energy score The energy score[2] is given by:

$$S_{\mathrm{E}}^{(\beta)}(P, \mathbf{x}) = 2 \cdot \mathbb{E}\left[\|\tilde{\mathbf{X}} - \mathbf{x}\|_2^\beta\right] - \mathbb{E}\left[\|\tilde{\mathbf{X}} - \tilde{\mathbf{X}}'\|_2^\beta\right], \quad \tilde{\mathbf{X}} \perp\!\!\!\perp \tilde{\mathbf{X}}' \sim P, \tag{1}$$

where $\beta \in (0, 2)$. This is a strictly proper SR for the class of probability measures P such that $\mathbb{E}_{\tilde{\mathbf{X}}\sim P}\|\tilde{\mathbf{X}}\|^\beta < \infty$ (Gneiting & Raftery, 2007). An unbiased estimate can be obtained by replacing the expectations in $S_{\mathrm{E}}^{(\beta)}$ with empirical means over draws from P (see Appendix C.1) We will fix $\beta = 1$ in the rest of this work.

Kernel score When $k(\cdot, \cdot)$ is a positive definite kernel, the kernel score for k can be defined as (Gneiting & Raftery, 2007):

$$S_k(P, \mathbf{x}) = \mathbb{E}[k(\tilde{\mathbf{X}}, \tilde{\mathbf{X}}')] - 2 \cdot \mathbb{E}[k(\tilde{\mathbf{X}}, \mathbf{x})], \quad \tilde{\mathbf{X}} \perp\!\!\!\perp \tilde{\mathbf{X}}' \sim P. \tag{2}$$

The kernel score is proper for the class of probability distributions P for which $\mathbb{E}_{\tilde{\mathbf{X}}, \tilde{\mathbf{X}}'\sim P}[k(\tilde{\mathbf{X}}, \tilde{\mathbf{X}}')]$ is finite (by Theorem 4 in Gneiting & Raftery (2007)). It is closely related to the kernel Maximum Mean Discrepancy (MMD, Gretton et al., 2012) and is strictly proper under conditions ensuring the MMD is a metric (Gretton et al., 2012). These conditions are satisfied, among others, by the Gaussian kernel (which we will use in this work):

$$k(\tilde{\mathbf{x}}, \mathbf{x}) = \exp\left(-\frac{\|\tilde{\mathbf{x}} - \mathbf{x}\|_2^2}{2\gamma^2}\right),$$

in which γ is a scalar bandwidth. As for the Energy Score, an unbiased estimate can be obtained by replacing the expectations in S_k with empirical means over draws from P (see Appendix C.1).

---

[2] The probabilistic forecasting literature (Gneiting & Raftery, 2007) use a different convention for the energy score and the subsequent kernel score, which amounts to multiplying our definitions by 1/2. We follow here the convention used in the statistical inference literature (Rizzo & Székely, 2016; Chérief-Abdellatif & Alquier, 2020; Nguyen et al., 2020)

Patched SR We now discuss a way to build a composite SR which encodes structural information in $\mathbf{X}$. In fact, if $\mathbf{X}$ has some structure (say, it is on a 1D or 2D grid), computing the raw SRs above discards that information. A way to encode some of it is to compute the SRs on localized *patches* across the grid and cumulate the score; in this way, short-scale correlations are given more importance. However, the resulting Scoring Rule is not strictly proper; to fix this, we add the SR computed over the full $\mathbf{x}$, which makes the overall SR strictly proper (see Lemma 3.4 in Pacchiardi et al. (2022)).

For a given SR S, therefore, the patched SR is:

$$S_p(P, \mathbf{x}) = w_1 S(P, \mathbf{x}) + w_2 \sum_{p \in \mathcal{P}} S(P|_p, \mathbf{x}|_p),$$

where $w_1, w_2 > 0$, $|_p$ denotes the restriction of a distribution or of a vector to a patch p and $\mathcal{P}$ is a set of patches. See Pacchiardi et al. (2022) for more discussion on patched SRs.

3.2 ScoRuTSBI: Scoring Rule Training for Simulation-Based Inference

Let us now go back to the Bayesian SBI setting introduced at the start of the paper and let us denote by $Q_\phi(\cdot|\mathbf{y})$ the approximate posterior parametrized by the generative network.

For a strictly proper SR S, solving the following problem:

$$\arg\min_\phi \mathbb{E}_{\mathbf{Y}\sim P} \mathbb{E}_{\boldsymbol{\theta}\sim\Pi(\cdot|\mathbf{Y})} S(Q_\phi(\cdot|\mathbf{Y}), \boldsymbol{\theta}) = \arg\min_\phi \mathbb{E}_{\boldsymbol{\theta}\sim\Pi} \mathbb{E}_{\mathbf{Y}\sim P(\cdot|\boldsymbol{\theta})} S(Q_\phi(\cdot|\mathbf{Y}), \boldsymbol{\theta}) \tag{3}$$

leads to $q_\phi(\cdot|\mathbf{y}) = \pi(\cdot|\mathbf{y})$ for all values of $\mathbf{y}$ for which $p(\mathbf{y}) > 0$.

An empirical analogue of the objective in Eq. (3) is obtained by replacing the expectations with empirical means over the training dataset, leading to the following empirical minimization problem:

$$\arg\min_\phi \frac{1}{n} \sum_{i=1}^n S(Q_\phi(\cdot|\mathbf{y}_i), \boldsymbol{\theta}_i); \tag{4}$$

computing the objective directly is intractable as, in general, we do not have access to $S(Q_\phi(\cdot|\mathbf{y}), \boldsymbol{\theta})$. Notice, however, that in order to solve Eq. (4) via Stochastic Gradient Descent (SGD) it is enough to obtain unbiased estimates of $\nabla_\phi S(Q_\phi(\cdot|\mathbf{y}_i), \boldsymbol{\theta}_i)$, which can be easily done via automatic differentiation whenever S admits an unbiased empirical estimator $\hat{S}$ such that:

$$\mathbb{E}\left[\hat{S}(\{\tilde{\boldsymbol{\theta}}_j^{(\mathbf{y})}\}_{j=1}^m, \boldsymbol{\theta})\right] = S(Q_\phi(\cdot|\mathbf{y}), \boldsymbol{\theta}),$$

where the expectation is over $\tilde{\boldsymbol{\theta}}_j^{(\mathbf{y})} \sim Q_\phi(\cdot|\mathbf{y})$. More details can be found in Appendix C.2. If S admits such an estimator, each step of SGD involves generating m simulations from the generative network $Q_\phi(\cdot|\mathbf{y}_i)$ for each $\mathbf{y}_i$ in the training batch. We term this approach ScoRuTSBI (Scoring Rule Training for Simulation-Based Inference). Algorithm 1 shows a single epoch of the resulting training algorithm, with batch size equal to 1 for simplicity. For the Energy and Kernel Scores introduced in Sec. 3.1.1, unbiased estimators are available; as such, we will use these scoring rules in the following.

Algorithm 1 ScoRuTSBI, single epoch (with batch size equal to 1).

```
Require: Generative network $g_\phi : \mathcal{Z} \times \mathcal{Y} \to \Theta$, SR $S$, learning rate $\epsilon$.
  for each training pair $(\boldsymbol{\theta}_i, \mathbf{y}_i)$ do
      Sample multiple $\mathbf{z}_1, \dots, \mathbf{z}_m$
      Obtain $\tilde{\boldsymbol{\theta}}_{i,j}^{\phi} = g_\phi(\mathbf{z}_j, \mathbf{y}_i)$
      Obtain unbiased estimate $\hat{S}(\{\tilde{\boldsymbol{\theta}}_{i,j}^{(\mathbf{y}_i)}\}_{j=1}^m, \boldsymbol{\theta}_i)$ of $S(Q_\phi(\cdot|\mathbf{y}_i), \boldsymbol{\theta}_i)$
      Set $\phi \leftarrow \phi - \epsilon \cdot \nabla_\phi \hat{S}(\{\tilde{\boldsymbol{\theta}}_{i,j}^{(\mathbf{y}_i)}\}_{j=1}^m, \boldsymbol{\theta}_i)$
  end for
```

3.3 Related approaches

3.3.1 Generative Adversarial Training for Simulation-Based Inference

In Ramesh et al. (2022), the posterior approximation Q_ϕ was trained in an adversarial framework in a method termed GATSBI (Generative Adversarial Training for Simulation-Based Inference). This requires introducing a *discriminator* or *critic* neural network $c_\psi : \Theta \times \mathcal{Y} \to \mathbb{R}$ with weights ψ whose task is to distinguish draws from the approximate and true posteriors. The loss employed in Ramesh et al. (2022) is the conditional version of the original Generative Adversarial Network (GAN) loss from Goodfellow et al. (2014), which was originally discussed in Mirza & Osindero (2014):

$$\begin{aligned} L(\phi, \psi) &= \mathbb{E}_{\boldsymbol{\theta} \sim \Pi} \mathbb{E}_{\mathbf{Y} \sim P(\cdot|\boldsymbol{\theta})} \mathbb{E}_{\mathbf{Z} \sim P_{\mathbf{z}}} \left[\log c_\psi(\boldsymbol{\theta}, \mathbf{Y}) + \log \left(1 - c_\psi \left(g_\phi(\mathbf{Z}, \mathbf{Y}), \mathbf{Y}\right)\right) \right] \\ &= \mathbb{E}_{\mathbf{Y} \sim P} \left[\mathbb{E}_{\boldsymbol{\theta} \sim \Pi(\cdot|\mathbf{Y})} \left(\log c_\psi(\boldsymbol{\theta}, \mathbf{Y}) \right) + \mathbb{E}_{\tilde{\boldsymbol{\theta}} \sim Q_\phi(\cdot|\mathbf{Y})} \left(\log \left(1 - c_\psi(\tilde{\boldsymbol{\theta}}, \mathbf{Y})\right)\right) \right], \end{aligned} \tag{5}$$

whose saddle point solution

$$\min_\phi \max_\psi L(\phi, \psi) \tag{6}$$

leads to $Q_\phi(\cdot|\mathbf{y})$ being the exact posterior for all choices of $\mathbf{y}$ for which $p(\mathbf{y}) > 0$ (provided q_ϕ and c_ψ have infinite expressive power; that in fact corresponds to minimizing the Jensen-Shannon divergence, see Appendix B for more details).

As typically done in GANs, GATSBI trains Q_ϕ alternating maximization steps over ψ with minimization steps over ϕ. At each step, the gradient is estimated by replacing the expectations in Eq. (5) with empirical means over (a mini-batch of) the training dataset and draws from the generative network. This alternating optimization is unstable and requires careful hyperparameters tuning and specialized training routines (Salimans et al., 2016). Despite those, adversarial training often leads to underestimating the distribution width, and even to collapse of the distribution on a single mode (Arora et al., 2018; Isola et al., 2017; Richardson & Weiss, 2018). Arora et al. (2017) showed how this mode collapse can happen due to the finite capacity of the discriminator, while Bellemare et al. (2017) theoretically linked it to the use of biased gradient estimates for ϕ in optimizing Eq. (6) (in fact, estimates of gradients with respect to ϕ rely on a value of ψ obtained by few optimization steps, rather than the value maximizing eq. 5).

This uncertainty underestimation may not be an issue in some applications of generative networks where uncertainty quantification is not important, but it can be detrimental for approximate posterior inference. In contrast, our Scoring Rule minimization formulation for SBI does not suffer from these theoretical issues introduced by the use of alternating minimization. Indeed, we show in Sec. 4 how the approximate posterior obtained with Scoring Rule minimization has better calibration than GATSBI (Section 4).

On the other hand, the unbiased empirical estimators of the Energy and Kernel Scores require multiple draws from the generative network per observation value (Eq. 3.2). To train GAN, instead, a single draw from the generative network is enough. In our experiments, however, 10 or fewer draws lead to satisfactory results with SR training. Additionally, as mentioned above, the SR approach does not require a discriminator network and has a more stable training process, which implies convergence is generally reached with fewer training epochs. These two factors lead to lower computational and memory cost with respect to adversarial training (see Section 4 for details).

3.3.2 Normalizing flows for SBI

Normalizing flows are generative networks which impose invertibility of the map $g_\phi(\mathbf{z}, \mathbf{y})$ with respect to $\mathbf{z}$. As such, density evaluation of the resulting q_ϕ is possible via the change-of-variables formula, so that ϕ is usually trained via maximum likelihood (Papamakarios et al., 2021). For instance, in Radev et al. (2020), the following problem was considered, where D_{KL} denotes the Kullback-Leibler divergence:

$$\begin{aligned}
&\underset{\boldsymbol{\phi}}{\operatorname{argmin}}\ \mathbb{E}_{\mathbf{Y}\sim P}\left[D_{KL}\left(\Pi(\cdot\mid\mathbf{Y})\|Q_{\boldsymbol{\phi}}(\cdot\mid\mathbf{Y})\right)\right]\\
=&\underset{\phi}{\operatorname{argmin}}\mathbb{E}_{\mathbf{Y}\sim P}\mathbb{E}_{\boldsymbol{\theta}\sim\Pi(\cdot|\mathbf{Y})}\left[-\log q_{\boldsymbol{\phi}}(\boldsymbol{\theta}\mid\mathbf{Y})\right]\\
=&\underset{\phi}{\operatorname{argmin}}\mathbb{E}_{\boldsymbol{\theta}\sim\Pi}\mathbb{E}_{\mathbf{Y}\sim P(\cdot|\boldsymbol{\theta})}\left[-\log q_{\boldsymbol{\phi}}(\boldsymbol{\theta}\mid\mathbf{Y})\right],
\end{aligned}\tag{7}$$

which corresponds to our SR-based approach in Eq. (3) by identifying $S(Q_\phi(\cdot|\mathbf{y}),\boldsymbol{\theta})=-\log q_\phi(\boldsymbol{\theta}|\mathbf{y})$, which is the strictly-proper logarithmic scoring rule (Gneiting & Raftery, 2007).

The Neural Posterior Estimation (NPE) methods presented in Papamakarios & Murray (2016); Lueckmann et al. (2017); Greenberg et al. (2019) are closely related to the objective in Eq. 7, but they focus on inferring a posterior distribution valid for a single observation and design sequential approaches to exploit simulations more efficiently. A single turn of those methods correspond exactly to Eq. (7).

Differently, Neural Likelihood Estimation (NLE) Papamakarios et al. (2019); Lueckmann et al. (2019) targets the likelihood instead of the posterior. Similarly, it is a sequential approach tailoring simulations to a single observation, but it can also be used in a single-turn fashion.

Sequential versions of our SR minimization approach can also be designed, see Appendix A.

3.4 Other uses of SR training

SR training has been used before for training generative networks: Bouchacourt et al. (2016), Gritsenko et al. (2020) and Harakeh & Waslander (2021) all used the Energy Score, focusing respectively on the tasks of hand pose estimation, speech synthesis and object estimation. With the exception of the latter, these works only exploited SR training as it lead to better generated samples, but not for its probabilistic performance. Pacchiardi et al. (2022) used SR training for the task of probabilistic forecasting, deriving theoretical guarantees for a predictive-sequential (Dawid, 1984) training objective. Finally, subsequently to the first version of this work, Bon et al. (2022) uses an objective similar to our Eq. (3) to calibrate an approximate posterior obtained with other approaches.

4 Simulation studies

We present here results on two low-dimensional benchmark problems and two high-dimensional models, one of which has an implicitly defined prior, which were studied in Ramesh et al. (2022). For all examples, we evaluate the performance of the different methods as in Ramesh et al. (2022). Besides that, we assess the calibration of the approximate posteriors by the discrepancy between credible intervals in the approximate posteriors and the frequency with which the true parameter belongs to the credible interval itself (we call this metric *calibration error*). We also evaluate how close the posterior means are to the true parameter value by the *Normalized Root Mean-Square Error* (NRMSE) and the *coefficient of determination* R^2; these metrics were used for LFI in Radev et al. (2020); we provide more detail in Appendix D. As all these metrics are for scalar variables, we compute their values independently for each component of $\boldsymbol{\theta}$ and report their average.

For the two low-dimensional benchmarks, Ramesh et al. (2022) compared GASTBI with NPE (Papamakarios & Murray, 2016; Lueckmann et al., 2017; Greenberg et al., 2019), NLE (Papamakarios et al., 2019; Lueckmann et al., 2019) , Neural Ratio Estimation (Hermans et al., 2020) and two versions of Approximate Bayesian Computation (Tavaré et al., 1997; Beaumont et al., 2009). They found that normalizing flows methods perform better than GATSBI in these cases. We find that ScoRuTSBI performs comparably to GATSBI here, so we report a detailed comparison of the performances of these two methods, referring the reader to Ramesh et al. (2022) for the performance of the other methods.

Generative neural networks have a competitive advantage in high-dimensional settings; indeed, Ramesh et al. (2022) found that GATSBI was competitive with normalizing-flow methods on one of the two high-dimensional cases (ABC methods, not being amortized, could not be run over the large number of observations in a reasonable amount of time). As such, we here report results of NPE, NLE, GATSBI and

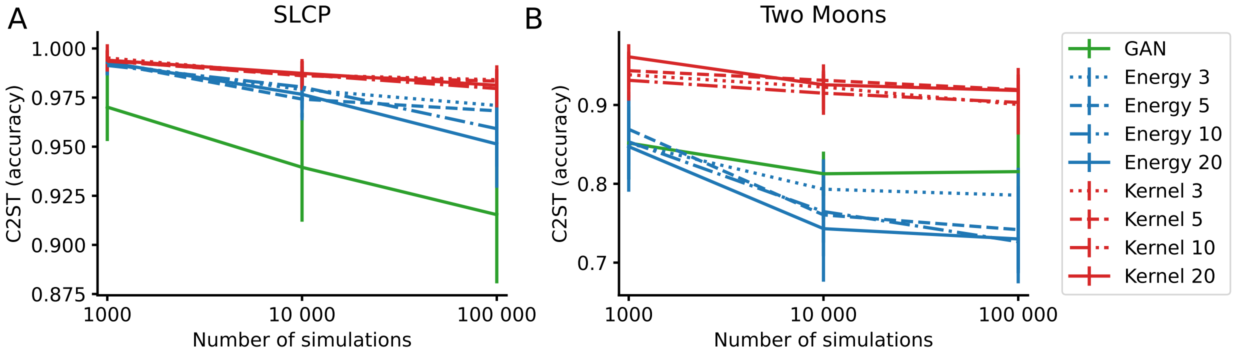


Figure 1: C2ST for ScoRuTSBI and GATSBI for the SLCP and Two Moons benchmarks; larger values are worse. For ScoRuTSBI, we report results for different choices of the number of generative network samples m used in training. SLCP: GATSBI performs better, but poorly on an absolute scale. Two Moons: ScoRuTSBI with the Energy Score perform better.

ScoRuTSBI for that example (Sec. 4.2). Finally, GATSBI and ScoRuTSBI are the only method that can handle the implicit prior of the other high-dimensional example (Sec. 4.3).

Additional training details of our approach for all examples are reported in Appendix E. We refer to Ramesh et al. (2022) for details of the other methods.

4.1 Benchmark models

We consider here the "Simple Likelihood Complex Posterior" (SLCP) and the "Two Moons" benchmarks; in the former, a 5-dimensional $\boldsymbol{\theta}$ defines the distribution of an 8-dimensional Gaussian $\mathbf{y}$ in a nonlinear manner. In the Two Moons model, both $\mathbf{y}$ and $\boldsymbol{\theta}$ are 2-dimensional. We refer to Ramesh et al. (2022) and references therein for more details[3] For both models, we train all methods on $n_{\text{train}} = 1000, 10000$ and 100000 posterior samples. We consider ScoRuTSBI with the Energy and Kernel Score trained with $m = 3, 5, 10$ or 20 samples from the generative network for each $\mathbf{y}_i$ in a training batch. ScoRuTSBI is trained on a single CPU, while GATSBI is trained on an NVIDIA Tesla-V100 GPU. For the Two Moons model, we do not use early stopping for ScoRuTSBI; additionally, we employ the optimal configuration found in Ramesh et al. (2022) for GAN.

For these two models, samples from reference posteriors are available (Lueckmann et al., 2021); therefore, as done in Ramesh et al. (2022), we assess the performance of the different methods via the discrimination ability of a classifier trained to distinguish samples from the reference and approximate posteriors (classification-based two-sample test, C2ST). If the classification accuracy is 0.5, the classifier is unable to distinguish between the two sets of samples, implying perfect posterior approximation.

In Figure 1, we report C2ST values for GATSBI and ScoRuTSBI for the different number of training simulations. For SLCP, GATSBI performs better (although the performance is poor on an absolute scale and worse than other SBI methods, see Ramesh et al., 2022). For the Two Moons example, ScoRuTSBI based on the Energy Score performs better.

In Tables 1 and 2, we report other performance metrics, together with the runtime and the epoch at which training was early stopped, for GATSBI and ScoRuTSBI, with $n_{\text{train}} = 100000$ and $m = 20$. Notice how ScoRuTSBI was trained in much shorter time (and on a single CPU). Additional results are reported in Appendices F.1 and F.2.

---

[3] These models are implemented in the `sbibm` Python package, whose accompanying paper (Lueckmann et al., 2021) provides additional details.

Table 1: SLCP: performance metrics, runtime and early stopping epoch for GATSBI, ScoRuTSBI (with Energy and Kernel Score), with $n_{\text{train}} = 100000$ and $m = 20$. Notice how ScoRuTSBI was trained on a single CPU, while GAN was trained on a GPU. The maximum number of training epochs was 20000.

	C2ST ↓	NRMSE ↓	Cal. Err. ↓	R^2 ↑	Runtime (sec)	Early stopping epoch
GAN	0.92 ± 0.03	0.23 ± 0.05	0.06 ± 0.03	0.35 ± 0.30	30963	20000
Energy	0.95 ± 0.02	0.22 ± 0.06	0.07 ± 0.04	0.38 ± 0.32	1645	2100
Kernel	0.98 ± 0.01	0.22 ± 0.06	0.13 ± 0.10	0.37 ± 0.31	1210	1200

Table 2: Two Moons: performance metrics, runtime and early stopping epoch for GAN, Energy and Kernel Score methods, with $n_{\text{train}} = 100000$ and $m = 20$. Notice how ScoRuTSBI was trained on a single CPU, while GAN was trained on a GPU. Here, no early stopping was used (the maximum number of training epochs was 20000).

	C2ST ↓	NRMSE ↓	Cal. Err. ↓	R^2 ↑	Runtime (sec)	Early stopping epoch
GAN	0.82 ± 0.07	0.20 ± 0.00	0.07 ± 0.02	0.51 ± 0.01	30232	20000
Energy	0.73 ± 0.04	0.20 ± 0.00	0.03 ± 0.00	0.51 ± 0.01	10805	20000
Kernel	0.92 ± 0.02	0.20 ± 0.00	0.03 ± 0.01	0.50 ± 0.01	10902	20000

4.2 Shallow water model

The shallow water model is obtained as the discretization of a PDE describing the propagation of an initial disturbance across the surface of a 1D shallow basin; the parameter $\boldsymbol{\theta} \in \mathbb{R}^{100}$ represents the depth of the basin at equidistant points; the simulator outputs the evolution over 100 time-steps (producing a raw observation of size $100 \times 100 = 10000$); then, a Fourier transform is computed and the real and imaginary parts are concatenated and summed to Gaussian noise, leading to $\mathbf{y} \in \mathbb{R}^{20k}$. More details are given in Ramesh et al. (2022). We test here ScoRuTSBI with the Energy and Kernel score with $m = 10$ computed in three different configurations: 1) on the full parameter space, 2) with patch size 10 and step 5, and 3) with patch size 20 and step 10. Training is done on 100k samples on a NVIDIA Tesla-V100 GPU; additional details are discussed in Appendix E.1. Among the different instances of ScoRuTSBI, the Energy Score with patch size 20 and step 10 performed better; therefore, we report only results for that method in the main body of the paper; results for the other configurations are given in Appendix F.3. We compare with the results obtained by GATSBI, NPE and NLE.

In Figure 2, we report posterior and posterior predictive samples for all methods, together with prior samples and the ground-truth depth profile. For ScoRuTSBI and NPE, posterior samples better follow the ground truth profile and, similarly, posterior predictive samples better match the true observation.

In Table 3, we report the performance metrics, runtime and epoch of early stopping of the GATSBI and ScoRuTSBI; notice how the calibration error is much smaller for the latter, whose training run faster. We also assess calibration via Simulation Based Calibration (Talts et al., 2018, details in Appendix D.2.2) in Figure 3. That as well highlights how the calibration of ScoRuTSBI is better than the one achieved by GAN and comparable with that achieved by NPE.

4.3 Noisy Camera model

Here, we consider $\boldsymbol{\theta} \in \mathbb{R}^{28\times 28}$ to be the images of the EMNIST dataset (Cohen et al., 2017), from which the data $\mathbf{y} \in \mathbb{R}^{28\times 28}$ are generated by applying some blurring (see Ramesh et al., 2022 for details). Posterior inference corresponds therefore to Bayesian denoising. In this model, the dimension of parameter space is larger than in typical SBI applications; additionally, the prior is defined implicitly as we can only generate samples from it. This prevents the application of many standard SBI methods. Besides GATSBI, we test here ScoRuTSBI with the Energy and Kernel score with $m = 10$ in three different configurations: 1) on the

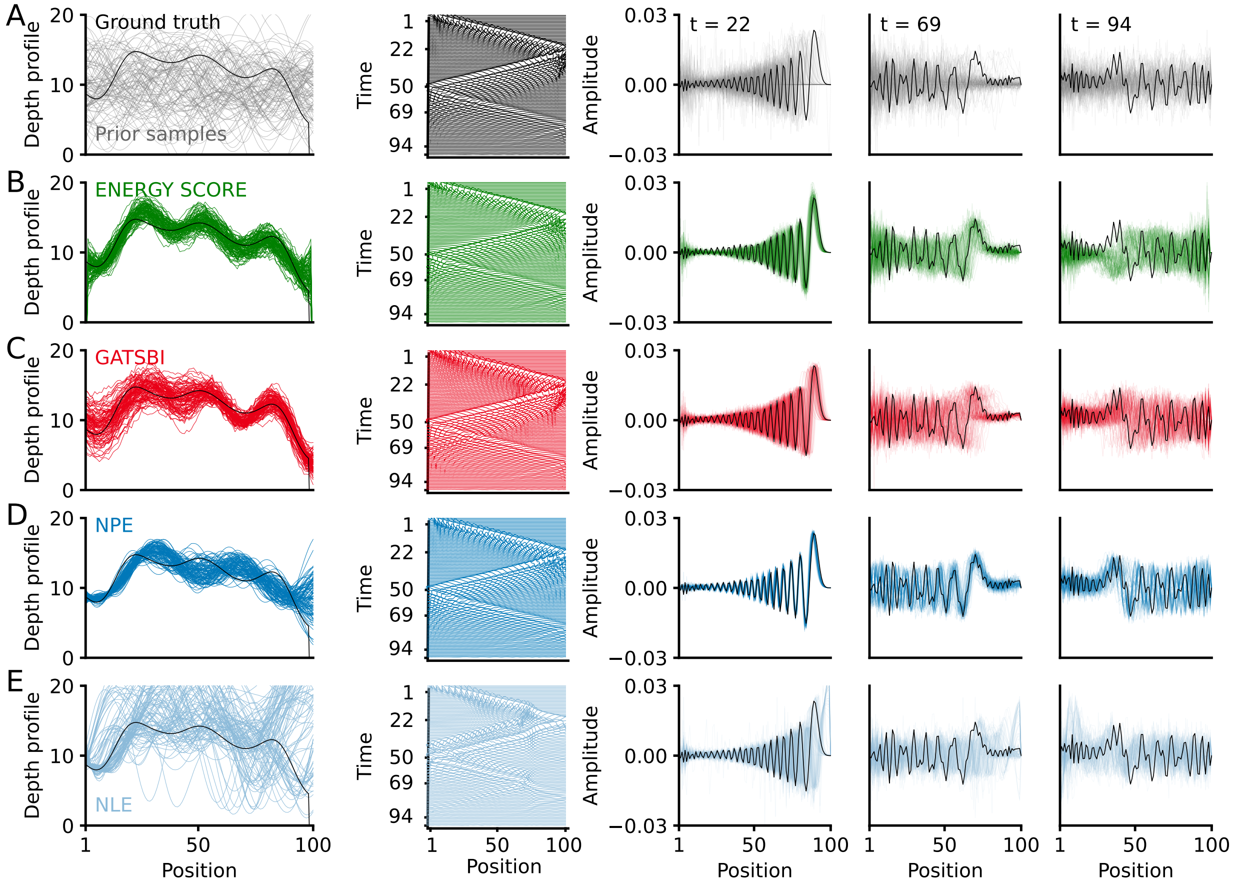


Figure 2: Shallow water model: inference results with GATSBI, NPE, NLE and ScoRuTSBI with Energy Score with patch size 20 and step 10. The figure structure closely follows that in Ramesh et al. (2022). Row A: Ground truth, observation and prior samples. Left: ground-truth depth profile and prior samples. Middle: surface wave simulated from ground-truth profile as a function of position and time. Right: wave amplitudes at three different fixed times for ground-truth depth profile (black), and waves simulated from multiple prior samples (gray). The remaining rows refer to ScoRuTSBI with the Energy Score (with patch size 20 and step 10), GATSBI, NPE AND NLE. For all methods, left represents posterior samples versus ground-truth (black) depth profiles, from which it can be seen how posterior samples for ScoRuTSBI better follow the truth with respect to GAN; middle represents surface wave simulated from a single posterior sample; right represents wave amplitudes simulated from multiple posterior samples, at three different fixed times, with black line denoting the actual observation; again, ScoRuTSBI better follows the observation, except for $t = 94$.

Table 3: Shallow Water model: performance metrics, runtime and early stopping epoch for GATSBI and ScoRuTSBI with the Energy Score with patch size 20 and step 10. The latter method achieved better results with shorter training time. We do not train GATSBI from scratch but rather relied on the trained network obtained in Ramesh et al. (2022). The training time we report here corresponds to what is mentioned in Ramesh et al. (2022), which used two GPUs for training (with respect to a single one for ScoRuTSBI). For the same reason, we do not report the epoch at which GAN training was early stopped.

	NRMSE ↓	Cal. Err. ↓	R^2 ↑	Runtime (sec)	Early stopping epoch
Energy	0.05 ± 0.01	0.03 ± 0.02	0.89 ± 0.05	60017	12400
GAN	0.07 ± 0.01	0.12 ± 0.09	0.78 ± 0.05	≈345600	-

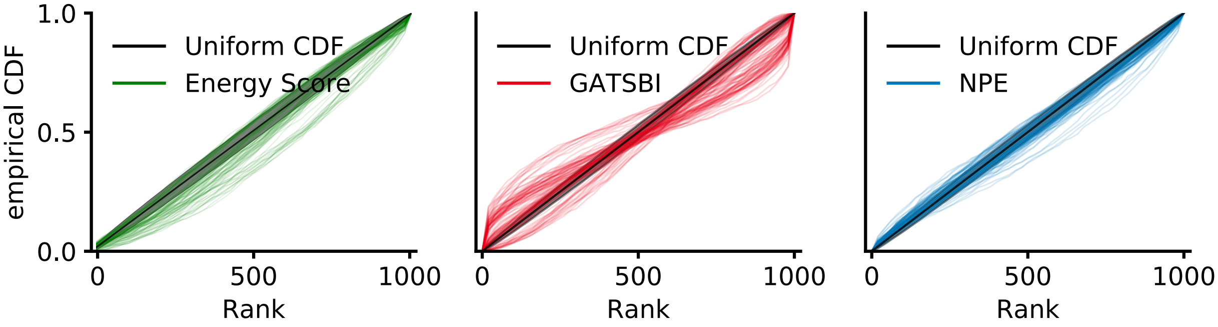


Figure 3: Shallow Water model: Simulation Based Calibration for ScoRuTSBI with Energy Score (patch size 20 and step 10), GATSBI and NPE. Each line corresponds to a single dimension of $\boldsymbol{\theta}$ and represents the CDF of the rank of the true parameter value with respect to a set of posterior samples. A calibrated posterior implies uniform CDF (diagonal black line, with associated 99% confidence region for that number of samples in gray).

full parameter space, 2) with patch size 14 and step 7, and 3) with patch size 8 and step 5. Training is done on 800 thousands samples on a NVIDIA Tesla-V100 GPU; additional details are discussed in Appendix E.2. Among the different instances of ScoRuTSBI, those with patch size 8 and step 5 performed better; therefore, we report only results for ScoRuTSBI with the Kernel and Energy Score in that configuration in the main body of the paper; results for the other configurations are given in Appendix F.4.

In Figure 4, we report posterior mean and standard deviation for a set of observations for the different methods. The two ScoRuTSBI variations lead to cleaner image reconstruction and more meaningful uncertainty quantification.

In Table 4, we report the performance metrics, runtime and epoch of early stopping of GATSBI and ScoRuTSBI with the two choices of Scoring Rule; the latter leads to smaller calibration error, although that is still quite poor in absolute terms. The R^2 values here are also poor. We believe these low metric values are due to each pixel only taking a discrete set of values between 0 and 1, with white spaces assigned 0 and darkest pixels being assigned 1. The generative network outputs is bounded in $(0, 1)$ as it is obtained via a continuous transformation from $\mathbb{R}$. For the calibration error (see Appendix D.2.1), that means that a credible interval obtained from the generative network cannot contain the extreme values 0 or 1; similarly, the approximate posterior mean can never be smaller than 0 or larger than 1, thus decreasing the R^2 values (see Appendix D.1.2). Additionally, we report here the un-normalized RMSE, as computing the normalization would lead to infinite values for the pixels in which the true value is 0 for all training samples (see Appendix D.1.1).

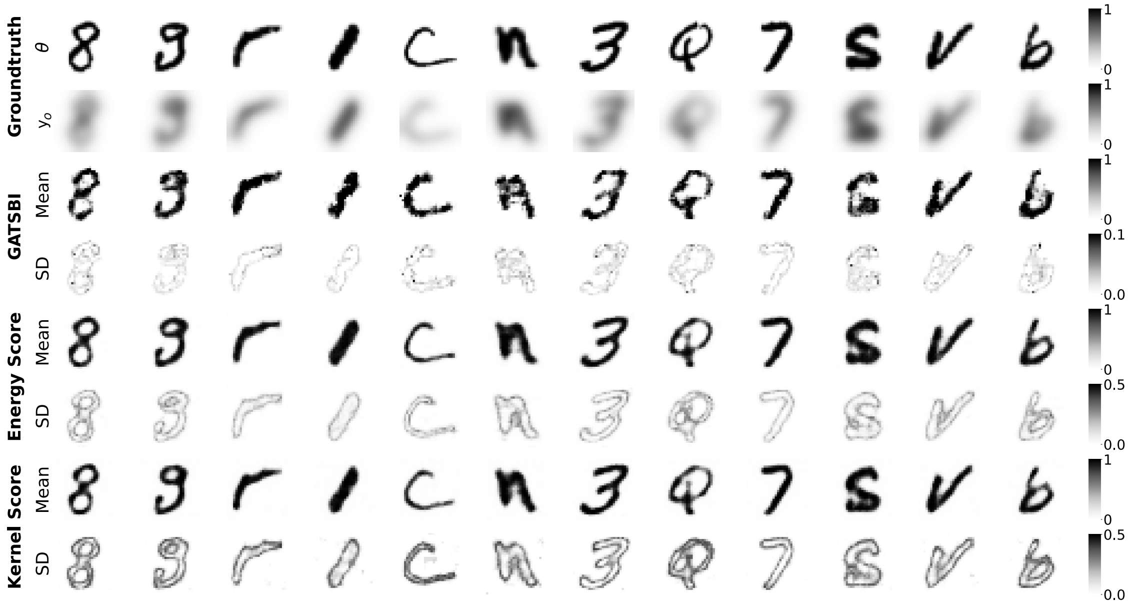


Figure 4: Noisy Camera model: ground truth and posterior inference with different methods, for a set of observations (each observation corresponds to a column). The first two rows represent the ground-truth values of $\boldsymbol{\theta}$ and the corresponding observation $\mathbf{y}_o$. The remaining rows represent mean and Standard Deviation (SD) for GATSBI and ScoRuTSBI with Energy and Kernel Score with patch size 8 and step 5. Notice how the posterior mean for ScoRuTSBI are neater than those obtained with GATSBI; additionally, the SD is larger close to the boundary of the reconstructed digit (notice the different color scale in the SD for GATSBI and for the SR methods).

Table 4: Noisy Camera model: performance metrics, runtime and early stopping epoch for GATSBI and ScoRuTSBI with Energy and Kernel Score (patch size 8 and step 5). ScoRuTSBI achieved better performance with shorter training time. All methods are trained on a single GPU.

	RMSE ↓	Cal. Err. ↓	R^2 ↑	Runtime (sec)	Early stopping epoch
GAN	0.25 ± 0.19	0.50 ± 0.00	-23.94 ± 366.08	45398	3600
Energy	0.06 ± 0.05	0.36 ± 0.12	-2.14 ± 55.86	22633	4000
Kernel	0.07 ± 0.05	0.36 ± 0.12	-10.29 ± 222.12	22545	3200

5 Conclusions

We considered using a generative network to represent posterior distributions for Bayesian Simulation-Based Inference and investigated training it via Scoring Rule minimization rather than in the adversarial setup of Ramesh et al. (2022). Our approach, termed Scoring Rule Training for Simulation-Based Inference (ScoRuTSBI) is theoretically grounded and does not suffer from training instability and biased gradients, as does the adversarial approach.

In low-dimensional benchmarks, ScoRuTSBI has comparable performance to the adversarial approach of Ramesh et al. (2022) (termed GATSBI), but both fall short when compared to their counterparts based on normalizing flows (for detailed results, refer to Ramesh et al., 2022). Thus, this disparity in performance likely stems from the inherent challenges faced by generative networks in low-dimensional spaces, rather than being attributed to any specific training algorithm. Therefore, we recommend the utilization of normalizing-flow-based techniques in such scenarios.

Furthermore, we are mainly interested in the performance of our method in high-dimensional settings, a challenging terrain for traditional simulation-based inference methods and normalizing-flow-based approaches alike. As documented by Ramesh et al. (2022), GATSBI exhibited superior performance when compared to normalizing-flow-based methods like Neural Posterior Estimation (Greenberg et al., 2019) and Neural Likelihood Estimation (Papamakarios et al., 2019) (both of which have demonstrated greater efficacy in high-dimensional contexts than related approaches, Lueckmann et al., 2021). Here, we found that ScoRuTSBI improves upon GATSBI while requiring a lower computational cost. Consequently, we believe that ScoRuTSBI holds substantial promise for addressing Bayesian simulation-based inference for high-dimensional models.

For ScoRuTSBI, employing patched scores (Sec. 3.1.1) leads to a small performance improvement over the vanilla ones on the high-dimensional examples (see Appendix F.3 and F.4). While we designed the patches to capture the data structure, the improvement we observe could simply be due to computing the Energy and Kernel scores on lower-dimensional objects. To disentangle these two effects, we could define scoring rules using a random subset of components of θ of the same size as the patches used above. We leave this for future work.

Analogously to the patched scores, it may be that employing a patched discriminator (Isola et al., 2017) improves results with GAN; however, we believe this would not completely close the performance gap, which is mostly due to the harder optimization objective in GAN. To this point, more advanced adversarial training algorithms than the original GAN objective (Goodfellow et al., 2014) may lead to better results; however, for probabilistic forecasting, the results in Pacchiardi et al. (2022) show Scoring Rule minimization to outperform state-of-the-art adversarial approach, while being cheaper and easier to train. We expect the same to hold for simulation-based inference.

In the present work, we did not provide any theoretical guarantees for ScoRuTSBI; it could be of interest to prove a generalization bound between the empirical (Eq. 4) and population (Eq. 3) objectives, or a consistency result for the minimizer of Eq. (4), similarly to what done for probabilistic forecasting in Pacchiardi et al. (2022) and for GATSBI in Wang & Ročková (2022). We leave these extensions for future work.

Finally, using a learned kernel to train a generative network via Kernel Score minimization could make the method more flexible. This could be done by learning the kernel adversarially (as in MMD GAN, Bińkowski et al., 2018), which however would break the ease of optimization which is the main advantage of the Scoring Rule approach. We hope that future research will address achieving both these goals together.

Acknowledgments

The authors thank Poornima Ramesh for helping to use the code used to create the results reported in Ramesh et al. (2022) and for providing additional result files.

References

Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pp. 214–223. PMLR, 2017.

Sanjeev Arora, Rong Ge, Yingyu Liang, Tengyu Ma, and Yi Zhang. Generalization and equilibrium in generative adversarial nets (GANs). In *International Conference on Machine Learning*, pp. 224–232. PMLR, 2017.

Sanjeev Arora, Andrej Risteski, and Yi Zhang. Do GANs learn the distribution? Some theory and empirics. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=BJehNfW0-`.

Mark A Beaumont, Jean-Marie Cornuet, Jean-Michel Marin, and Christian P Robert. Adaptive approximate bayesian computation. *Biometrika*, 96(4):983–990, 2009.

Marc G Bellemare, Ivo Danihelka, Will Dabney, Shakir Mohamed, Balaji Lakshminarayanan, Stephan Hoyer, and Rémi Munos. The Cramer distance as a solution to biased Wasserstein gradients. *arXiv preprint arXiv:1705.10743*, 2017.

Espen Bernton, Pierre E. Jacob, Mathieu Gerber, and Christian P. Robert. Approximate Bayesian computation with the Wasserstein distance. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 81(2):235–269, 2019. doi: https://doi.org/10.1111/rssb.12312. URL `https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/rssb.12312`.

Mikołaj Bińkowski, Danica J Sutherland, Michael Arbel, and Arthur Gretton. Demystifying MMD GANs. In *International Conference on Learning Representations*, 2018.

Joshua J Bon, David J Warne, David J Nott, and Christopher Drovandi. Bayesian score calibration for approximate models. *arXiv preprint arXiv:2211.05357*, 2022.

Diane Bouchacourt, Pawan K Mudigonda, and Sebastian Nowozin. DISCO nets: DISsimilarity COefficient networks. *Advances in Neural Information Processing Systems*, 29:352–360, 2016.

Badr-Eddine Chérief-Abdellatif and Pierre Alquier. MMD-Bayes: Robust Bayesian estimation via maximum mean discrepancy. In *Symposium on Advances in Approximate Bayesian Inference*, pp. 1–21. PMLR, 2020.

Jon Cockayne, Matthew M. Graham, Chris J. Oates, T. J. Sullivan, and Onur Teymur. Testing whether a learning procedure is calibrated. *Journal of Machine Learning Research*, 23(203):1–36, 2022. URL `http://jmlr.org/papers/v23/21-1065.html`.

Gregory Cohen, Saeed Afshar, Jonathan Tapson, and Andre Van Schaik. EMNIST: Extending MNIST to handwritten letters. In *2017 international joint conference on neural networks (IJCNN)*, pp. 2921–2926. IEEE, 2017.

A Philip Dawid. Present position and potential developments: Some personal views statistical theory the prequential approach. *Journal of the Royal Statistical Society: Series A (General)*, 147(2):278–290, 1984.

Tilmann Gneiting and Adrian E Raftery. Strictly proper scoring rules, prediction, and estimation. *Journal of the American statistical Association*, 102(477):359–378, 2007.

Tilmann Gneiting, Fadoua Balabdaoui, and Adrian E Raftery. Probabilistic forecasts, calibration and sharpness. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 69(2):243–268, 2007.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

David Greenberg, Marcel Nonnenmacher, and Jakob Macke. Automatic posterior transformation for likelihood-free inference. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 2404–2414. PMLR, 09–15 Jun 2019. URL `http://proceedings.mlr.press/v97/greenberg19a.html`.

Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *The Journal of Machine Learning Research*, 13(1):723–773, 2012.

Alexey Gritsenko, Tim Salimans, Rianne van den Berg, Jasper Snoek, and Nal Kalchbrenner. A spectral energy distance for parallel speech synthesis. *Advances in Neural Information Processing Systems*, 33: 13062–13072, 2020.

Ali Harakeh and Steven L. Waslander. Estimating and evaluating regression predictive uncertainty in deep object detectors. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=YLewtnvKgR7`.

Joeri Hermans, Volodimir Begy, and Gilles Louppe. Likelihood-free MCMC with amortized approximate ratio estimators. In *International Conference on Machine Learning*, pp. 4239–4248. PMLR, 2020.

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1125–1134, 2017.

Diederik P. Kingma and Max Welling. Auto-encoding variational Bayes. In Yoshua Bengio and Yann LeCun (eds.), *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014. URL `http://arxiv.org/abs/1312.6114`.

Jarno Lintusaari, Michael U. Gutmann, Ritabrata Dutta, Samuel Kaski, and Jukka Corander. Fundamentals and recent developments in approximate Bayesian computation. *Systematic Biology*, 66(1):e66–e82, 2017. ISSN 1076836X. doi: 10.1093/sysbio/syw077. URL `https://doi.org/10.1093/sysbio/syw077`.

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

Jan-Matthis Lueckmann, Giacomo Bassetto, Theofanis Karaletsos, and Jakob H Macke. Likelihood-free inference with emulator networks. In *Symposium on Advances in Approximate Bayesian Inference*, pp. 32–53. PMLR, 2019.

Jan-Matthis Lueckmann, Jan Boelts, David Greenberg, Pedro Goncalves, and Jakob Macke. Benchmarking simulation-based inference. In Arindam Banerjee and Kenji Fukumizu (eds.), *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, volume 130 of *Proceedings of Machine Learning Research*, pp. 343–351. PMLR, 13–15 Apr 2021.

Paul Marjoram, John Molitor, Vincent Plagnol, and Simon Tavaré. Markov chain Monte Carlo without likelihoods. *Proceedings of the National Academy of Sciences*, 100(26):15324–15328, 2003.

Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

Hien Duy Nguyen, Julyan Arbel, Hongliang Lü, and Florence Forbes. Approximate Bayesian computation via the energy statistic. *IEEE Access*, 8:131683–131698, 2020.

Sebastian Nowozin, Botond Cseke, and Ryota Tomioka. f-GAN: Training generative neural samplers using variational divergence minimization. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pp. 271–279, 2016.

Lorenzo Pacchiardi, Rilwan Adewoyin, Peter Dueben, and Ritabrata Dutta. Probabilistic forecasting with conditional generative networks via scoring rule minimization. *arXiv preprint arXiv:2112.08217*, 2022.

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

George Papamakarios, David Sterratt, and Iain Murray. Sequential neural likelihood: Fast likelihood-free inference with autoregressive flows. In Kamalika Chaudhuri and Masashi Sugiyama (eds.), *Proceedings of Machine Learning Research*, volume 89 of *Proceedings of Machine Learning Research*, pp. 837–848. PMLR, 16–18 Apr 2019. URL `http://proceedings.mlr.press/v89/papamakarios19a.html`.

George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. *Journal of Machine Learning Research*, 22(57):1–64, 2021. URL `http://jmlr.org/papers/v22/19-1028.html`.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019.

Stefan T Radev, Ulf K Mertens, Andreas Voss, Lynton Ardizzone, and Ullrich Köthe. BayesFlow: Learning complex stochastic models with invertible neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

Poornima Ramesh, Jan-Matthis Lueckmann, Jan Boelts, Álvaro Tejero-Cantero, David S. Greenberg, Pedro J. Goncalves, and Jakob H. Macke. GATSBI: Generative adversarial training for simulation-based inference. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=kR1hC6j48Tp`.

Eitan Richardson and Yair Weiss. On GANs and GMMs. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pp. 5852–5863, 2018.

Maria L Rizzo and Gábor J Székely. Energy distance. *Wiley interdisciplinary reviews: Computational statistics*, 8(1):27–38, 2016.

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training GANs. *Advances in neural information processing systems*, 29, 2016.

Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. *Advances in neural information processing systems*, 32, 2019.

Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*, 2020.

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

Simon Tavaré, David J Balding, Robert C Griffiths, and Peter Donnelly. Inferring coalescence times from DNA sequence data. *Genetics*, 145(2):505–518, 1997.

Yuexi Wang and Veronika Ročková. Adversarial Bayesian simulation. *arXiv preprint arXiv:2208.12113*, 2022.

Samuel Wiqvist, Jes Frellsen, and Umberto Picchini. Sequential neural posterior and likelihood approximation, 2021.

A Sequential training

In the main text, we have considered the training data from the simulator model $(\boldsymbol{\theta}_i, \mathbf{y}_i)_{i=1}^n$ to be generated independently from the observation on which inference is performed; under this assumption, we have discussed ways to learn posterior approximations valid for all values of $\mathbf{y}$ such that $p(\mathbf{y}) > 0$. Once the neural network is trained, therefore, inference can be performed for as many observations as we wish. This is a so-called *amortized* setup (Radev et al., 2020).

However, practitioners may require posterior inference for a single observation $\mathbf{y}_o$. What they are interested in, therefore, is the quality of the approximation for values of $\boldsymbol{\theta}$ with large posterior density for the observed $\mathbf{y}_o$. In this case, generating training samples independently from $\mathbf{y}_o$ may be wasteful: a more efficient method (in terms of simulations from the model $p(\cdot|\boldsymbol{\theta})$) would generate more training samples $\boldsymbol{\theta}_i$'s close to the modes of the true posterior, as those convey more information on the precise posterior shape. This can be done in a sequential fashion: given a small amount of training data, a first approximation Q_{ϕ_1} is obtained; from that, additional training samples $(\boldsymbol{\theta}_i, \mathbf{y}_i)$ are generated by $\boldsymbol{\theta}_i \sim Q_{\phi_1}(\cdot|\mathbf{y}_o), \mathbf{y}_i \sim P(\cdot|\boldsymbol{\theta}_i)$ and used to (re-)train an approximation Q_{ϕ_2}. This procedure is iterated several times, allowing the training samples to progressively focus around the posterior modes and thus refining the approximation (Lueckmann et al., 2017; Greenberg et al., 2019).

However, naively following that strategy is incorrect. To see this, assume that, at the second round, we just train on samples drawn from the approximate posterior $\tilde{\Pi} = Q_{\phi_1}(\cdot|\mathbf{y}_o)$ obtained at the first round.

Such a sampled pair $(\boldsymbol{\theta}_i, \mathbf{y}_i)$ was drawn from a joint density $\tilde{\pi}(\boldsymbol{\theta}_i)p(\mathbf{y}_i|\boldsymbol{\theta}_i) = \tilde{p}(\mathbf{y}_i)\tilde{\pi}(\boldsymbol{\theta}_i|\mathbf{y}_i)$, where $\tilde{\pi}$ on the left-hand side of the equality is the density of the proposal $\tilde{\Pi}$ and the quantities on the right-hand side are univocally defined by the left-hand side. The optimal $\phi^\star$ obtained via SR-minimization thus corresponds to $q_{\phi^\star}(\cdot|\mathbf{y}) = \tilde{\pi}(\cdot|\mathbf{y})$, which is not the correct target.

The traditional way to fix this entails introducing importance weights in the training objective (Eq. 3):

$$\mathbb{E}_{\boldsymbol{\theta}\sim\Pi}\mathbb{E}_{\mathbf{Y}\sim P(\cdot|\boldsymbol{\theta})}S(Q_\phi(\cdot|\mathbf{Y}), \boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{\theta}\sim\tilde{\Pi}}\frac{\pi(\boldsymbol{\theta})}{\tilde{\pi}(\boldsymbol{\theta})}\mathbb{E}_{\mathbf{Y}\sim P(\cdot|\boldsymbol{\theta})}S(Q_\phi(\cdot|\mathbf{Y}), \boldsymbol{\theta}).$$

As $\tilde{\pi}(\boldsymbol{\theta})$ cannot be evaluated, a solution is to fit a probabilistic classifier (at each round of the sequential procedure) to samples from $\pi(\boldsymbol{\theta})$ and $\tilde{\pi}(\boldsymbol{\theta})$ and use it to estimate the ratio $\frac{\pi(\boldsymbol{\theta})}{\tilde{\pi}(\boldsymbol{\theta})}$. This classifier is not required for the normalizing flows approaches, where the ratio can be evaluated explicitly (Lueckmann et al., 2017; Greenberg et al., 2019) (unless the prior π is also defined implicitly, as in the camera model example in Section 4). For the GAN approach, a similar importance weights approach requires additionally to estimate the ratio $\frac{\tilde{p}(\mathbf{y})}{p(\mathbf{y})}$ (Ramesh et al., 2022).

An alternative approach, which was applied to the GAN approach in Ramesh et al. (2022), involves correcting the distribution of the variable $\mathbf{Z}$ which is transformed by the generative network. Specifically, Ramesh et al. (2022) showed that $\pi(\boldsymbol{\theta}|\mathbf{y}) = \tilde{\pi}(\boldsymbol{\theta}|\mathbf{y})w(\boldsymbol{\theta}, \mathbf{y}) \iff \tilde{\pi}(\boldsymbol{\theta}|\mathbf{y}) = \pi(\boldsymbol{\theta}|\mathbf{y})(w(\boldsymbol{\theta}, \mathbf{y}))^{-1}$, where $w(\boldsymbol{\theta}, \mathbf{y}) = \frac{\pi(\boldsymbol{\theta})}{\tilde{\pi}(\boldsymbol{\theta})}\frac{\tilde{p}(\mathbf{y})}{p(\mathbf{y})}$. Therefore you can consider a modified approximation $\tilde{Q}_\phi(\cdot|\mathbf{Y})$ and a new training objective:

$$\mathbb{E}_{\mathbf{Y}\sim\tilde{P}}\mathbb{E}_{\boldsymbol{\theta}\sim\tilde{\Pi}(\cdot|\mathbf{Y})}S(\tilde{Q}_\phi(\cdot|\mathbf{Y}), \boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{\theta}\sim\tilde{\Pi}}\mathbb{E}_{\mathbf{Y}\sim P(\cdot|\boldsymbol{\theta})}S(\tilde{Q}_\phi(\cdot|\mathbf{Y}), \boldsymbol{\theta}) \tag{8}$$

whose minimization leads to $\tilde{Q}_\phi(\cdot|\mathbf{Y}) = \tilde{\Pi}(\cdot|\mathbf{Y})$. By setting

$$\tilde{Q}_\phi(\cdot|\mathbf{Y}) = Q_\phi(\cdot|\mathbf{Y})(w(\boldsymbol{\theta}, \mathbf{y}))^{-1},$$

you ensure $Q_\phi(\cdot|\mathbf{Y}) = \Pi(\cdot|\mathbf{Y})$. To train ϕ using the objective in Eq. (8), draws from $\tilde{Q}_\phi(\cdot|\mathbf{Y})$ are required; those can be obtained by sampling $\mathbf{z} \sim \tilde{P}_\mathbf{z}$, whose density is $\tilde{p}_\mathbf{z}(\mathbf{z}) = p_\mathbf{z}(\mathbf{z})(w(g_\phi(\mathbf{z}, \mathbf{y}), \mathbf{y}))^{-1}$, and computing $\boldsymbol{\theta} = g_\phi(\mathbf{z}, \mathbf{y})$, which is thus a sample from $\tilde{Q}_\phi(\cdot|\mathbf{Y})$. Compared to using importance weights, the variance of the training objective is here smaller. However, rejection sampling or MCMC are needed to sample from $\tilde{P}_\mathbf{z}$, and two ratios have to be estimated via probabilistic classifiers ($\frac{\tilde{p}(\mathbf{y})}{p(\mathbf{y})}$ and $\frac{\pi(\boldsymbol{\theta})}{\tilde{\pi}(\boldsymbol{\theta})}$), making this strategy more expensive than using importance weights

On the examples considered in Ramesh et al. (2022), the sequential approaches did not perform better than the amortized one, mainly due to the additional computational cost associated to estimating the ratios. For that reason, we did not investigate these methods for our approach.

B *f*-GAN

The problem in Eq. (6) can be obtained as a relaxation of the following one:

$$\arg\min_\phi \mathbb{E}_{\mathbf{Y}\sim P}\left[D_{JS}(\Pi(\cdot|\mathbf{Y})\|Q_\phi(\cdot|\mathbf{Y}))\right],$$

where D_{JS} is the Jensen-Shannon divergence. The objective in the above problem is 0 if and only if $\Pi(\cdot|\mathbf{y}) = Q_\phi(\cdot|\mathbf{y})$ for each $\mathbf{y} : p(\mathbf{y}) > 0$. We report here a more general result by considering a class of divergences known as f-divergences, to which the Jensen-Shannon one belongs. We follow Nowozin et al. (2016) in doing so[4].

By temporarily discarding the dependence on $\mathbf{Y}$, an f-divergence is defined as:

$$D_f(P\|Q_\phi) = \int q_\phi(\boldsymbol{\theta}) f\left(\frac{p(\boldsymbol{\theta})}{q_\phi(\boldsymbol{\theta})}\right) d\mu(\boldsymbol{\theta}),$$

---

[4] An analogous procedure allows to obtain a tractable training objective for the 1-Wasserstein distance as well (Arjovsky et al., 2017)

where $f : \mathbb{R}_+ \to \mathbb{R}$ is a convex, lower-semicontinuous function for which $f(1) = 0$, and where q_ϕ and p are densities of Q_ϕ and P with respect to a base measure μ. We want now to fix ϕ by:

$$\arg\min_\phi D_f(P||Q_\phi). \tag{9}$$

Let now dom_f denote the domain of f. By exploiting the Fenchel conjugate $f^*(t) = \sup_{u\in\mathrm{dom}_f} \{ut - f(u)\}$, Nowozin et al., 2016 obtain the following variational lower bound:

$$D_f(P||Q_\phi) \geq \sup_{c\in\mathcal{C}} \left(\mathbb{E}_{\boldsymbol{\theta}\sim P} c(\boldsymbol{\theta}) - \mathbb{E}_{\tilde{\boldsymbol{\theta}}\sim Q_\phi} f^*(c(\tilde{\boldsymbol{\theta}})) \right),$$

which holds for any set of functions $\mathcal{C}$ from $\mathcal{Y}$ to dom_{f^*}. By considering a parametric set of functions $\mathcal{C} = \{c_\psi : \mathcal{Y} \to \mathrm{dom}_{f^*}, \psi \in \Psi\}$, a surrogate to the problem in Eq. (9) becomes:

$$\min_\phi \max_\psi \left(\mathbb{E}_{\boldsymbol{\theta}\sim P} c_\psi(\boldsymbol{\theta}) - \mathbb{E}_{\tilde{\boldsymbol{\theta}}\sim Q_\phi} f^*(c_\psi(\tilde{\boldsymbol{\theta}})) \right).$$

By reintroducing the dependence on $\mathbf{Y}$, the above generalizes to:

$$\min_\phi \max_\psi \mathbb{E}_{\mathbf{Y}\sim P} \left(\mathbb{E}_{\boldsymbol{\theta}\sim P(\cdot|\mathbf{Y})} c_\psi(\boldsymbol{\theta}, \mathbf{Y}) - \mathbb{E}_{\tilde{\boldsymbol{\theta}}\sim Q_\phi(\cdot|\mathbf{Y})} f^*(c_\psi(\tilde{\boldsymbol{\theta}}, \mathbf{Y})) \right), \tag{11}$$

where now the function c_ψ also depends on the value of $\mathbf{Y}$.

In practice, c_ψ is parametrized by a Neural Network. To solve the problem in Eq. (11), people usually employ alternating optimization over ϕ and ψ by following stochastic gradients; this technique is called f-GAN. With a finite number of steps over ψ, this leads to biased gradient estimates for ϕ. In Algorithm 2, we show a single epoch (i.e. a loop on the full training dataset) of conditional f-GAN training; for simplicity, we consider here using a single pair $(\boldsymbol{\theta}_i, \mathbf{y}_i)$ to estimate the expectations in Eq. (11) (i.e., the batch size is 1), but using a larger number of samples is possible. Notice how in Algorithm 2 we update the critic once every generator update; however, multiple critic updates can be performed at each generator update.

Algorithm 2 Single epoch conditional f-GAN training.

```
Require: Parametric map $g_\phi$, critic network $c_\psi$, learning rates $\epsilon$, $\gamma$.
  for each training pair $(\boldsymbol{\theta}_i, \mathbf{y}_i)$ do
      Sample $\mathbf{z} \sim P_\mathbf{z}$
      Obtain $\tilde{\boldsymbol{\theta}}_i^\phi = g_\phi(\mathbf{z}, \mathbf{y}_i)$
      Set $\psi \leftarrow \psi + \gamma \cdot \nabla_\psi \Big[ c_\psi(\boldsymbol{\theta}_i, \mathbf{y}_i) - f^*(c_\psi(\tilde{\boldsymbol{\theta}}_i^\phi, \mathbf{y}_i)) \Big]$
      Set $\phi \leftarrow \phi - \epsilon \cdot \nabla_\phi \Big[ - f^*(c_\psi(\tilde{\boldsymbol{\theta}}_i^\phi, \mathbf{y}_i)) \Big]$
  end for
```

C Unbiased gradient estimates

We discuss here how we can obtain unbiased gradient estimates for the Scoring Rule training objective in Eq. (4) with respect to the parameters of the generative network ϕ.

In order to do that, we first discuss how to obtain unbiased estimates of the SRs we use across this work. Then, we show how those allow one to obtain unbiased gradient estimates. The steps we follow are the same as in Pacchiardi et al. (2022) for the setting of probabilistic forecasting.

C.1 Unbiased scoring rule estimates

Assume we have draws $\tilde{\mathbf{x}}_j \sim P, j = 1, \ldots, m$.

Energy Score An unbiased estimate of the energy score can be obtained by unbiasedly estimating the expectations in $S_{\mathrm{E}}^{(\beta)}(P, \mathbf{x})$ in Eq. (1):

$$\hat{S}_{\mathrm{E}}^{(\beta)}(P, \mathbf{x}) = \frac{2}{m} \sum_{j=1}^{m} \|\tilde{\mathbf{x}}_j - \mathbf{x}\|_2^{\beta} - \frac{1}{m(m-1)} \sum_{\substack{j,k=1 \\ k \neq j}}^{m} \|\tilde{\mathbf{x}}_j - \tilde{\mathbf{x}}_k\|_2^{\beta}.$$

Kernel Score Similarly to the energy score, we obtain an unbiased estimate of $S_k(P, \mathbf{x})$ in Eq. (2) by:

$$\hat{S}_k(P, \mathbf{x}) = \frac{1}{m(m-1)} \sum_{\substack{j,k=1 \\ k \neq j}}^{m} k(\tilde{\mathbf{x}}_j, \tilde{\mathbf{x}}_k) - \frac{2}{m} \sum_{j=1}^{m} k(\tilde{\mathbf{x}}_j, \mathbf{x}).$$

Sum of SRs When adding multiple SRs, an unbiased gradient of the sum can be obtained by adding unbiased estimates of the two addends.

C.2 Unbiased estimate of the training objective

Recall now that we want to solve:

$$\hat{\phi} := \arg\min_{\phi} J(\phi), \quad J(\phi) = \frac{1}{n} \sum_{i=1}^{n} S(Q_\phi(\cdot|\mathbf{y}_i), \boldsymbol{\theta}_i). \tag{12}$$

To do this, we exploit Stochastic Gradient Descent (SGD), which requires unbiased estimates of $J(\phi)$. Notice how, for all the Scoring Rules used across this work, as well as any weighted sum of those, we can write: $S(P, \mathbf{x}) = \mathbb{E}_{\tilde{\mathbf{X}}, \tilde{\mathbf{X}}' \sim P}\left[h(\tilde{\mathbf{X}}, \tilde{\mathbf{X}}', \mathbf{x})\right]$ for some function h; namely, the SR is defined through an expectation over (possibly multiple) samples from P. That is the form exploited in Appendix C.1 to obtain unbiased SR estimates.

Now, we will use this fact to obtain unbiased estimates for the objective in Eq. (12).

$$J(\phi) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{E}_{\tilde{\boldsymbol{\theta}}, \tilde{\boldsymbol{\theta}}' \sim Q_\phi(\cdot|\mathbf{y}_i)}\left[h(\tilde{\boldsymbol{\theta}}, \tilde{\boldsymbol{\theta}}', \boldsymbol{\theta}_i)\right] = \frac{1}{n} \sum_{i=1}^{n} \mathbb{E}_{\mathbf{Z}, \mathbf{Z}' \sim P_{\mathbf{z}}}\left[h(g_\phi(\mathbf{Z}, \mathbf{y}_i), g_\phi(\mathbf{Z}', \mathbf{y}_i), \boldsymbol{\theta}_i)\right],$$

where we used the fact that Q_ϕ is the distribution induced by a generative network with transformation g_ϕ; this is called the reparametrization trick (Kingma & Welling, 2014). Now:

$$\begin{aligned} \nabla_\phi J(\phi) &= \nabla_\phi \frac{1}{n} \sum_{i=1}^{n} \mathbb{E}_{\mathbf{Z}, \mathbf{Z}' \sim P_{\mathbf{z}}}\left[h(g_\phi(\mathbf{Z}, \mathbf{y}_i), g_\phi(\mathbf{Z}', \mathbf{y}_i), \boldsymbol{\theta}_i)\right] \\ &= \frac{1}{n} \sum_{i=1}^{n} \mathbb{E}_{\mathbf{Z}, \mathbf{Z}' \sim P_{\mathbf{z}}}\left[\nabla_\phi h(g_\phi(\mathbf{Z}, \mathbf{y}_i), g_\phi(\mathbf{Z}', \mathbf{y}_i), \boldsymbol{\theta}_i)\right]. \end{aligned}$$

In the latter equality, the exchange between expectation and gradient is not a trivial step, due to the non-differentiability of functions (such as ReLU) used in g_ϕ. Fortunately, Theorem 5 in Bińkowski et al. (2018) proved that to be valid almost surely with respect to a measure on the space Φ to which the weights of the neural network ϕ belong, under mild conditions on the NN architecture.

We can now easily obtain an unbiased estimate of the above using samples $\mathbf{z}_{i,j} \sim P_{\mathbf{z}}, j = 1, \ldots, m$, for each $i \in \{1, \ldots, n\}$. Additionally, Stochastic Gradient Descent usually considers a small batch of training samples at each step, obtained by taking a random subset (or batch) $\mathcal{B} \subseteq \{1, 2, \ldots, n\}$. Therefore, the following unbiased estimate of $\nabla_\phi J(\phi)$ can be obtained:

$$\widehat{\nabla_\phi J(\phi)} = \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \frac{1}{m(m-1)} \sum_{\substack{j,k=1 \\ j \neq k}}^{m} \nabla_\phi h(g_\phi(\mathbf{z}_{i,j}; \mathbf{y}_i), g_\phi(\mathbf{z}_{i,k}; \mathbf{y}_i), \boldsymbol{\theta}_i).$$

In practice, the above is obtained by computing the gradient of the following unbiased estimate of $J(\phi)$ via autodifferentiation libraries (see for instance Paszke et al., 2019):

$$\hat{J}(\phi) = \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \frac{1}{m(m-1)} \sum_{\substack{j,k=1 \\ j \neq k}}^{m} h(g_\phi(\mathbf{z}_{i,j}; \mathbf{y}_i), g_\phi(\mathbf{z}_{i,k}; \mathbf{y}_i), \boldsymbol{\theta}_i).$$

In Algorithm 1, we train a generative network for a single epoch using a scoring rule S for which unbiased estimators can be obtained by using $m > 1$ samples from Q_ϕ. Compare it with the adversarial approach reported in Algorithm 2; in the SR approach, multiple samples from the generative networks are required at each step ($m > 1$), while a unique one is enough for the adversarial approach. Conversely, the SR approach does not require an additional critic network and learning rate γ and is simpler and faster to train (see the results in Sec. 4 and Pacchiardi et al., 2022 for more details). As in Algorithm 2 , we use a single pair $(\boldsymbol{\theta}_i, \mathbf{y}_i)$ to estimate the gradient.

D Details on performance measures

Here, we review the measures of performance used in the empirical studies. We follow Radev et al. (2020) in defining these measures and report them here for ease of reference. All these metrics are for univariate $\boldsymbol{\theta}$; when handling multivariate $\boldsymbol{\theta}$, we therefore compute them on each dimension separately and report the average.

D.1 Deterministic performance measures

We discuss two measures of performance of a deterministic forecast $\hat{\boldsymbol{\theta}}_i$ for a realization $\boldsymbol{\theta}_i$; across our work, we take $\hat{\boldsymbol{\theta}}_i$ to be the mean of the (univariate) probability distribution $Q_\phi(\cdot|\mathbf{y}_i)$.

D.1.1 RMSE

We first introduce the Root Mean-Square Error (RMSE) as:

$$\mathrm{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^{n} \left(\hat{\boldsymbol{\theta}}_i - \boldsymbol{\theta}_i\right)^2},$$

where we consider here for simplicity $i = 1, \ldots, n$. From the above, we obtain the Normalized RMSE (NRMSE) as:

$$\mathrm{NRMSE} = \frac{RMSE}{\max_i\{\boldsymbol{\theta}_i\} - \min_i\{\boldsymbol{\theta}_i\}}.$$

NRMSE $= 0$ implies $\hat{\boldsymbol{\theta}}_i = \boldsymbol{\theta}_i$ for all i's. NRMSE $\in [0, 1]$ and allows to compare performance over different tasks. Notice however that, when $\max_i\{\boldsymbol{\theta}_i\} = \min_i\{\boldsymbol{\theta}_i\}$, NRMSE diverges; in that case, we consider the un-normalized RMSE.

D.1.2 Coefficient of determination

The coefficient of determination R^2 measures how much of the variance in $\{\boldsymbol{\theta}_i\}_{i=1}^n$ is explained by $\{\hat{\boldsymbol{\theta}}_i\}_{i=1}^n$. Specifically, it is given by:

$$\mathrm{R}^2 = 1 - \frac{\sum_{i=1}^n \left(\boldsymbol{\theta}_i - \hat{\boldsymbol{\theta}}_i\right)^2}{\sum_{i=1}^n \left(\boldsymbol{\theta}_i - \bar{\boldsymbol{\theta}}\right)^2},$$

where $\bar{\boldsymbol{\theta}} = \frac{1}{n} \sum\limits_{i=1}^{n} \boldsymbol{\theta}_i$. $\mathrm{R}^2 \leq 1$ and $\mathrm{R}^2 = 1 \implies \hat{\boldsymbol{\theta}}_i = \boldsymbol{\theta}_i$ for all i's.

D.2 Calibration measures

Here, we review two measures of calibration of a probabilistic forecast. Both measures consider the univariate marginals of the approximate posterior distribution $Q_\phi(\cdot|\mathbf{y}_i)$; for the component l, let us denote it by $Q_{\phi,l}(\cdot|\mathbf{y}_i)$.

D.2.1 Calibration error

The calibration error (Radev et al., 2020) quantifies how well the credible intervals of the approximate posteriors $Q_{\phi,l}(\cdot|\mathbf{y}_i)$ for different $\mathbf{y}_i$ match the empirical distribution of $\theta_{i,l}$. Specifically, let $\alpha(l)$ be the proportion of times the verification $\theta_{i,l}$ falls into an α-credible interval of $Q_{\phi,l}(\cdot|\mathbf{y}_i)$, computed over all values of i. If the marginal forecast distribution is perfectly calibrated for component l, $\alpha(l) = \alpha$ for all values of $\alpha \in (0, 1)$.

Therefore, we define the calibration error as the median of $|\alpha(l) - \alpha|$ over 100 equally spaced values of $\alpha \in (0, 1)$. Therefore, the calibration error is a value between 0 and 1, where 0 denotes perfect calibration.

In practice, the credible intervals of the predictive are estimated using a set of samples from $Q_\phi(\cdot|\mathbf{y}_i)$.

The calibration error can be related to the *strong calibration* of Cockayne et al. (2022), which implies correct coverage for credible sets (see their Remark 2.9).

D.2.2 Simulation-Based Calibration (SBC)

SBC (Talts et al., 2018) tests a self-consistency property of the Bayesian posterior in a posterior approximation. In fact, the Bayesian posterior satisfies the following equality:

$$\pi(\boldsymbol{\theta}) = \int p(\boldsymbol{\theta}, \tilde{\boldsymbol{\theta}}, \tilde{\mathbf{y}}) d\tilde{\mathbf{y}} d\tilde{\boldsymbol{\theta}} = \int p(\boldsymbol{\theta}, \tilde{\mathbf{y}} \mid \tilde{\boldsymbol{\theta}}) \pi(\tilde{\boldsymbol{\theta}}) d\tilde{\mathbf{y}} d\tilde{\boldsymbol{\theta}} = \int \pi(\boldsymbol{\theta} \mid \tilde{\mathbf{y}}) p(\tilde{\mathbf{y}} \mid \tilde{\boldsymbol{\theta}}) \pi(\tilde{\boldsymbol{\theta}}) d\tilde{\mathbf{y}} d\tilde{\boldsymbol{\theta}} \tag{13}$$

in practice, this means that, if you sample from the prior $\tilde{\boldsymbol{\theta}} \sim \pi$, use that to generate a sample from the likelihood $\tilde{\mathbf{y}} \sim p(\cdot|\boldsymbol{\theta})$ and use the latter in turn to generate a posterior sample $\boldsymbol{\theta} \sim \pi(\cdot|\tilde{\mathbf{y}})$, $\boldsymbol{\theta}$ is distributed according to the prior $\pi(\boldsymbol{\theta})$. If you repeat the same procedure by sampling $\boldsymbol{\theta}$ from an *approximate* posterior, say $\boldsymbol{\theta} \sim Q_\phi(\cdot|\tilde{\mathbf{y}})$, then $\boldsymbol{\theta} \sim \pi$ is a necessary condition for $q_\phi(\cdot|\mathbf{y}) = \pi(\cdot|\mathbf{y})$, i.e. for the approximate posterior to be exact. Notice, however, how this is *not* a sufficient condition: the equality can be satisfied even if $q_\phi(\cdot|\mathbf{y})$ is different from the posterior (it is in fact trivially satisfied $q_\phi(\cdot|\mathbf{y}) = \pi$, i.e., when the approximate posterior corresponds to the prior).

A way to empirically test the above property involves, for a given prior sample $\tilde{\boldsymbol{\theta}}$, drawing from the likelihood multiple times $\mathbf{y}_i \sim p(\cdot|\tilde{\boldsymbol{\theta}}), i = 1, \ldots, N$ and, for each of these, obtaining a single approximate posterior sample $\boldsymbol{\theta}_i \sim q_\phi(\cdot|\mathbf{y}_i)$. Given these, you compute the rank of $\tilde{\boldsymbol{\theta}}$: $r = \sum_{i=1}^N \mathbf{1}_{[\boldsymbol{\theta}_i < \tilde{\boldsymbol{\theta}}]}$ (this only makes sense if $\boldsymbol{\theta}$ is univariate; otherwise, you compute the rank independently for each dimension of $\boldsymbol{\theta}$). If $\boldsymbol{\theta}_i$'s were effectively distributed from the prior, r is a uniform random variable on $\{1, 2, \ldots, N\}$. Therefore, repeating this procedure for different prior samples $\tilde{\boldsymbol{\theta}}$ and visualizing the distribution of the resulting r' s (for instance, through a histogram or by plotting the CDF) gives an indication of whether an equivalence such as Eq. (13) is satisfied for q_ϕ. See Algorithm 2 in Radev et al. (2020) for a precise description of this procedure, which goes under the name of Simulation-Based Calibration (SBC). SBC tests the *weak calibration* of Cockayne et al. (2022); additionally, it is closely related to the concept of probabilistic calibration and rank histogram in the framework of probabilistic forecasting (Gneiting et al., 2007).

E Experimental details

Precise configuration details can be found in the code accompanying the paper <link removed for anonymity>.

E.1 Shallow Water Model

We train all methods for at most 40k epochs on 100k training samples. For the SR method, we tried both $m = 3$ and $m = 10$, with the latter resulting in improved performance; all the results reported in the paper refer to $m = 10$.

GAN used a batch size of 125 (as in Ramesh et al., 2022), while SR methods used a batch size of 60 (otherwise, GPU memory overflow occurs).

Recall that the parameters $\boldsymbol{\theta} \in \mathbb{R}^{100}$ are arranged along a 1D uniform grid. When using the patched SR configuration, we consider patches of size `patch_size` disposed at a distance `patch_step` from each other. Therefore, the number of patches is

$$\texttt{n\_patches} = (100 - \texttt{patch\_size})/\texttt{patch\_step} + 1.$$

We used therefore the following patched SR configurations on the 1D grid:

1. `patch_size` $= 10$ and `patch_step`$= 5$, resulting in `n_patches` $= 19$.
2. `patch_size` $= 20$ and `patch_step`$= 10$, resulting in `n_patches` $= 9$.

The patched SR is added to the overall score over the full parameter space.

The training time (per epoch) is roughly constant in the un-patched and the two different patched configurations.

E.2 Camera Model

We train all methods for at most 10k epochs on 800k training samples. For the SR method, we tried both $m = 3$ and $m = 10$, with the latter resulting in better performance.

Both the SR and GAN methods used a batch size of 800 as in Ramesh et al. (2022).

Here, the parameters $\boldsymbol{\theta}$ are on a 28×28 square grid. When using the patched SR configuration, we consider patches of size `patch_size`$\times$`patch_size` disposed at a distance `patch_step` from each other in both spatial dimensions. The number of patches is obtained as

$$\texttt{n\_patches} = [(28 - \texttt{patch\_size})/\texttt{patch\_step} + 1]^2.$$

We used therefore the following patched SR configurations on the 2D grid:

1. `patch_size` $= 14$ and `patch_step` $= 7$, resulting in `n_patches` $= 9$.
2. `patch_size`$= 8$ and `patch_step` $= 5$, resulting in `n_patches` $= 25$.

The patched SR is added to the overall score over the full parameter space.

The training time (per epoch) is roughly constant in the un-patched and the two different patched configurations.

F Additional experimental results

F.1 SLCP

In Figure 5, we report the posterior samples obtained with the Energy Score with $m = 20$ and compare them with the samples from the reference posterior. In Figure 6, we report Simulation-Based Calibration results (see Appendix D.2.2): for each dimension of $\boldsymbol{\theta}$, the corresponding histogram represents the distribution of the rank of the true parameter value in a set of samples from the approximate posterior. We show that for GAN and for the Energy Score with $m = 20$.

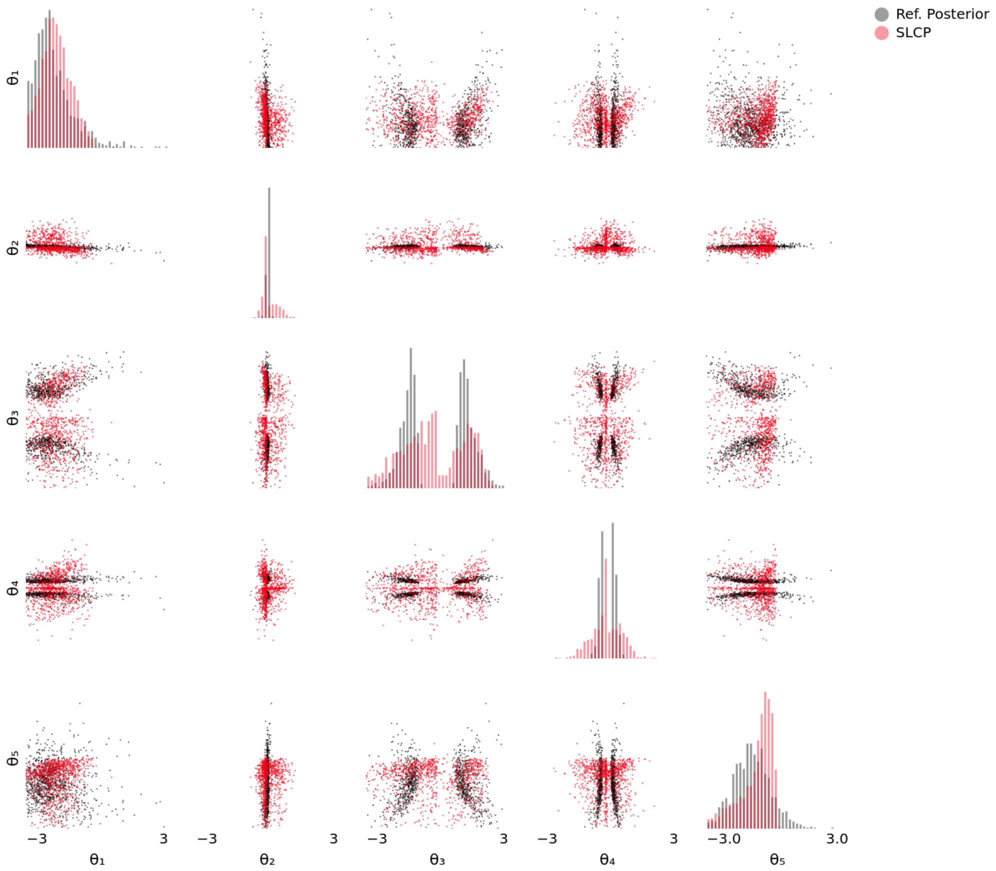
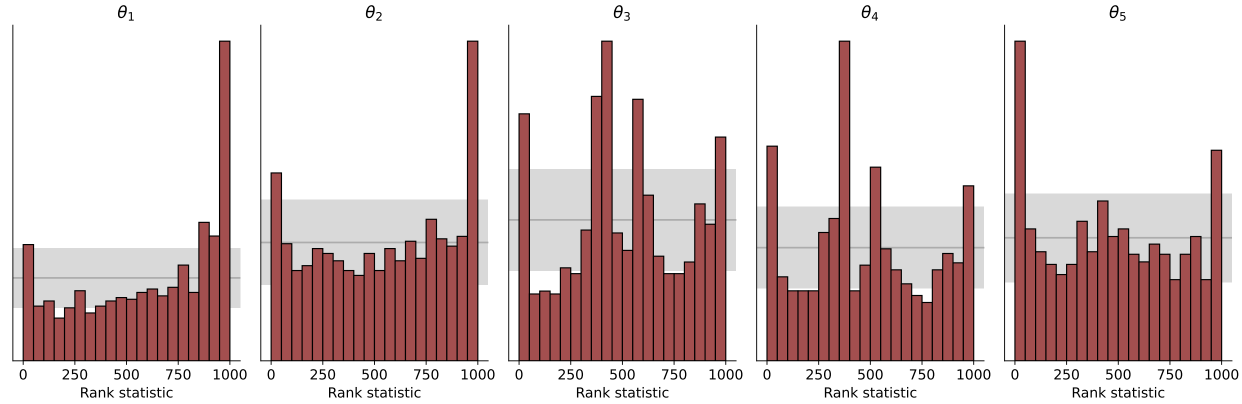
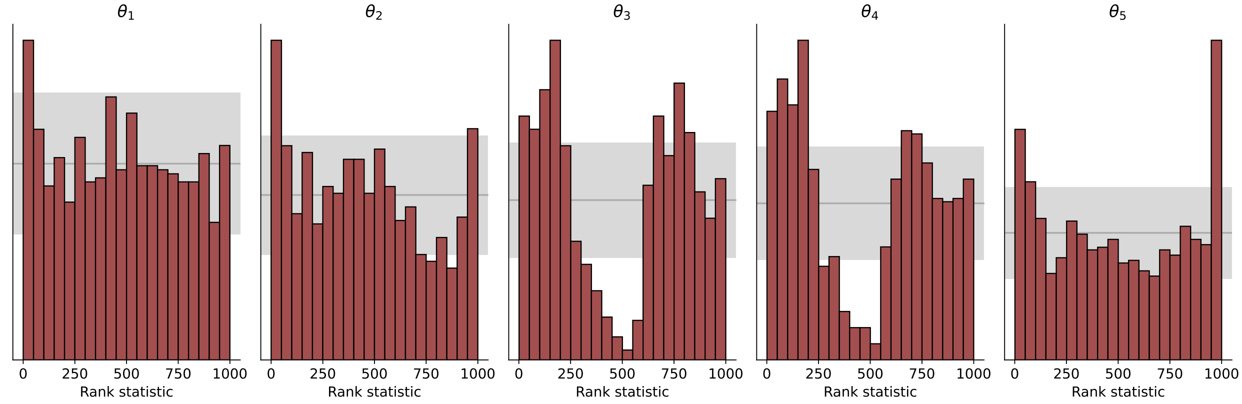


Figure 5: SLCP: posterior samples for Energy Score trained with $m = 20$ and reference posterior samples. Diagonal panels represent univariate marginals, while off-diagonals panels represent bivariate marginals. A similar graph for GAN can be found in the supplementary material in Ramesh et al. (2022).



(a) GAN



(b) Energy Score, $m = 20$

Figure 6: SLCP: Simulation-Based Calibration results represented as rank histograms; for each dimension of $\boldsymbol{\theta}$, the corresponding histogram represents the distribution of the rank of the true parameter value in a set of samples from the approximate posterior. If the approximate posterior is calibrated, histogram bars should be in the grey region with 99% probability.

Tables 5, 6, 7, 8, 9 and 10 report the different performance metrics, the runtime, and the early stopping epoch for all methods (columns) and all number of training samples (rows); for Energy and Kernel Score, the number in the column header denotes the number of draws from the generative network during training for each $\mathbf{y}_i$ in the training batch.

Table 5: SLCP: classification-based two-sample test (C2ST); lower is better.

	GAN	Energy 3	Energy 5	Energy 10	Energy 20	Kernel 3	Kernel 5	Kernel 10	Kernel 20
1000	0.97 ± 0.02	0.99 ± 0.01	0.99 ± 0.01	0.99 ± 0.00	0.99 ± 0.01	1.00 ± 0.01	0.99 ± 0.01	0.99 ± 0.01	0.99 ± 0.01
10000	0.94 ± 0.03	0.98 ± 0.01	0.97 ± 0.01	0.98 ± 0.01	0.98 ± 0.01	0.99 ± 0.01	0.99 ± 0.01	0.99 ± 0.01	0.99 ± 0.01
100000	0.92 ± 0.03	0.97 ± 0.01	0.97 ± 0.02	0.96 ± 0.02	0.95 ± 0.02	0.98 ± 0.01	0.98 ± 0.01	0.98 ± 0.01	0.98 ± 0.01

Table 6: SLCP: NRMSE; lower is better.

	GAN	Energy 3	Energy 5	Energy 10	Energy 20	Kernel 3	Kernel 5	Kernel 10	Kernel 20
1000	0.24 ± 0.05	0.25 ± 0.05	0.25 ± 0.05	0.25 ± 0.05	0.25 ± 0.06	0.25 ± 0.05	0.25 ± 0.05	0.25 ± 0.05	0.25 ± 0.05
10000	0.23 ± 0.05	0.23 ± 0.05	0.23 ± 0.05	0.23 ± 0.05	0.23 ± 0.05	0.23 ± 0.05	0.23 ± 0.05	0.23 ± 0.05	0.23 ± 0.05
100000	0.23 ± 0.05	0.22 ± 0.05	0.22 ± 0.06	0.22 ± 0.06	0.22 ± 0.06	0.22 ± 0.06	0.22 ± 0.06	0.22 ± 0.05	0.22 ± 0.06

Table 7: SLCP: calibration error; lower is better.

	GAN	Energy 3	Energy 5	Energy 10	Energy 20	Kernel 3	Kernel 5	Kernel 10	Kernel 20
1000	0.13 ± 0.05	0.19 ± 0.07	0.20 ± 0.05	0.20 ± 0.05	0.22 ± 0.07	0.24 ± 0.09	0.23 ± 0.10	0.24 ± 0.08	0.24 ± 0.08
10000	0.08 ± 0.03	0.11 ± 0.05	0.10 ± 0.05	0.12 ± 0.07	0.10 ± 0.07	0.15 ± 0.10	0.13 ± 0.09	0.14 ± 0.10	0.16 ± 0.10
100000	0.06 ± 0.03	0.08 ± 0.07	0.08 ± 0.04	0.07 ± 0.05	0.07 ± 0.04	0.13 ± 0.11	0.13 ± 0.10	0.12 ± 0.08	0.13 ± 0.10

Table 8: SLCP: R^2; larger is better.

	GAN	Energy 3	Energy 5	Energy 10	Energy 20	Kernel 3	Kernel 5	Kernel 10	Kernel 20
1000	0.25 ± 0.29	0.24 ± 0.30	0.22 ± 0.30	0.25 ± 0.31	0.18 ± 0.35	0.22 ± 0.30	0.22 ± 0.30	0.24 ± 0.30	0.23 ± 0.31
10000	0.35 ± 0.30	0.35 ± 0.30	0.35 ± 0.30	0.35 ± 0.30	0.34 ± 0.31	0.35 ± 0.29	0.35 ± 0.30	0.34 ± 0.30	0.34 ± 0.30
100000	0.35 ± 0.30	0.36 ± 0.30	0.37 ± 0.30	0.38 ± 0.32	0.38 ± 0.32	0.37 ± 0.31	0.36 ± 0.31	0.36 ± 0.30	0.37 ± 0.31

Table 9: SLCP: runtime in seconds; recall that GAN was trained on GPU while the SR methods were trained on a single CPU.

	GAN	Energy 3	Energy 5	Energy 10	Energy 20	Kernel 3	Kernel 5	Kernel 10	Kernel 20
1000	4796	654	692	620	885	515	531	682	1330
10000	9671	651	658	639	720	636	658	655	697
100000	30963	1060	1160	1305	1645	1245	1044	1057	1210

Table 10: SLCP: epoch at which early stopping occurred; the max number of training epochs was 20000.

	GAN	Energy 3	Energy 5	Energy 10	Energy 20	Kernel 3	Kernel 5	Kernel 10	Kernel 20
1000	20000	1000	1000	1000	1100	1100	1000	1000	1000
10000	20000	1100	1000	1100	1100	1100	1100	1000	1000
100000	20000	1000	1200	1500	2100	1600	1100	1000	1200

F.2 Two Moons

In Figure 7, we report posterior samples obtained with the Energy Score with $m = 20$ and compare them with samples from the reference posterior. In Figure 8, we report Simulation-Based Calibration results (see Appendix D.2.2): for each dimension of $\boldsymbol{\theta}$, the corresponding histogram represents the distribution of the rank of the true parameter value in a set of samples from the approximate posterior. We show that for GAN and for the Energy Score with $m = 20$.

Tables 11, 12, 13, 14, 15 and 16 report the different performance metrics, the runtime and the early stopping epoch for all methods (columns) and all number of training samples (rows); for Energy and Kernel Score, the number in the column header denotes the number of draws from the generative network during training for each $\mathbf{y}_i$ in the training batch.

Table 11: Two Moons: classification-based two-sample test (C2ST); lower is better.

	GAN	Energy 3	Energy 5	Energy 10	Energy 20	Kernel 3	Kernel 5	Kernel 10	Kernel 20
1000	0.85 ± 0.05	0.85 ± 0.06	0.87 ± 0.05	0.85 ± 0.03	0.85 ± 0.04	0.94 ± 0.03	0.94 ± 0.02	0.93 ± 0.03	0.96 ± 0.02
10000	0.81 ± 0.03	0.79 ± 0.04	0.76 ± 0.05	0.76 ± 0.04	0.74 ± 0.07	0.92 ± 0.03	0.93 ± 0.01	0.91 ± 0.03	0.93 ± 0.01
100000	0.82 ± 0.07	0.79 ± 0.03	0.74 ± 0.06	0.73 ± 0.05	0.73 ± 0.04	0.90 ± 0.04	0.92 ± 0.03	0.90 ± 0.02	0.92 ± 0.02

Table 12: Two Moons: NRMSE; lower is better.

	GAN	Energy 3	Energy 5	Energy 10	Energy 20	Kernel 3	Kernel 5	Kernel 10	Kernel 20
1000	0.20 ± 0.00	0.20 ± 0.00	0.20 ± 0.00	0.20 ± 0.00	0.20 ± 0.00	0.21 ± 0.00	0.21 ± 0.00	0.21 ± 0.00	0.20 ± 0.00
10000	0.20 ± 0.00	0.20 ± 0.00	0.20 ± 0.00	0.20 ± 0.00	0.20 ± 0.00	0.20 ± 0.00	0.20 ± 0.00	0.20 ± 0.00	0.20 ± 0.00
100000	0.20 ± 0.00	0.20 ± 0.00	0.20 ± 0.00	0.20 ± 0.00	0.20 ± 0.00	0.20 ± 0.00	0.20 ± 0.00	0.20 ± 0.00	0.20 ± 0.00

Table 13: Two Moons: calibration error; lower is better.

	GAN	Energy 3	Energy 5	Energy 10	Energy 20	Kernel 3	Kernel 5	Kernel 10	Kernel 20
1000	0.07 ± 0.01	0.05 ± 0.01	0.09 ± 0.02	0.07 ± 0.01	0.06 ± 0.00	0.08 ± 0.01	0.11 ± 0.00	0.14 ± 0.02	0.12 ± 0.01
10000	0.06 ± 0.01	0.04 ± 0.02	0.03 ± 0.01	0.04 ± 0.03	0.03 ± 0.01	0.04 ± 0.00	0.03 ± 0.01	0.03 ± 0.02	0.03 ± 0.01
100000	0.07 ± 0.02	0.04 ± 0.01	0.03 ± 0.00	0.04 ± 0.02	0.03 ± 0.00	0.04 ± 0.00	0.03 ± 0.01	0.06 ± 0.01	0.03 ± 0.01

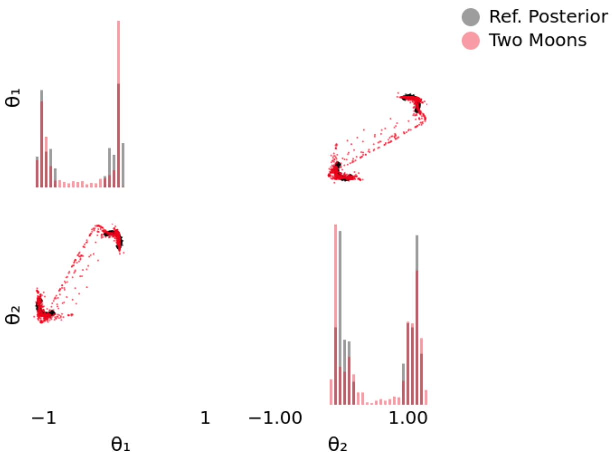
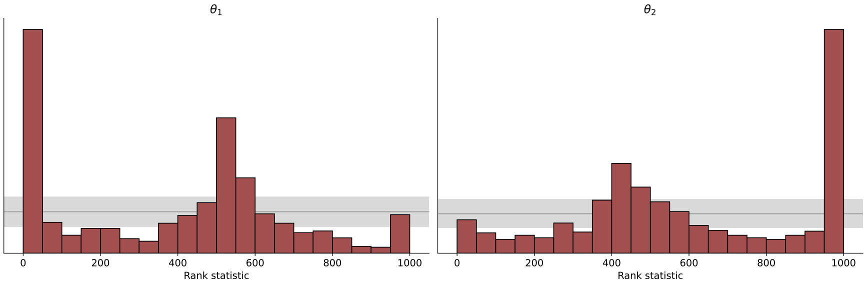
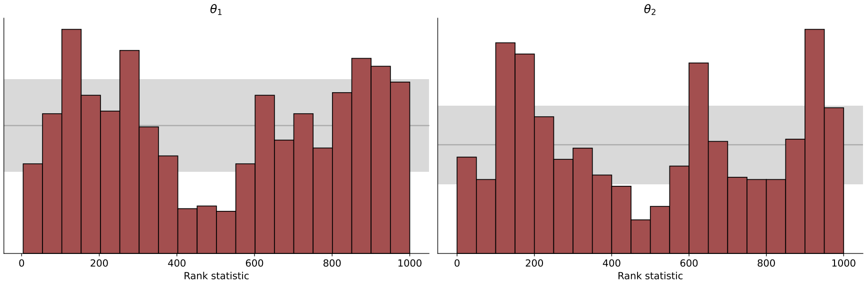


Figure 7: Two Moons: posterior samples for Energy Score trained with $m = 20$ and reference posterior samples. Diagonal panels represent univariate marginals, while off-diagonal panels represent bivariate marginals. A similar graph for GAN can be found in the supplementary material in Ramesh et al. (2022).



(a) GAN



(b) Energy Score, $m = 20$

Figure 8: Two Moons: Simulation-Based Calibration results represented as rank histograms; for each dimension of $\boldsymbol{\theta}$, the corresponding histogram represents the distribution of the rank of the true parameter value in a set of samples from the approximate posterior. If the approximate posterior is calibrated, histogram bars should be in the grey region with 99% probability.

Table 14: Two Moons: R^2; larger is better.

	GAN	Energy 3	Energy 5	Energy 10	Energy 20	Kernel 3	Kernel 5	Kernel 10	Kernel 20
1000	0.50 ± 0.01	0.49 ± 0.01	0.50 ± 0.01	0.50 ± 0.01	0.51 ± 0.01	0.48 ± 0.01	0.49 ± 0.01	0.48 ± 0.01	0.49 ± 0.01
10000	0.49 ± 0.01	0.50 ± 0.01	0.51 ± 0.01	0.51 ± 0.01	0.51 ± 0.01	0.50 ± 0.01	0.50 ± 0.01	0.50 ± 0.01	0.50 ± 0.01
100000	0.51 ± 0.01	0.50 ± 0.01	0.50 ± 0.01	0.50 ± 0.01	0.51 ± 0.01	0.50 ± 0.01	0.51 ± 0.01	0.50 ± 0.01	0.50 ± 0.01

Table 15: Tow Moons: runtime in seconds; recall that GAN was trained on GPU while the SR methods were trained on a single CPU.

	GAN	Energy 3	Energy 5	Energy 10	Energy 20	Kernel 3	Kernel 5	Kernel 10	Kernel 20
1000	4799	578	690	759	896	585	613	651	852
10000	8163	1775	1917	2415	3228	1708	1883	2329	3267
100000	30232	9266	9388	9903	10805	9283	9479	9859	10902

Table 16: Two Moons: epoch at which early stopping occurred; the max number of training epochs was 20000.

	GAN	Energy 3	Energy 5	Energy 10	Energy 20	Kernel 3	Kernel 5	Kernel 10	Kernel 20
1000	20000	20000	20000	20000	20000	20000	20000	20000	20000
10000	20000	20000	20000	20000	20000	20000	20000	20000	20000
100000	20000	20000	20000	20000	20000	20000	20000	20000	20000

F.3 Shallow Water Model

In Figure 9, we show results, analogously to what done in Figure 2, for all methods. Table 17 reports the different performance metrics, the runtime and the early stopping epoch for all methods. Finally, Figure 10 reports Simulation-Based Calibration results for all SR methods.

Table 17: Shallow Water model: performance metrics, runtime and early stopping epoch for all methods. We do not train GAN from scratch but rather relied on the trained network obtained in Ramesh et al. (2022). The training time we report here is what is mentioned in Ramesh et al. (2022), which used two GPUs for training (in contrast, we used a single GPU for the SR methods). For the same reason, we do not report the epoch at which GAN training was early stopped.

	RMSE ↓	Cal. Err. ↓	R^2 ↑	Runtime (sec)	Early stopping epoch
Energy	0.05 ± 0.01	0.03 ± 0.02	0.87 ± 0.05	51328	10400
Energy patched 10 20	0.05 ± 0.01	0.03 ± 0.02	0.89 ± 0.05	60017	12400
Energy patched 5 10	0.06 ± 0.01	0.03 ± 0.02	0.86 ± 0.06	49626	9600
Kernel	0.06 ± 0.01	0.11 ± 0.05	0.84 ± 0.06	39608	7800
Kernel patched 10 20	0.06 ± 0.01	0.09 ± 0.04	0.86 ± 0.06	47642	9000
Kernel patched 5 10	0.06 ± 0.01	0.09 ± 0.04	0.86 ± 0.06	44590	9200
GAN	0.07 ± 0.01	0.12 ± 0.09	0.78 ± 0.05	≈345600	-

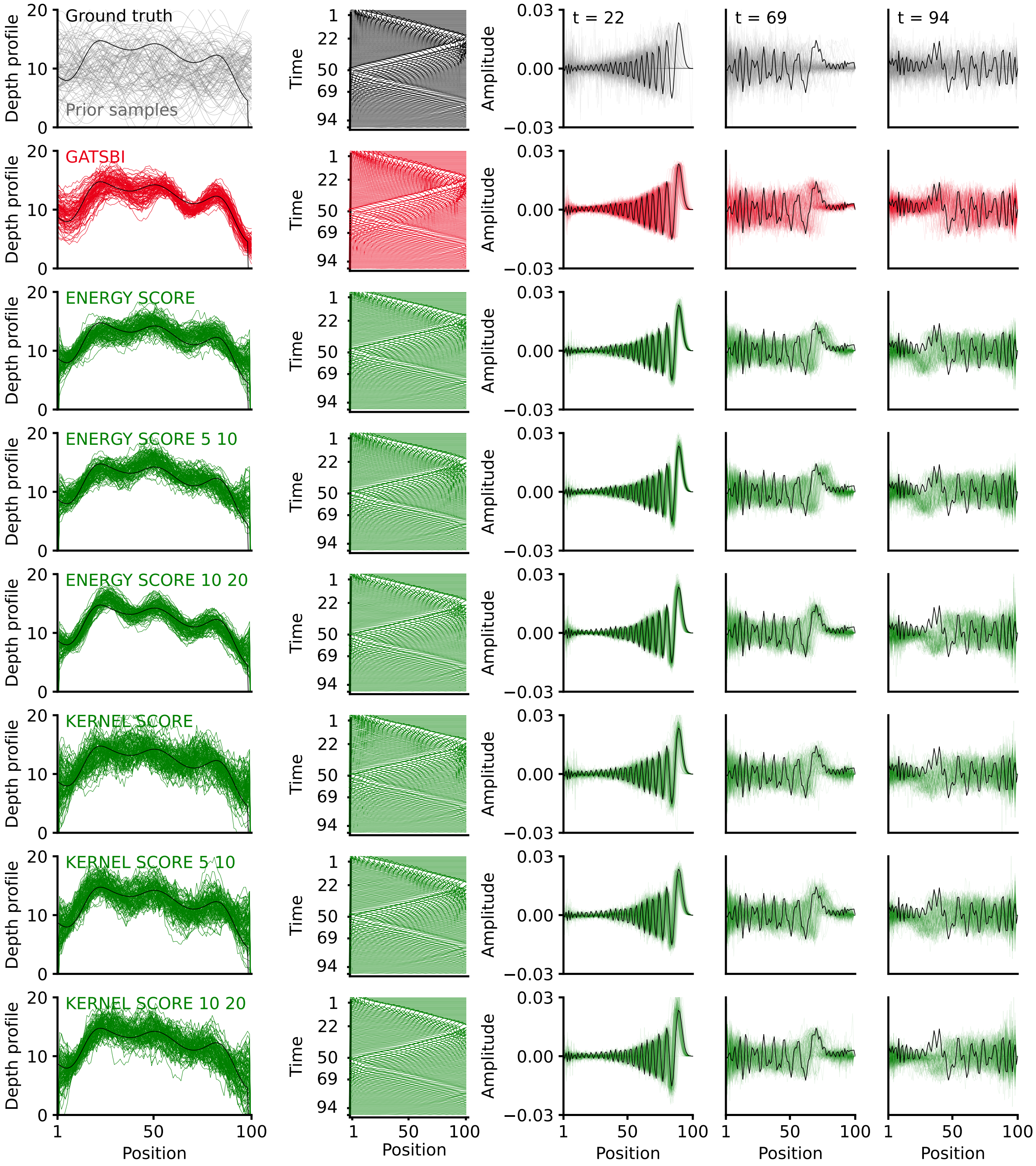
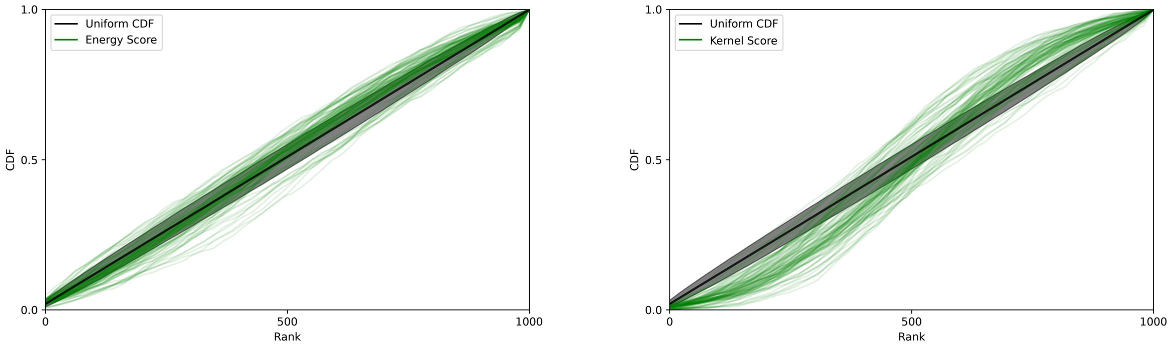
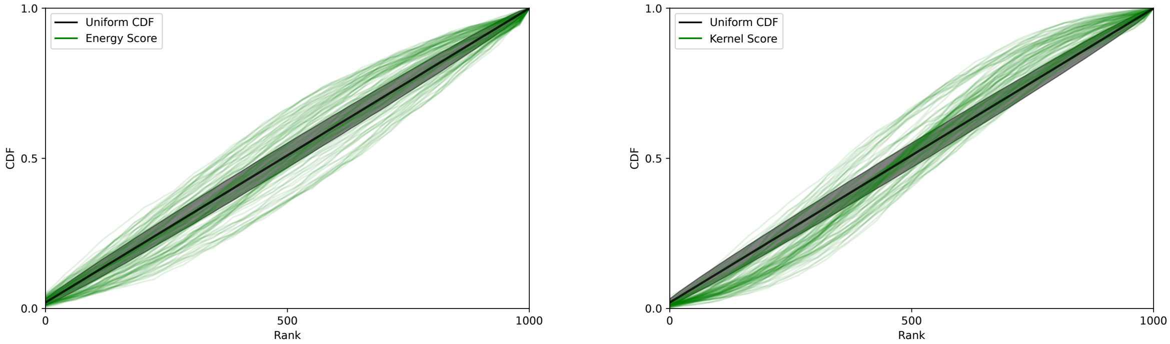


Figure 9: Shallow water model: inference results with all methods. See Figure 2 for a description of the different panels.



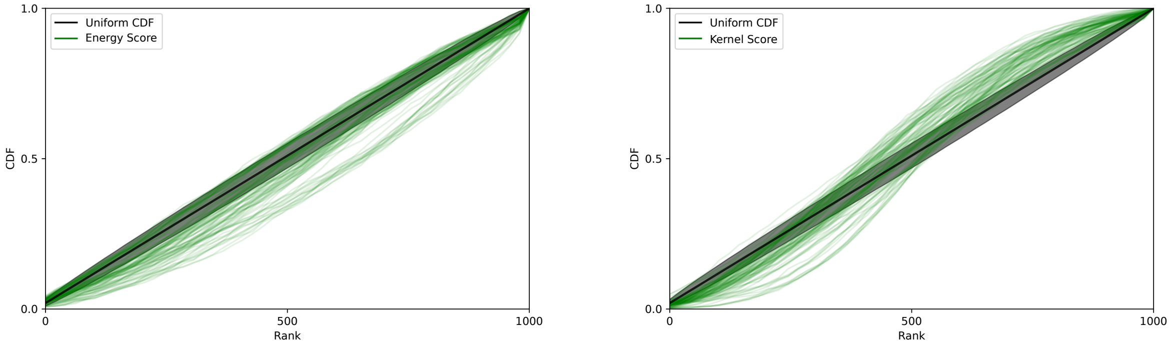
(a) Energy Score

(b) Kernel Score



(c) Energy Score patched 5, 10

(d) Kernel Score patched 5, 10



(e) Energy Score patched 10, 20

(f) Kernel Score patched 10, 20

Figure 10: Shallow Water model: Simulation Based Calibration for all SR methods. Each line corresponds to a single dimension of $\boldsymbol{\theta}$ and represents the CDF of the rank of the true parameter value with respect to a set of posterior samples. A calibrated posterior implies uniform CDF (diagonal black line, with associated 99% confidence region for the considered number of samples in gray).

F.4 Camera model

In Figure 11, we show results, analogously to what is done in Figure 4, for all methods. Table 18 reports the different performance metrics, runtime, and early stopping epoch for all methods.

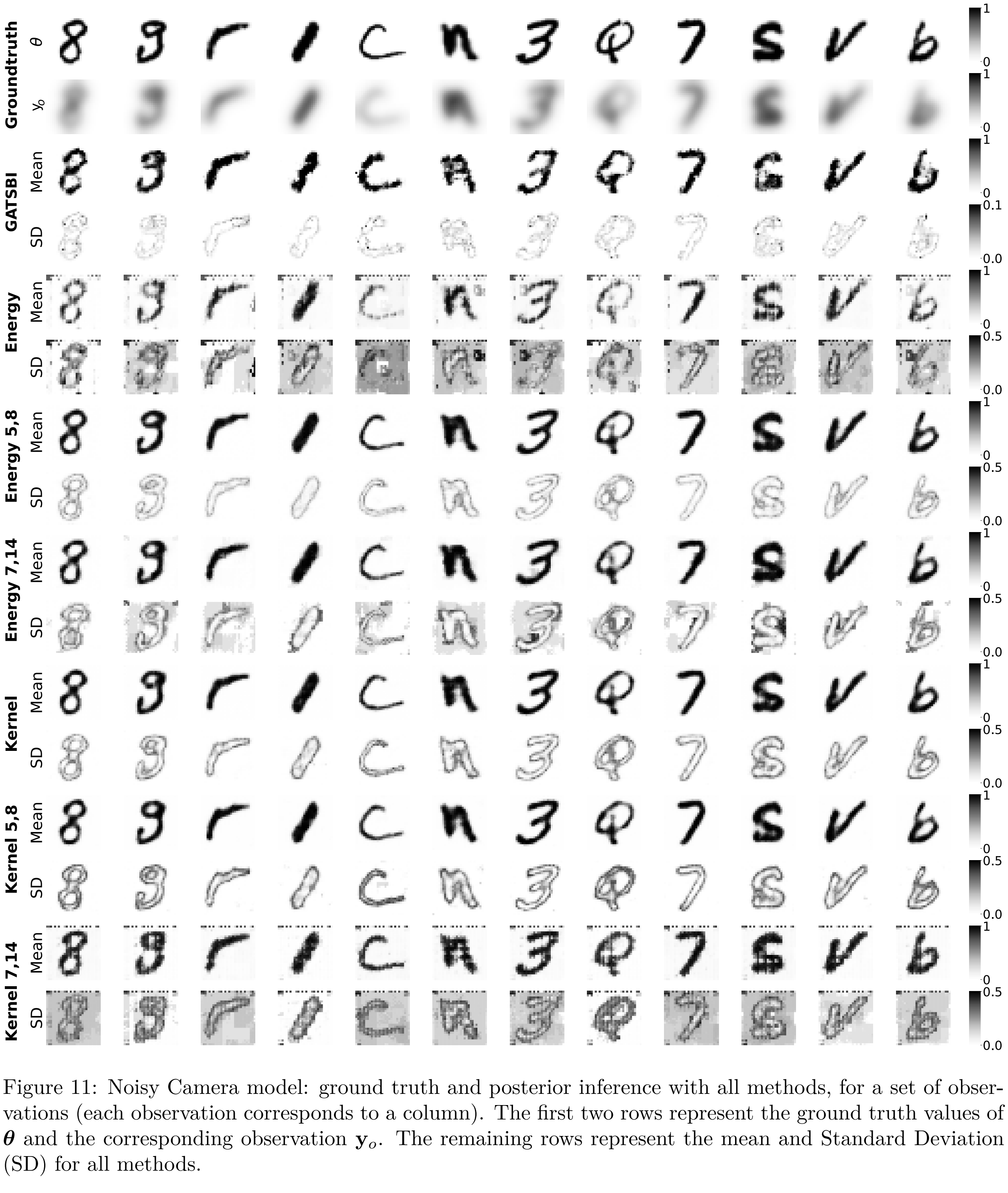


Figure 11: Noisy Camera model: ground truth and posterior inference with all methods, for a set of observations (each observation corresponds to a column). The first two rows represent the ground truth values of $\boldsymbol{\theta}$ and the corresponding observation $\mathbf{y}_o$. The remaining rows represent the mean and Standard Deviation (SD) for all methods.

Table 18: Noisy Camera model: performance metrics, runtime and early stopping epoch for all methods.

	RMSE ↓	Cal. Err. ↓	R^2 ↑	Runtime (sec)	Early stopping epoch
GAN	0.25 ± 0.19	0.50 ± 0.00	-23.94 ± 366.08	45398	3600
Energy	0.08 ± 0.05	0.36 ± 0.12	-24.39 ± 450.13	24555	4200
Energy patched 5 8	0.06 ± 0.05	0.36 ± 0.12	-2.14 ± 55.86	22633	4000
Energy patched 7 14	0.07 ± 0.05	0.37 ± 0.12	-10.33 ± 227.38	24033	3600
Kernel	0.06 ± 0.05	0.32 ± 0.15	-7.22 ± 164.26	21862	3200
Kernel patched 5 8	0.07 ± 0.05	0.36 ± 0.12	-10.29 ± 222.12	22545	3200
Kernel patched 7 14	0.10 ± 0.06	0.38 ± 0.11	-144.56 ± 2952.80	20605	3600