APPENDIX

A PROOFS

A.1 PROOF OF CONVERGENCE FOR GATSBI

Proposition 1. *Given a fixed generator f_ϕ, the discriminator $\mathrm{D}_{\psi^*}(\theta, x)$ maximising equation 2 satisfies*

$$\mathrm{D}_{\psi^*}(\theta, x) = \frac{p(\theta|x)}{p(\theta|x) + q_\phi(\theta|x)},$$

and the corresponding loss function for the generator parameters is the Jensen-Shannon divergence (JSD) between the true and approximate posterior:

$$L_{\psi^*}(\phi) = 2\,\mathrm{JSD}(p(\theta|x) \;||\; q_\phi(\theta|x)) - \log\ 4$$

Proof. We start with equation 2. The proof proceeds as for Proposition 1 and 2 in Goodfellow et al. (2014). For convenience, we elide the arguments of the various quantities, so that q_ϕ denotes $q_\phi(\theta|x)$, p denotes $p(\theta|x)$ and D_ψ denotes $\mathrm{D}_\psi(\theta, x)$.

$$\begin{aligned}
L(\phi, \psi) &= \mathbb{E}_{p(x)}\Big[\mathbb{E}_p \log \mathrm{D}_\psi + \mathbb{E}_{q_\phi} \log\left(1 - \mathrm{D}_\psi\right)\Big] \\
&= \mathbb{E}_{p(x)}\left[\int p \log \mathrm{D}_\psi \ \mathrm{d}\theta + \int q_\phi \log(1 - \mathrm{D}_\psi)\ \mathrm{d}\theta\right] \\
&= \mathbb{E}_{p(x)}\Big[\int \Big(p \log \mathrm{D}_\psi + q_\phi \log(1 - \mathrm{D}_\psi)\Big)\ \mathrm{d}\theta\Big].
\end{aligned}$$

For any function $g(x) = a \log x + b \log(1-x)$, where $a, b \in \mathbb{R}^2 \setminus \{0, 0\}$ and $x \in (0, 1)$, the maximum of the function is at $x = {}^a\!/_{a+b}$. Hence, $L(\phi, \psi)$, for a fixed ϕ, achieves it's maximum at:

$$\mathrm{D}_{\psi^*} = \frac{p}{p + q_\phi}$$

Note that $p(x)$ drops out of the expression for D_{ψ^*} since it is common to both terms in $L(\phi, \psi)$. Plugging this into equation 2 and dropping the expectation over x without loss of generality:

$$\begin{aligned}
L_{\psi^*}(\phi) &= \mathbb{E}_p\ \log \frac{p}{p + q_\phi} + \mathbb{E}_{q_\phi} \log\ \frac{q_\phi}{p + q_\phi} \\
&= \mathbb{E}_p\ \log \frac{2\,p}{p + q_\phi} + \mathbb{E}_{q_\phi} \log \frac{2\,q_\phi}{p + q_\phi} - \log\ 4 \\
&= 2\,\mathrm{JSD}(p \;||\; q_\phi) - \log\ 4.
\end{aligned}$$

The JSD is always non-negative, and zero only when $p(\theta|x) = q_\phi(\theta|x)$. Thus, the global minimum $L_{\psi^*}(\phi^*) = -\log\ 4$ is achieved only when the GAN generator converges to the ground-truth posterior $p(\theta|x)$. □

A.2 PROOF FOR GATSBI MAXIMIZING ELBO LOSS

Proposition 2. *If ϕ^* and ψ^* denote the Nash equilibrium of a min-max game defined by equation 2 then ϕ^* also maximises the evidence lower bound of a VAE with a fixed decoder i.e.,*

$$\phi^* = \arg\max_\phi L_{\mathrm{E}}(\phi) \tag{8}$$

$$= \arg\max_\phi\ \mathbb{E}_{p(x)}\mathbb{E}_{q_\phi(\theta|x)}\left(\log \frac{\pi(\theta)}{q_\phi(\theta|x)} + \log p(x|\theta)\right) \tag{9}$$

Proof. The proposition is trivially true if $q_{\phi^*}(\theta|x) = p(\theta|x)$, i.e., $p(\theta|x)$ is the true minimum of both the JSD and D_{KL}. However, we here show that the equivalence holds even when $q_{\phi^*}(\theta|x)$ is *not*

the true posterior, e.g. if $q_\phi(\theta|x)$ belongs to a family of distributions that does not include the true posterior.

Given that ϕ^* and ψ^* denote the Nash equilibrium in equation 2, we know from Proposition 1 that the optimal discriminator is given by

$$\mathrm{D}_{\psi^*}(\theta, x) = \frac{p(\theta|x)}{p(\theta|x) + q_{\phi^*}(\theta|x)}.$$

To lighten notation, we elide the parameters of the networks and their arguments, denoting $D_{\psi^*}(\theta|x)$ as D_{ψ^*}, $q_{\phi^*}(\theta|x)$ as q^*, and so on. If we plug D_{ψ^*} into Eq. 2, we have

$$\begin{aligned}\phi^* &= \arg\min_\phi \ \mathbb{E}_{p(\theta|x)} \ \log \ \mathrm{D}_{\psi^*} + \mathbb{E}_{q_\phi}\Big(\log\big(1 - \mathrm{D}_{\psi^*}\big)\Big) \\ &= \arg\min_\phi \mathbb{E}_{q_\phi} \ \log \ \big(1 - \mathrm{D}_{\psi^*}\big) \end{aligned} \tag{10}$$

Note that this is true *only* at the Nash equilibrium, where D_{ψ^*} is a function of q_{ϕ^*} and *not* q_ϕ. This allows us to drop the first term from the equation. In other words, if we switch out q_{ϕ^*} with any other q_ϕ in the expectation, equation 10 is no longer minimum w.r.t ϕ, even though D_{ψ^*} is optimal.

Let us define $\mathrm{D}_\psi := \sigma(R_\psi)$, where $\sigma(\cdot) := {}^1\!/_{1 + \exp(-\,\cdot)}$. Then from equation 3,

$$\sigma(R_{\psi^*}) = \frac{p(\theta|x)}{p(\theta|x) + q_{\phi^*}} \implies \frac{1}{1 + \mathrm{e}^{-R_{\psi^*}}} = \frac{1}{1 + {}^{q_{\phi^*}}\!/_{p(\theta|x)}}$$

Comparing the l.h.s. and r.h.s. of the equation above, we get

$$R_{\psi^*} = \log \frac{p(\theta|x)}{q_{\phi^*}}. \tag{11}$$

Since both $\log$ and σ are monotonically increasing functions, we also have from equation 10:

$$\begin{aligned}\phi^* &= \arg\min_\phi \mathbb{E}_{q_\phi} \ \log\big(1 - \sigma(R^*)\big) \\ &= \arg\min_\phi \mathbb{E}_{q_\phi} \ \big(1 - \sigma(R_{\psi^*})\big) \\ &= \arg\max_\phi \mathbb{E}_{q_\phi} \ \sigma(R_{\psi^*}) \\ &= \arg\max_\phi \mathbb{E}_{q_\phi}(R_{\psi^*}) \end{aligned} \tag{12}$$

In other words, q_{ϕ^*} maximises the function $\mathbb{E}_{q_\phi}(R_{\psi^*})$. Now, to prove equation 9, we need to show that $L_{\mathrm{E}}(\phi) < L_{\mathrm{E}}(\phi^*) \ \forall \ \phi \neq \phi^*$.

$$\begin{aligned} L_{\mathrm{E}}(\phi) &= \mathbb{E}_{p(x)}\mathbb{E}_{q_\phi}\left(\log \pi(\theta) - \log q_\phi + \log p(x|\theta)\right) \\ &= \mathbb{E}_{p(x)}\mathbb{E}_{q_\phi}\left(\log \frac{p(\theta|x)}{q_\phi} + \log \frac{p(x|\theta)\pi(\theta)}{p(\theta|x)}\right) \\ &= \mathbb{E}_{p(x)}\mathbb{E}_{q_\phi}\left(\log \frac{p(\theta|x)}{q_\phi} + \log p(x)\right) \\ &= \mathbb{E}_{p(x)}\mathbb{E}_{q_\phi}\left(\log \frac{p(\theta|x)}{q_{\phi^*}} + \log p(x)\right) - \mathbb{E}_{p(x)}(D_{\mathrm{KL}}(q_\phi||q_{\phi^*})) \\ &< \ \mathbb{E}_{p(x)}\mathbb{E}_{q_\phi}\left(\log \frac{p(\theta|x)}{q_{\phi^*}} + \log p(x)\right) \\ &= \mathbb{E}_{p(x)}\mathbb{E}_{q_\phi}\left(R_{\psi^*} + \log p(x)\right) && \text{from equation 11} \\ &< \ \mathbb{E}_{p(x)}\mathbb{E}_{q_{\phi^*}}\left(R_{\psi^*}\right) + \mathbb{E}_{p(x)}\mathbb{E}_{q_\phi} \log p(x) && \text{from equation 12} \\ &= \mathbb{E}_{p(x)}\mathbb{E}_{q_{\phi^*}}\left(R_{\psi^*} + \log p(x)\right) + \mathbb{E}_{p(x)}\mathbb{E}_{q_\phi} \log p(x) - \mathbb{E}_{p(x)}\mathbb{E}_{q_{\phi^*}} \log p(x) \\ &= \mathbb{E}_{p(x)}\mathbb{E}_{q_{\phi^*}}\left(\log \frac{p(\theta|x)}{q_{\phi^*}} + \log p(x)\right) \\ &= L_{\mathrm{E}}(\phi^*) \end{aligned}$$

$$\implies L_{\mathrm{E}}(\phi) < L_{\mathrm{E}}(\phi^*)$$

Hence, the approximate posterior obtained by optimising the GAN objective also maximises the evidence lower bound of the corresponding VAE. □

A.3 CONNECTION BETWEEN LFVI, DEEP POSTERIOR SAMPLING AND GATSBI

Adversarial inference approaches maximise the Evidence Lower Bound (ELBO) equation 6 to train a VAE, and use a discriminator to approximate intractable ratios of densities in the loss (see Table 1). Likelihood-free variational inference (Tran et al., 2017, LFVI) is one such method.

Table 1: Comparison of adversarial inference algorithms: BiGAN (Donahue et al., 2019), ALI (Dumoulin et al., 2016), AAE (Makhzani et al., 2015), AVB (Mescheder et al., 2017), and LFVI (Tran et al., 2017).

ALGORITHM	DISCRIMINATOR RATIO	GENERATOR LOSS FUNCTION
BIGAN, ALI	$p_\alpha(x\|u)p(u)/q_\phi(u\|x)p(x)$	$\mathrm{JSD}(p_\alpha(u,x)\|\|q_\phi(u,x))$
AAE	$p(u)/q_\phi(u)$	$\mathrm{JSD}(p(u)\|\|q_\phi(u))$
AVB	$p(u)p(x)/q_\phi(u\|x)p(x)$	$D_{\mathrm{KL}}(q_\phi(u\|x)\|\|p(u))$
LFVI	$p(u\|x)p(x))/q_\phi(u\|x)p(x)$	$D_{\mathrm{KL}}(q_\phi(u\|x)\|\|p(u\|x))$
GATSBI	$p(u\|x)p(x)/q_\phi(u\|x)p(x)$	$\mathrm{JSD}(p(u\|x)\|\|q_\phi(u\|x))$

In the most general formulation, LFVI learns a posterior over both latents u and global parameters β which are latents shared across multiple observations i.e., $q_\phi(z, \beta|x)$ and maximises the ELBO given by

$$L_{\mathrm{LF}}(\phi) = \mathbb{E}_{q_\phi(\beta)} \log \frac{p(\beta)}{q_\phi(\beta)} + \mathbb{E}_{q_\phi(u|x)p'(x)} \log \frac{p(x|u)p(u)}{q_\phi(u|x)p'(x)} + \text{const.} \tag{13}$$

where $p'(x)$,[3] an *empirical distribution* over observations, and *not* necessarily $p(x)$, the marginal likelihood of the simulator. A discriminator, $\mathrm{D}_\psi(x, u)$,[4] is trained with the cross-entropy loss to approximate the intractable ratio $\frac{p(x|u)p(u)}{q_\phi(u|x)p'(x)}$ in the second term. Using the nomenclature from Huszár (2017), we note that D_ψ is *joint-contrastive*: it simultaneously discriminates between tuples $(x, u) \sim p(x|u)p(u)$ and $(\hat{x}, \hat{u}) \sim q_\phi(\hat{u}|\hat{x})p'(\hat{x})$. GATSBI, by contrast, is *prior-contrastive*: its discriminator only discriminates between parameters θ, given a fixed x.

However, when $p'(x) = p(x)$ and β is constant, the LFVI discriminator becomes prior-contrastive, equation 13 is a function of both the discriminator parameters ψ and generator parameters ϕ, and it differs from GATSBI only by a single term i.e., from equation 2 and equation 13 and ignoring the constant:

$$L_{\mathrm{GATSBI}}(\phi, \psi) = -L_{\mathrm{LF}}(\phi, \psi) + \mathbb{E}_{q_\phi(u|x)p(x)} \log \mathrm{D}_\psi(x, u) \tag{14}$$

The second term corresponds to the non-saturating GAN loss (Goodfellow et al., 2014). In this

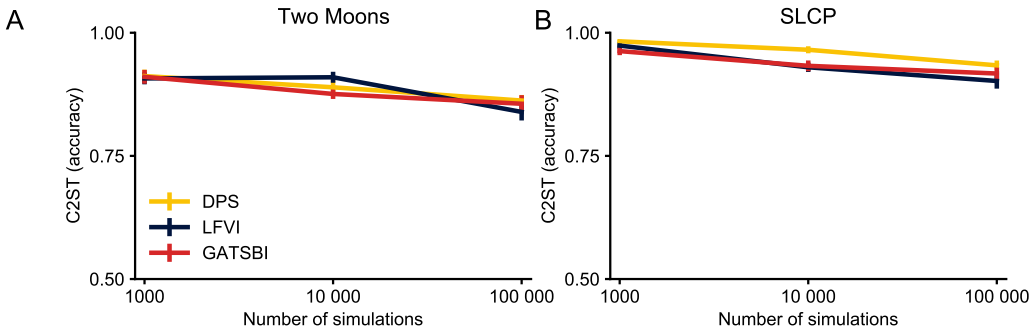

Figure 6: **GATSBI, LFVI and Deep Poseterior Sampling (DPS) on benchmark problems.** Mean C2ST score (± standard error of the mean) for 10 observations each. A. On Two Moons, the C2ST scores for GATSBI (red), LFVI (navy) and Deep Posterior Sampling (DPS, yellow) are qualitatively similar across all simulation budgets and on B. SLCP, DPS is slightly worse than LFVI and GATSBI.

[3] In Tran et al. (2017), $p'(x)$ is denoted $q(x)$

[4] denoted $r_\psi(x, u)$ in Tran et al. (2017)

setting, with an optimal discriminator, GATSBI minimises a JSD, whereas LFVI minimises the reverse D_{KL}.

Adler and Öktem (2018) introduce Deep Posterior Sampling which also implements an adversarial algorithm for posterior estimation. The set-up is similar to GATSBI, but GANs are trained using a Wasserstein loss as in Arjovsky et al. (2017). The Wasserstein loss imposes stronger conditions on the GAN networks in order for the generator to recover the target distribution, i.e., the discriminator has to be 1-Lipschitz and the generator K-Lipschitz (Arjovsky et al., 2017; Qi, 2018). However, the JSD loss function allowed us to outline GATSBI's connection to adversarial VAEs and subsequently its advantages for SBI (see Sec. 2.4, Suppl. Sec. A.2 and the discussion above). Whether this would also be possible with the Wasserstein loss remains a subject for future work. Nevertheless, the sequential extension of GATSBI using the energy-based correction (see Sec. 2.5 and Supp. Sec. A.4) could in principle also be used with the Wasserstein metric.

Mescheder et al. (2018) state that the WGAN converges only when the discriminator minimizes the Wasserstein metric at every step, which does not happen in practice. Fedus et al. (2017) argue that a GAN generator does not necessarily minimise a JSD at *every* update step since the discriminator is optimal only in the limit of infinite data. Hence, neither asymptotic property can be used to reason about GAN behaviour in practice. As a consequence, it is difficult to predict the conditions under which LFVI, or Deep Posterior Sampling would outperform GATSBI, or vice-versa. Nevertheless, given the same networks and hyperperparameters (with slight modifications to the discriminator for Deep Posterior Sampling, see App. D.1 for details), we found empirically that LFVI, Deep Posterior Sampling and GATSBI are qualitatively similar on Two Moons, and that Deep Posterior Sampling is slightly worse that the other two algorithms on SLCP: i.e., there is no advantage to using one algorithm over the other on the problems investigated (see Fig. 6).

A.4 Sequential GATSBI

For many SBI applications, the density estimator is only required to generate good posterior samples for a particular experimentally observed data x_{o}. This can be achieved by training the density estimator using samples θ from a proposal prior $\tilde{\pi}(\theta)$ instead of the prior $\pi(\theta)$. The proposal prior ideally produces parameters θ that are localised around the modes of $p(\theta|x_{\mathrm{o}})$ and can guide the density estimator towards inferring a posterior that is accurate for $x \approx x_{\mathrm{o}}$. If we replace the true prior $\pi(\theta)$ with a proposal prior $\tilde{\pi}(\theta) = q_\phi(\theta|x_{\mathrm{o}})$, i.e., the posterior estimated by GATSBI, and sample from the respective distributions

$$\theta, x \sim \tilde{\pi}(\theta)p(x|\theta),$$

the corresponding GAN loss (from equation equation 2) is:

$$\tilde{L}(\phi, \psi) = \mathbb{E}_{\tilde{\pi}(\theta)p(x|\theta)p(z)} \left[\log \mathrm{D}_\psi(\theta, x) + \log(1 - \mathrm{D}_\psi(f_\phi(z, x), x)) \right] \tag{15}$$

$$= \mathbb{E}_{\tilde{p}(\theta|x)\tilde{p}(x)} \log \mathrm{D}_\psi(\theta, x) + \mathbb{E}_{q_\phi(\theta|x)\tilde{p}(x)} \log(1 - \mathrm{D}_\psi(\theta, x)). \tag{16}$$

This loss would allow us to obtain a generator that produces samples θ that are likely to generate outputs x close to x_o, when plugged into the simulator. However, Proposition 1 in Appendix A.1 shows that this loss would result in $q_\phi(\theta|x)$ converging to the proposal posterior $\tilde{p}(\theta|x)$ rather than the ground-truth posterior $p(\theta|x)$. In order to learn a conditional density that is accurate for $x \approx x_{\mathrm{o}}$ but nevertheless converges to the correct posterior, we need to correct the approximate posterior for the bias due to the proposal prior. We outline three different approaches to this correction step:

Using energy-based GANs Although it is possible to use correction factors directly in the GATSBI loss function, as we outline in the next section, these corrections can lead to unstable training (Papamakarios et al., 2019). Here, we outline an approach in which we train on samples from the proposal prior without explicitly introducing correction factors into the loss function. Instead, we change the setup of the generator to produce 'corrected' samples, which are then used to compute the usual cross-entropy loss, and finally we train the discriminator and generator.

Let us start by introducing the correction factor $\omega(\theta, x) = \frac{\pi(\theta)}{\tilde{\pi}(\theta)} \frac{\tilde{p}(x)}{p(x)}$, such that

$$p(\theta|x) = \tilde{p}(\theta|x)\, \omega(\theta, x). \tag{17}$$

In the original formulation of GATSBI, sampling from $q_\phi(\theta|x)$ entails sampling latents $z \sim p(z)$, and transforming them by a deterministic function $f_\phi(x, z)$ to get parameters θ (see equation equation 1).

Following the energy-based GAN formulation (Azadi et al., 2019; Che et al., 2020), we define an *intermediate* latent distribution $p_t(z)$:

$$p_t(z) = p(z)(\omega(f_\phi(x, z), x))^{-1}. \tag{18}$$

$p_t(z)$ is the distribution of latent variables that, when passed through the function $f_\phi(x, z)$, are most likely to produce samples from the *approximate proposal posterior* $\tilde{q}_\phi(\theta|x) = q_\phi(\theta|x)(\omega(\theta, x))^{-1}$. For GANs, $p(z)$ is typically a tractable distribution whose likelihood can be computed, and from which one can sample, and thus, we can use MCMC or rejection sampling to also sample from $p_t(z)$ (see Appendix D.1 for details). The resulting loss function is:

$$\tilde{L}(\phi, \psi) = \mathbb{E}_{\tilde{p}(\theta|x)\tilde{p}(x)}\ \log \mathrm{D}_\psi(\theta, x) + \mathbb{E}_{p_t(z)\tilde{p}(x)} \log(1 - \mathrm{D}_\psi(f_\phi(x, z), x)) = \mathbb{E}_{\tilde{p}(x)}\big[L_1 + L_2\big]. \tag{19}$$

Note that there are no explicit correction factors in the loss.

We now show that optimising the loss function equation 19 leads to the generator converging to the correct posterior distribution. Let us first focus on the second term L_2:

$$\begin{aligned}
L_2 &= \mathbb{E}_{p_t(z)} \log(1 - \mathrm{D}_\psi(f_\phi(x, z), x)) \\
&= \int p_t(z) \log(1 - \mathrm{D}_\psi(f_\phi(x, z), x)) \\
&= \int p(z)\ (\omega(f_\phi(x, z), x))^{-1}\ \log(1 - \mathrm{D}_\psi(f_\phi(x, z), x)) && \text{from equation 18} \\
&= \int q_\phi(\theta|x)\ (\omega(\theta, x))^{-1}\ \log(1 - \mathrm{D}_\psi(\theta, x)) && \text{reparam. trick} \\
&= \int \tilde{q}_\phi(\theta|x)\ \log(1 - \mathrm{D}_\psi(\theta, x)) \\
&= \mathbb{E}_{\tilde{q}_\phi(\theta|x)} \log(1 - \mathrm{D}_\psi(\theta, x)).
\end{aligned}$$

Thus, from Proposition 1, we can conclude that by optimising the loss function equation 19, $\tilde{q}_\phi(\theta|x) \to \tilde{p}(\theta|x)$. This implies that the generator network, which represents $q_\phi(\theta|x)$, converges to $p(\theta|x)$, since both the approximate and target proposal posteriors are related respectively to the approximate and true posteriors by the same multiplicative factor $\omega(\theta, x)$. Note that the generator is more accurate in estimating the posterior given x_o (or $x \approx x_\mathrm{o}$), i.e., $p(\theta|x_\mathrm{o})$ than given x far from x_o, since it is trained on samples from the proposal prior.

In practice, this scheme does produce improvements in the learned posterior. However, it is computationally expensive, because *every update* to the generator and discriminator requires a round of MCMC or rejection sampling to obtain the 'corrected' samples. Moreover, if we use the generator from the previous round as the proposal prior in the next round, we need to train a classifier to approximate $\omega(\theta, x)$ at every round.[5] Finally, this approach has additional hyperparameters that need to be tuned during GAN training, which could make it prohibitively difficult to use for most applications.

Below, we outline theoretical arguments for two additional approaches, although we only provide empirical results for the second approach.

Using importance weights Lueckmann et al. (2017) solve the problem of bias from using a proposal prior by introducing importance weights in their loss function. One can use the same trick for GATSBI,by introducing the importance weights $\omega(\theta, x) = \frac{\pi(\theta)}{\tilde{\pi}(\theta)} \frac{\tilde{p}(x)}{p(x)}$ into the loss defined in equation 16:

$$\tilde{L}(\phi, \psi) = \mathbb{E}_{\tilde{\pi}(\theta)p(x|\theta)p(z)} \left[\omega(\theta, x) \log \mathrm{D}_\psi(\theta, x) + \log(1 - \mathrm{D}_\psi(f_\phi(z, x), x))\right] = L_1 + L_2. \tag{20}$$

[5]Note that the correction factor could be computed in closed form if we had a generator with an evaluable density: we would not have to train a classifier to approximate it.

Optimising this loss allows $q_\phi(\theta|x)$ to converge to the true posterior $p(\theta|x)$. Let us focus on the first term L_1:

$$\begin{aligned}
L_1 &= \mathbb{E}_{\tilde{p}(\theta|x)\tilde{p}(x)}\omega(\theta,x)\log \mathrm{D}_\psi(\theta,x) \\
&= \int_x\int_\theta \tilde{p}(x)\tilde{p}(\theta|x)\frac{\pi(\theta)}{\tilde{\pi}(\theta)}\frac{\tilde{p}(x)}{p(x)}\log \mathrm{D}_\psi(\theta,x) \\
&= \int_x\int_\theta p(x|\theta)\tilde{\pi}(\theta)\frac{\pi(\theta)}{\tilde{\pi}(\theta)}\frac{\tilde{p}(x)}{p(x)}\log \mathrm{D}_\psi(\theta,x) \\
&= \int_x\int_\theta p(x|\theta)\pi(\theta)\frac{\tilde{p}(x)}{p(x)}\log \mathrm{D}_\psi(\theta,x) \\
&= \int_x\int_\theta p(x)p(\theta|x)\frac{\tilde{p}(x)}{p(x)}\log \mathrm{D}_\psi(\theta,x) \\
&= \int_x\int_\theta \tilde{p}(x)p(\theta|x)\log \mathrm{D}_\psi(\theta,x) \\
&= \mathbb{E}_{\tilde{p}(x)p(\theta|x)}\log \mathrm{D}_\psi(\theta,x).
\end{aligned}$$

Thus, from Proposition 1, we can conclude that by optimising the loss function equation 20, the generator $q_\phi(\theta|x)$ converges to the true posterior. However, the importance-weight correction could lead to high-variance gradients (Papamakarios et al., 2019). This would be particularly problematic for GANs, where the loss landscape for each network is modified with updates to its adversary, and there is no well-defined optimum. High-variance gradients could cause training to take longer or even prevent it from converging altogether.

Using inverse importance weights Since using importance weights in the loss can lead to high-variance gradients, we could instead consider using the inverse of the importance weights $(\omega(\theta,x))^{-1} = \frac{\tilde{\pi}(\theta)}{\pi(\theta)}\frac{p(x)}{\tilde{p}(x)}$ in the second term in equation 16:

$$\tilde{L}(\phi,\psi) = \mathbb{E}_{\tilde{p}(\theta|x)\tilde{p}(x)}\log \mathrm{D}_\psi(\theta,x) + \mathbb{E}_{q_\phi(\theta|x)\tilde{p}(x)}\ (\omega(\theta,x))^{-1}\ \log(1-\mathrm{D}_\psi(\theta,x)) = L_1 + L_2. \tag{21}$$

Optimising $\tilde{L}(\phi,\psi)$ from equation 21 will result in the generator approximating the true posterior at convergence. Focusing on the second term of the loss function L_2:

$$\begin{aligned}
L_2 &= \mathbb{E}_{q_\phi(\theta|x)\tilde{p}(x)}\ (\omega(\theta,x))^{-1}\ \log(1-\mathrm{D}_\psi(\theta,x)) \\
&= \iint \tilde{p}(x)q_\phi(\theta|x)\frac{\tilde{\pi}(\theta)}{\pi(\theta)}\frac{p(x)}{\tilde{p}(x)}\log(1-\mathrm{D}_\psi(\theta,x)) \\
&= \iint \tilde{p}(x)\tilde{q}_\phi(\theta|x)\log(1-\mathrm{D}_\psi(\theta,x)) \\
&= \mathbb{E}_{\tilde{p}(x)\tilde{q}_\phi(\theta|x)}\log(1-\mathrm{D}_\psi(\theta,x)).
\end{aligned}$$

Thus, from Proposition 1, we can conclude that by optimising the loss function equation 21, $\tilde{q}_\phi(\theta|x)$ converges to $\tilde{p}(\theta|x)$. Since $\tilde{q}_\phi(\theta|x)$ differs from $q_\phi(\theta|x)$ by the same factor as $\tilde{p}(\theta|x)$ from $p(\theta|x)$, i.e., $(\omega(\theta,x))^{-1}$ (see equation equation 17), this implies that $q_\phi(\theta|x) \to p(\theta|x)$.

B TRAINING ALGORITHMS

Algorithm 1 GATSBI

Input : prior $\pi(\theta)$, simulator $p(x|\theta)$, generator f_ϕ, discriminator D_ψ, learning rate λ
Output: Trained GAN networks f_{ϕ^*} and D_{ψ^*}
$\Theta = \{\theta_1, \theta_2, \ldots, \theta_n\} \overset{\text{i.i.d}}{\sim} \pi(\theta)$
$\mathbf{X} = \{x_1, x_2, \ldots, x_n\} \sim p(x_i|\theta_i)$
while not converged **do**
  for discriminator iterations **do**
    Sample mini-batch $\mathbf{X}_d$, Θ_d from $\mathbf{X}$, Θ
    $\mathbf{Z} \sim p(z)$, $\hat{\Theta}_d = f_\phi(\mathbf{Z}, \mathbf{X}_d)$
    $L = \sum_{\mathbf{X}_d} \left(\sum_{\Theta_d} \log \mathrm{D}_\psi(\Theta_d, \mathbf{X}_d) + \sum_{\hat{\Theta}_d} \log(1 - \mathrm{D}_\psi(\hat{\Theta}_d, \mathbf{X}_d)) \right)$
    $\psi \leftarrow \psi + \lambda \nabla_\psi L$
  end for
  for generator iterations **do**
    Sample mini-batch $\mathbf{X}_g$, Θ_g from $\mathbf{X}$, Θ
    $\mathbf{Z} \sim p(z)$, $\hat{\Theta}_g = f_\phi(\mathbf{Z}, \mathbf{X}_g)$
    $L = -\sum_{\mathbf{X}_g} \sum_{\hat{\Theta}_g} \log(1 - \mathrm{D}_\psi(\hat{\Theta}_g, \mathbf{X}_g))$
    $\phi \leftarrow \phi + \lambda \nabla_\phi L$
  end for
end while

Algorithm 2 Sequential GATSBI with energy-based correction

Input: $\pi(\theta)$, simulator $p(x|\theta)$, classifier ω = 1, observation x_o, f_ϕ, D_ψ, learning rate λ
Output: Trained GAN networks f_{ϕ^*} and D_{ψ^*}
for $i = 1 \cdots$ number of rounds **do**
  $\Theta_i = \{\theta_1, \theta_2, \ldots, \theta_n\} \overset{\text{i.i.d}}{\sim} \pi(\theta)$
  $\mathbf{X}_i = \{x_1, x_2, \ldots, x_n\} \sim p(x|\theta)$
  if $i > 1$ **then**:
    $\omega_\theta \leftarrow \max_{\omega_\theta} \log \sigma(\omega_\theta(\Theta_0)) + \log(1 - \sigma(\omega_\theta(\Theta_i)))$
    $\omega_x \leftarrow \max_{\omega_x} \log \sigma(\omega_x(\mathbf{X}_0)) + \log(1 - \sigma(\omega_x(\mathbf{X}_i)))$
    $\omega = \frac{\omega_\theta}{\omega_x}$
  end if
  while not converged **do**
    for discriminator iterations **do**
      Sample mini-batch $\mathbf{X}_d$, Θ_d from $\mathbf{X}_i$, Θ_i
      $\mathbf{Z} \sim p_t(\mathbf{Z}) = p(\mathbf{Z})(\omega(f_\phi(\mathbf{Z}, \mathbf{X}_d), \mathbf{X}_d))^{-1}$
      $\hat{\Theta}_d = f_\phi(\mathbf{Z}, \mathbf{X}_d)$
      $L = \sum_{\mathbf{X}_d} (\sum_{\Theta_d} \log \mathrm{D}_\psi(\Theta_d, \mathbf{X}_d) + \sum_{\hat{\Theta}_d} \log(1 - \mathrm{D}_\psi(\hat{\Theta}_d, \mathbf{X}_d)))$
      $\psi \leftarrow \psi + \lambda \nabla_\psi L$
    end for
    for generator iterations **do**
      Sample mini-batch $\mathbf{X}_g$, Θ_g from $\mathbf{X}_i$, Θ_i
      $\mathbf{Z} \sim p_t(\mathbf{Z}) = p(\mathbf{Z})(\omega(f_\phi(\mathbf{Z}, \mathbf{X}_g), \mathbf{X}_g))^{-1}$
      $\hat{\Theta}_g = f_\phi(\mathbf{Z}, \mathbf{X}_g)$
      $L = -\sum_{\mathbf{X}_g} \sum_{\hat{\Theta}_g} \log(1 - \mathrm{D}_\psi(\hat{\Theta}_g, \mathbf{X}_g))$
      $\phi \leftarrow \phi + \lambda \nabla_\phi L$
    end for
  end while
  $\pi(\theta) = f_\phi(\theta; x_\mathrm{o})$
end for

Algorithm 3 Sequential GATSBI with inverse importance weights

Input: $\pi(\theta)$, simulator $p(x|\theta)$, classifier ω = 1, observation x_{o}, f_ϕ, D_ψ, learning rate λ
Output: Trained GAN networks f_{ϕ^*} and D_{ψ^*}

for $i = 1 \cdots$ number of rounds **do** $\Theta_i = \{\theta_1, \theta_2, \ldots, \theta_n\} \overset{\text{i.i.d}}{\sim} \pi(\theta)$ $\mathbf{X}_i = \{x_1, x_2, \ldots, x_n\} \sim p(x|\theta)$
  if $i > 1$ **then**:
    $\omega_\theta \leftarrow \max_{\omega_\theta} \log \sigma(\omega_\theta(\Theta_0)) + \log(1 - \sigma(\omega_\theta(\Theta_i)))$
    $\omega_x \leftarrow \max_{\omega_x} \log \sigma(\omega_x(\mathbf{X}_0)) + \log(1 - \sigma(\omega_x(\mathbf{X}_i)))$
    $\omega = \frac{\omega_\theta}{\omega_x}$
  end if
  while not converged **do**
    for discriminator iterations **do**
      Sample mini-batch $\mathbf{X}_d$, Θ_d from $\mathbf{X}_i$, Θ_i
      $\mathbf{Z} \sim p(\mathbf{Z})$
      $\hat{\Theta}_d = f_\phi(\mathbf{Z}, \mathbf{X}_d)$
      $L = \sum_{\mathbf{X}_d}(\sum_{\Theta_d} \log \mathrm{D}_\psi(\Theta_d, \mathbf{X}_d) + \sum_{\hat{\Theta}_d}(\omega(\hat{\Theta}_d, \mathbf{X}_d))^{-1} \log(1 - \mathrm{D}_\psi(\hat{\Theta}_d, \mathbf{X}_d)))$
      $\psi \leftarrow \psi + \lambda \nabla_\psi L$
    end for
    for generator iterations **do**
      Sample mini-batch $\mathbf{X}_g$, Θ_g from $\mathbf{X}_i$, Θ_i
      $\mathbf{Z} \sim p(\mathbf{Z})$
      $\hat{\Theta}_g = f_\phi(\mathbf{Z}, \mathbf{X}_g)$
      $L = -\sum_{\mathbf{X}_g} \sum_{\hat{\Theta}_g}(\omega(\hat{\Theta}_g, \mathbf{X}_g))^{-1} \log(1 - \mathrm{D}_\psi(\hat{\Theta}_g, \mathbf{X}_g))$
      $\phi \leftarrow \phi + \lambda \nabla_\phi L$
    end for
  end while
  $\pi(\theta) = f_\phi(\theta; x_{\mathrm{o}})$
end for

C ADDITIONAL RESULTS

C.1 POSTERIORS FOR BENCHMARK PROBLEMS

We show posterior plots for the benchmark problems: SLCP (Fig. 7) and Two Moons (Fig. 8). In both figures, panels on the diagonal display the histograms for each parameter, while the off-diagonal panels show pairwise posterior marginals, i.e., 2D histograms for pairs of parameters, marginalised over the remaining parameter dimensions.

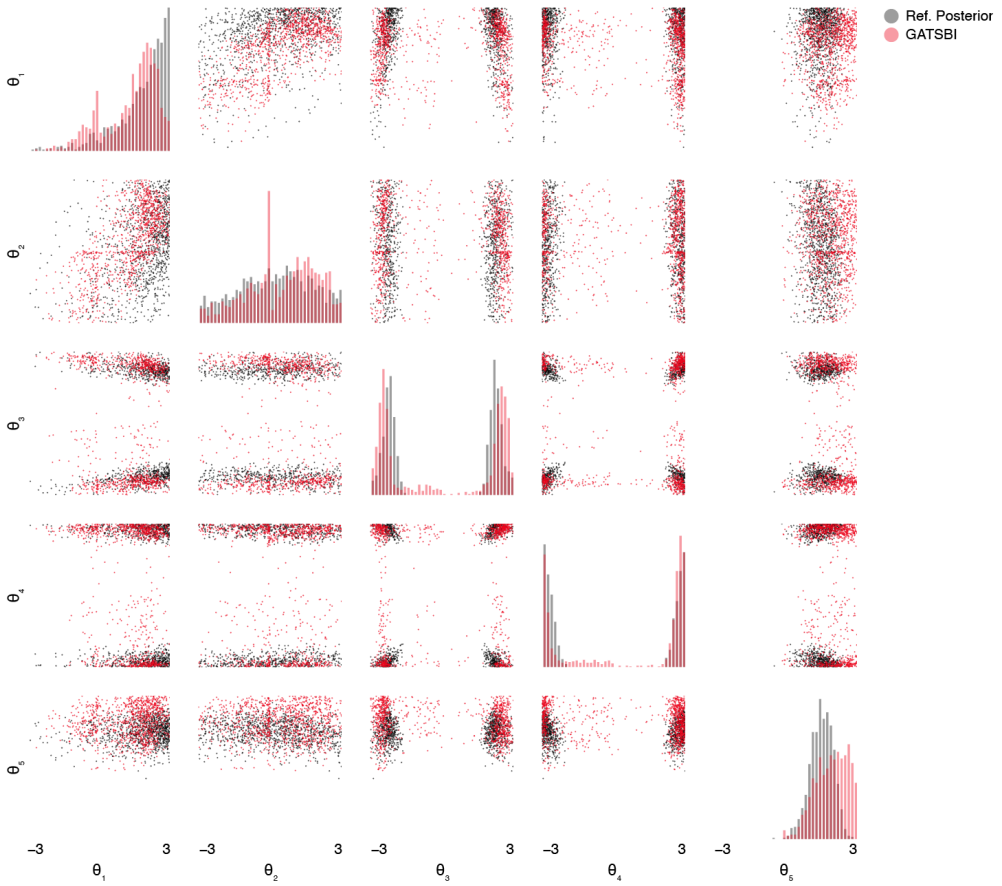

Figure 7: Inference for one test observation of the SLCP problem. Posterior samples for GATSBI trained on 100k simulations (red), and reference posterior samples (black). The GATSBI posterior samples cover well the disjoint modes of the posterior, although GATSBI sometimes produces samples in regions of low density in the reference posterior.

C.2 SEQUENTIAL GATSBI

We found that sequential GATSBI with the energy-based correction produced a modest improvement over amortised GATSBI for the Two Moons model with 1k and 10k simulation budgets, and no improvement at all with 100k (see Fig. 9). The inverse importance weights correction did not produce an improvement for any simulation budget. Sequential GATSBI performance was also sensitive to hyperparameter settings and network initialisation. We hypothesise that further improvement is possible with better hyperparameter or network architecture tuning.

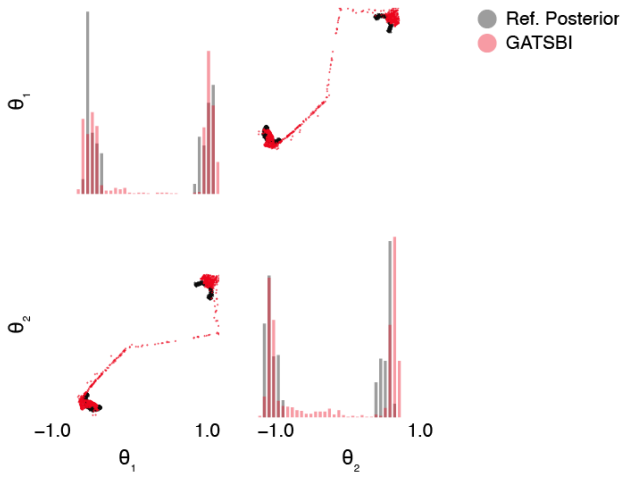

Figure 8: Inference for one test observation of the Two Moons problem. Posterior samples for GATSBI trained on 100k simulations (red), and reference posterior samples (black). GATSBI captures the global bi-modal structure in the reference posterior, but not the local crescent shape. It also generates some samples in regions of low density in the reference posterior.

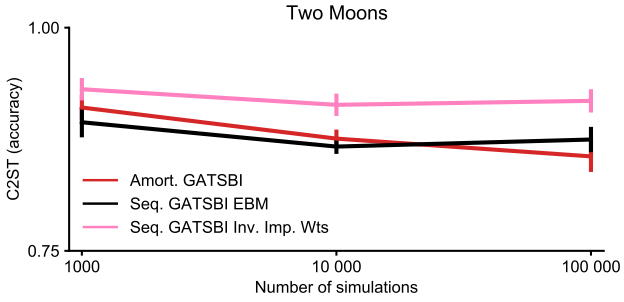

Figure 9: Sequential GATSBI performance for the Two Moons Model. The energy-based correction (EBM) results in a slight improvement over amortised GATSBI for 1k and 10k simulations, but the inverse importance weights correction does not.

D IMPLEMENTATION DETAILS

All networks and training algorithms were implemented in PyTorch (Paszke et al., 2019). We used Weights and Biases (Biewald, 2020) to log experiments with different hyperparameter sets and applications. We ran the high-dimensional experiments (camera model and shallow water model) on Tesla-V100 GPUs: the shallow water model required training to be parallelised across 2 GPUs at a time, and took about 4 days to converge and about 1.5 days for the camera model on one Tesla V100. We used RTX-2080Tis for the benchmark problems: the amortised GATSBI runs lasted a maximum of 1.5 days for the 100k budget; the sequential GATSBI runs took longer with the maximum being 8 days for the energy-based correction with a budget of 100k. On similar resources, NPE took about 1 day to train on the shallow water model, and 3 weeks and 2 days to train on the camera model. NPE took about 10 min for 100k simulations on both benchmark problems. SMC-ABC and rejection-ABC both took about 6s on the benchmark problems with a budget of 100k.

D.1 SIMPLE-LIKELIHOOD COMPLEX POSTERIOR (SLCP) AND TWO-MOONS

Prior and simulator For details of the prior and simulator, we refer to Lueckmann et al. (2021).

GAN architecture The generator was a 5-layer MLP with 128 hidden units in the first four layers. The final layer had output features equal to the parameter dimension of the problem. A leaky ReLU nonlinearity with slope 0.1 followed each layer. A noise vector sampled from a standard normal distribution (two-dimensional for Two Moons, and 5-dimensional for SLCP) was injected at the fourth layer of the generator. The generator received the observations as input to the first layer, used the first three layers to embed the observation, and multiplied the embedding with the injected noise at the fourth layer, which was then passed through the subsequent layers to produce an output of the same dimensions as the parameters. The discriminator was a 6-layer MLP with 2048 hidden units in the first five layers and the final layer returned a scalar output. A leaky ReLU nonlinearity with slope 0.1 followed each layer, except for the last, which was followed by a sigmoid nonlinearity. The discriminator received both the observations and the parameters sampled alternatively from the generator and the prior as input. Observations and parameters were concatenated and passed through the six layers of the network. For Deep Posterior Sampling, the discriminator did not have the final sigmoid layer.

Training details The generator and discriminator were trained in parallel for 1k, 10k and 100k simulations, with a batch size = $\min$(10% of the simulation budget, 1000). For each simulation budget, 100 samples were held out for validation. We used 10 discriminator updates for 1k and 10k simulation budgets, and 100 discriminator updates for the 100k simulation budget, per generator update. Note that the increase in discriminator updates for 100k simulations is intended to compensate for the reduced relative batch size i.e., 1000 batches = 0.01%. The networks were optimised with the cross-entropy loss. We used the Adam optimiser (Kingma and Ba, 2015) with learning rate=0.0001, β_1=0.9 and β_2=0.99 for both networks. We trained the networks for 10k, 20k and 20k epochs for the three simulation budgets respectively. To ensure stable training, we used spectral normalisation (Miyato et al., 2018) for the discriminator network weights. For the comparison with LFVI, we kept the architecture and hyperparameters the same as for GATSBI, but trained the generator to minimise $L(\phi) = \mathbb{E}_{q_\phi(\theta|x)}[\log \frac{1-\mathrm{D}_\psi(\theta,x)}{\mathrm{D}_\psi(\theta,x)}]$. Similarly, for Deep Posterior Sampling we kept the same architecture (minus the final sigmoid layer for the discriminator) and hyperparameters, but trained the generator and discriminator on the Wasserstein loss: $L(\phi, \psi) = \mathbb{E}_{p(\theta|x)}\, \mathrm{D}_\psi(\theta) - \mathbb{E}_{q_\phi(\theta|x)}\, \mathrm{D}_\psi(\theta)$.

Optimised hyperparameters for Two Moons model The generator was a 2-layer MLP with 128 hidden units in the first layer and 2 output features in the second layer (same as the parameter dimension). Each layer was followed by a leaky ReLU nonlinearity with slope 0.1. Two-dimensional white noise was injected into the second layer, after it was multiplied with the output of the first layer. The discriminator was a 4-layer MLP with 2048 hidden units in the first 3 layers each followed by a leaky ReLU nonlinearity of slope 0.1, and a single output feature in the last layer followed by a sigmoid nonlinearity. We trained the networks in tandem for approximately 10k, 50k and 25k epochs for the 1k, 10k and 100k simulation budgets respectively. There were 10 discriminator updates and 10 generator updates per epoch, with the batch size set to 10% of the simulation budget (also for

100k simulations). All other hyperparameters (learning rate, β_1, β_2, etc.) were the same as for the non-optimised architecture.

Hyperparameters for sequential GATSBI The architecture of the generator and discriminator were the same as for amortised GATSBI. We trained the networks for 2 rounds. In the first round, the networks were trained with the same hyperparameters as for amortised GATSBI, on samples from the prior: the only exceptions were the number of samples we held out for the 1k budget (10 instead of 100) and the number of discriminator updates per epoch for the 100k budget (10 instead of 100). Both exceptions were to ensure that there were always 10 discriminator updates per epoch, and speeding up training as much as possible. In the second round, the networks were trained using the energy-based correction with samples from the proposal prior, as well as samples from the prior used in the first round. All other hyperparameters were the same as for round one. The simulation budget was split equally across the two rounds, and the networks were trained anew for each of 10 different observations. The number of epochs was the same for the first and second round: 5k, 10k and 20k for the 1k, 10, and 100k budget respectively. We trained 2 classifiers at the beginning of round two: one to approximate the ratio $\frac{\pi(\theta)}{\tilde{\pi}(\theta)}$ and the other to approximate $\frac{p(x)}{\tilde{p}(x)}$. Both classifiers were 4-layer MLPs with 128 hidden units in each layer, and a ReLU nonlinearity following each layer. The classifiers were trained on samples from the proposal prior and prior, and the proposal marginal likelihood and likelihood respectively, using the MLPClassifier class with default hyperparameters (except for "max_iter"=5000) from scikit-learn (Pedregosa et al., 2011). For the energy-based correction, we used rejection sampling to sample from the corrected distribution $p_t(z)$: for a particular observation x, we sampled z $p(z) = \mathcal{N}(0, 1)$, evaluated the probability of acceptance $p = (\omega(f_\phi(z, x), z))^{-1}/M$, and accepted z if $u < p$ where u is a uniform random variable. To compute the scale factor M, we simply took the maximum value of $p(z)\omega(f_\phi(z, x), z))^{-1}$ within each batch. For the inverse importance weights correction, we computed $(\omega(\theta, x))^{-1}$, used it to calculate the loss as in Equation equation 21 for each discriminator and generator update in the second round.

D.2 SHALLOW WATER MODEL

Prior $\theta \sim \mathcal{N}(\mu\mathbf{1}_{100}, \Sigma),\ \theta \in \mathbb{R}^{100}$
$\mu = 10, \Sigma_{ij} = \sigma \exp(-(i-j)^2/\tau), \sigma = 15, \tau = 100.$

The values for $\boldsymbol{\mu}$ and Σ were chosen to ensure that the different depth profile samples produced discernible differences in the corresponding simulated surface waves, particularly in Fourier space. For example, combinations of μ values > 25 (deeper basins), σ values < 10 and τ values > 100 (smoother depth profiles) resulted in visually indistinguishable surface wave simulations.

Simulator

$$\mathbf{x}|\theta = f(\theta) + 0.25\boldsymbol{\epsilon}$$
$$\boldsymbol{\epsilon} = \begin{bmatrix} \epsilon_{1,1} & \cdots & \epsilon_{1,100} \\ \vdots & \ddots & \vdots \\ \epsilon_{200,1} & \cdots & \epsilon_{200,100} \end{bmatrix} \quad \epsilon_{ij} \sim \mathcal{N}(0, 1).$$

$f(\theta)$ is obtained by solving the 1D Saint-Venant equations on a 100-element grid, performing a 2D Fourier transform and stacking the the real and imaginary part to form a $2\times100\times100$-dimensional array.

The equations were solved using a semi-implicit solver (Backhaus, 1983) with a weight of 0.5 for each time level, implemented in Fortran (F90). The time-step size dt was set to 300s and the simulation was run for a total of 3600s. The grid spacing dx was 0.01, with dry cells at both boundaries using a depth of -10. We used a bottom drag coefficient of 0.001 and gravity=$9.81^m/_{s^2}$. An initial surface disturbance of amplitude 0.2 was injected at $x = 2$, to push the system out of equilibrium.

We chose to perform inference with observations in the Fourier domain for the following reason. Since waves are a naturally periodic delocalised phenomenon, it makes sense to run inference on their Fourier-transformed amplitudes, so that convolutional filters can pick up on localised features. We used the scipy fft2 package (Virtanen et al., 2020).

GAN architecture The generator network was similar to the DCGAN generator (Radford et al., 2015). There were five sequentially stacked blocks of the following form: a 2D convolutional layer, followed by a batch-norm layer and ReLU nonlinearity, except for the last layer, which had only a convolutional layer followed by a $\tanh$ nonlinearity. The observations input to the generator were of size batch size $\times$ 200 $\times$ 100. The input channels, output channels, kernel size, stride and padding for each of the six convolutional layers were as follows: 1 - (2, 512, 4, 1, 0), 2 - (512, 256, 4, 2, 1), 3 - (256, 128, 4, 2, 1), 4 - (128, 128, 4, 2, 1), 5 - (128, 1, 4, 2, 1). The final block was followed by a fully-connected readout layer that returned a 100-dimensional vector. Additionally, we sampled 25-dimensional noise, where each element of the array was drawn independently from a standard Gaussian. and added it to the output of the $\tanh$ layer, just before the readout layer.

The discriminator network was similar to the DGCAN discriminator: it contained embedding layers that mirrored the generator network minus the injected noise, followed by 4 fully-connected layers with 256 units each and a leaky ReLU nonlinearity of slope 0.2 after each fully-connected layer. The final fully-connected layer, however, was followed by a sigmoid nonlinearity. The discriminator received both the Fourier-transformed waves and a depth profile alternatively from the generator and the prior as input. The Fourier-transformed waves were passed through the embedding layers, the embedding was concatenated with the input depth profile and then passed through the fully-connected layers.

Training details The two networks were trained in parallel for $\sim$40k epochs, with 100k training samples, of which 100 were held out for testing. We used a batch size of 125, the cross-entropy loss and the Adam optimiser with learning rate$= 0.0001$, $\beta_1 = 0.9$ and $\beta_2 = 0.99$ for both networks. In each epoch, there was 1 discriminator update for every generator update. To ensure stability of training, we used spectral normalisation for the discriminator weights, and clipped the gradient norms for both networks to 0.01, unrolled the discriminator(Metz et al., 2017) with 10 updates i.e., in each epoch, we updated the discriminator 10 times, but reset it to the state after the first update following the generator update.

NPE and NLE We trained NPE and NLE as implemented in the sbi package (Tejero-Cantero et al., 2020) on the shallow water model. We set training hyperparameters as described in Lueckmann et al. (2021), except for the training batch size which we set to 100 for NPE and NLE. The number of hidden units in the density and ratio estimators which we set to 100 (default is 50). For NPE, we included an embedding net to embed the 20k-dimensional observations to the number of hidden units. This embedding net was identical to the one used for the GATSBI discriminators, and it was trained jointly with the corresponding density or ratio estimators. We trained with exactly the same 100k training samples used for GATSBI. MCMC sampling parameters for NLE were set as in Lueckmann et al. (2021).

To calculate **correlation coefficients** for the GATSBI and NPE posteriors, we sampled 1000 depth profiles from the trained networks for each of 1000 different observations from a test set. We then calculated the mean of the 1000 depth profile samples per observation, and computed the correlation coeffiecient of the mean with the corresponding groundtruth depth profile. Thus, we had 1000 different correlation coefficients; we report the mean and the standard deviation for these correlation coefficients.

Simulation-based calibration (Talts et al., 2020) offers a way to evaluate simulation-based inference in the absence of ground-truth posteriors. SBC checks whether the approximate posterior $q_\phi(\theta|x)$, when marginalised over multiple observations x, converges to the prior $\pi(\theta)$. A posterior that satisfies this condition is well-calibrated, although it is not a sufficient test of the quality of the learned posterior, since a posterior distribution that is equal to the prior would also be well-calibrated. However, when complemented with posterior predictive checks it provides a good test for intractable inference problems.

We performed SBC on the shallow water model. To obtain a test data set $\{\theta_i, x_i\}_{i=1}^N$, we sampled $N = 1000$ parameters θ_i from the prior and generated corresponding observations x_i from the simulator. For each x_i, we then obtained a set of $L = 1000$ posterior samples using the GATSBI generator and calculated the rank of the test parameter θ_i under the L GATSBI posterior samples as described in algorithm 1 in Talts et al. (2020), separately for each posterior dimension. For the ranking we used a Gaussian random variable with zero mean and variance 10. We then used

bins of $n = 20$ to compute and plot the histogram of the rank statistic. According to SBC, if the marginalised approximate posterior truly matched the prior, the rank statistic for each dimension should be uniformly distributed. Performing SBC can be computationally expensive because the inference has to be repeated for every test data point. In our scenario it was feasible only because GATSBI and NPE perform amortised inference and do not require retraining or MCMC sampling (as in the case of NLE) for every new x. We followed the same procedure to do SBC on NPE as for GATSBI.

D.3 NOISY CAMERA MODEL

Prior The parameters θ were 28×28-dimensional images sampled randomly from the 800k images in the EMNIST dataset.

Simulator The simulator takes a clean image as input, and corrupts it by first adding Poisson noise, followed by a convolution with a Gaussian point-spread function: $\mathbf{m} \sim \mathrm{Poisson}(\theta)$
$\mathbf{x}|\mathbf{th} = f * m; \quad f(t) = \exp(-\frac{t^2}{\sigma^2}))$ where $*$ denotes a convolution operation with a series of 1D Gaussian filters given by f. We set the width of the Gaussian function $\sigma = 3$.

GAN architecture The generator network was similar to the Pix2Pix generator (Isola et al., 2016): there were 9 blocks stacked sequentially; the first 4 blocks consisted of a 2D convolutional layer, followed by a leaky ReLU nonlinearity with slope 0.2 and a batchnorm layer; the next 4 blocks consisted of transpose a convolutional layer, followed by a leaky ReLU layer of slope 0.2 and a batchnorm layer; the final block had a transposed convolutional layer followed by a sigmoid nonlinearity. The input channels, output channels, kernel size, stride and padding for each of the convolutional or transposed convolutional layers in the 9 blocks were as follows: 1 - (1, 8, 2, 2, 1), 2 - (8, 16, 2, 2, 1), 3 - (16, 32, 2, 2, 1), 4 - (32, 64, 3, 1, 0), 5 - (128, 32, 3, 2, 1), 6 - (64, 16, 2, 2, 1), 7 - (32, 8, 3, 2, 1), 8 - (16, 4, 2, 2, 1), 9 - (4, 1, 1, 1, 0). There were skip-connections from block 1 to block 8, block 2 to block 7, block 3, to block 6 and from block 4 to block 5. 200-dimensional white noise was injected into the fifth block, after convolving it with a 1D convolutional filter and multiplying it with the output of the fourth block.

The discriminator network was again similar to the Pix2Pix discriminator: we concatenated the image from the generator or prior with the noisy image from the simulator, and passed this through 4 blocks consisting of a 2D convolutional layer, a leaky ReLU nonlinearity of slope 0.2 and a batch-norm layer, and finally through a 2D convolutional layer and a sigmoid nonlinearity. The input channels, output channels, kernel size, stride and padding for each of the convolutional were as follows: 1 - (2, 8, 2, 2, 1), 2 - (8, 16, 2, 2, 1), 3 - (16, 32, 2, 2, 1), 4 - (32, 64, 2, 2, 1), 5 - (64, 1, 3, 1, 0).

Training details The generator and discriminator were trained in tandem for 10k epochs, with 800k training samples, of which 100 were held out for testing. We used a batch size of 800, the cross-entropy loss and the Adam optimiser with learning rate$= 0.0002$, $\beta_1 = 0.5$ and $\beta_2 = 0.99$ for both networks. In each epoch, there was a single discriminator update for every second generator update. To ensure that training was stable, we used spectral normalisation for the discriminator weights, and clipped the gradient norms for both networks to 0.01.

NPE We trained NPE using the implementation in the sbi package (Tejero-Cantero et al., 2020). We set training hyperparameters as described in Lueckmann et al. (2021), except for the training batch size which we set to 1 (in order to ensure that we did not run out of memory while training), and the number of hidden units in the density estimators which we set to 100. 28×28 dimensional observations were passed directly to the flow, without an embedding net or computing any summary statistics, as was also the case for GATSBI. We trained with exactly the same 800k training samples used for GATSBI.

REFERENCES

Jonas Adler and Ozan Öktem. Deep bayesian inversion, 2018.

Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 214–223. PMLR, 06–11 Aug 2017.

Samaneh Azadi, Catherine Olsson, Trevor Darrell, Ian Goodfellow, and Augustus Odena. Discriminator Rejection Sampling. *arXiv:1810.06758 [cs, stat]*, 2019. arXiv: 1810.06758.

Jan O Backhaus. A semi-implicit scheme for the shallow water equations for application to shelf sea modelling. *Continental Shelf Research*, 2(4):243–254, 1983.

Lukas Biewald. Experiment tracking with weights and biases, 2020. Software available from wandb.com.

Tong Che, Ruixiang Zhang, Jascha Sohl-Dickstein, Hugo Larochelle, Liam Paull, Yuan Cao, and Yoshua Bengio. Your GAN is Secretly an Energy-based Model and You Should use Discriminator Driven Latent Sampling. *arXiv:2003.06060 [cs, stat]*, 2020.

Jeff Donahue, Trevor Darrell, and Philipp Krähenbühl. Adversarial feature learning. *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, pages 1–18, 2019.

Vincent Dumoulin, Ishmael Belghazi, Ben Poole, Olivier Mastropietro, Alex Lamb, Martin Arjovsky, and Aaron Courville. Adversarially learned inference. *arXiv preprint arXiv:1606.00704*, 2016.

William Fedus, Mihaela Rosca, Balaji Lakshminarayanan, Andrew M Dai, Shakir Mohamed, and Ian Goodfellow. Many paths to equilibrium: Gans do not need to decrease a divergence at every step. *arXiv preprint arXiv:1710.08446*, 2017.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

Ferenc Huszár. Variational inference using implicit distributions. *arXiv preprint arXiv:1702.08235*, 2017.

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. *arXiv preprint arXiv:1611.07004*, 2016.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations, ICLR*, 2015.

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 30*, pages 1289–1299. Curran Associates, Inc., 2017.

Jan-Matthis Lueckmann, Jan Boelts, David Greenberg, Pedro Goncalves, and Jakob Macke. Benchmarking simulation-based inference. In *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, pages 343–351, 2021.

Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, Ian Goodfellow, and Brendan Frey. Adversarial autoencoders. *arXiv preprint arXiv:1511.05644*, 2015.

Lars Mescheder, Sebastian Nowozin, and Andreas Geiger. Adversarial variational Bayes: Unifying variational autoencoders and generative adversarial networks. *34th International Conference on Machine Learning, ICML 2017*, 5:3694–3707, 2017.

Lars Mescheder, Andreas Geiger, and Sebastian Nowozin. Which training methods for gans do actually converge?, 2018.

Luke Metz, Ben Poole, David Pfau, and Jascha Sohl-Dickstein. Unrolled generative adversarial networks, 2017.

Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. *CoRR*, abs/1802.05957, 2018.

George Papamakarios, David Sterratt, and Iain Murray. Sequential neural likelihood: Fast likelihood-free inference with autoregressive flows. In *Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 89 of *Proceedings of Machine Learning Research*, pages 837–848. PMLR, 2019.

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 *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Guo-Jun Qi. Loss-sensitive generative adversarial networks on lipschitz densities, 2018.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

Sean Talts, Michael Betancourt, Daniel Simpson, and Aki Vehtari. Validating Bayesian Inference Algorithms with Simulation-Based Calibration. pages 1–26, 2020.

Alvaro Tejero-Cantero, Jan Boelts, Michael Deistler, Jan-Matthis Lueckmann, Conor Durkan, Pedro J. Gonçalves, David S. Greenberg, and Jakob H. Macke. sbi: A toolkit for simulation-based inference. *Journal of Open Source Software*, 5(52):2505, 2020.

Dustin Tran, Rajesh Ranganath, and David Blei. Hierarchical implicit models and likelihood-free variational inference. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.