Appendices

A Algorithms

A.1 Rejection Approximate Bayesian Computation (REJ-ABC)

Algorithm 1: Rejection ABC

while *in simulation budget* **do**
  Sample $\boldsymbol{\theta}'$ from $p(\boldsymbol{\theta})$
  Simulate data $\mathbf{x}'$ from $p(\mathbf{x}|\boldsymbol{\theta}')$
  if $d(\mathbf{x}', \mathbf{x}_o) \leq \epsilon$ **then**
    Accept $\boldsymbol{\theta}'$
  else
    Reject $\boldsymbol{\theta}'$
  end
end
return Accepted samples $\{\boldsymbol{\theta}'\}$ from $\hat{p}(\boldsymbol{\theta}|d(\mathbf{x}, \mathbf{x}_o) \leq \epsilon)$

Classical Approximate Bayesian Computation (ABC) is based on Monte Carlo rejection sampling (Tavaré et al., 1997; Pritchard et al., 1999): In rejection ABC, the evaluation of the likelihood is replaced by a comparison between observed data $\mathbf{x}_o$ and simulated data $\mathbf{x}$, based on a distance measure $d(\mathbf{x}, \mathbf{x}_o)$. Samples $\boldsymbol{\theta}$ from the approximate posterior are obtained by collecting simulation parameters that result in simulated data that is close to the observed data.

More formally, given observed data $\mathbf{x}_o$, a prior $p(\boldsymbol{\theta})$ over parameters of simulation-based model $p(\mathbf{x}|\boldsymbol{\theta})$, a distance measure $d(\mathbf{x}, \mathbf{x}_o)$ and an acceptance threshold ϵ, rejection ABC obtains parameter samples $\boldsymbol{\theta}$ from the approximate posterior as outlined in Algorithm 1.

In theory, rejection ABC obtains samples from the true posterior $p(\boldsymbol{\theta}|\mathbf{x}_o)$ in the limit $\epsilon \to 0$ and $N \to \infty$, where N is the simulation budget. In practice, its accuracy depends on the trade-off between simulation budget and the rejection criterion ϵ. Rejection ABC suffers from the curse of dimensionality, i.e., with linear increase in the dimensionality of $\mathbf{x}$, an exponential increase in simulation budget is required to maintain accurate results.

For the benchmark, we did not use a fixed ϵ-threshold, but quantile-based rejection. Depending on the simulation budget (1k, 10k, 100k), we used a quantile of (0.1, 0.01, or, 0.001), so that REJ-ABC returned 100 samples with smallest distance to $\mathbf{x}_o$ in each of these cases (see Appendix H for different hyperparameter choices). In order to compute metrics on 10k samples, we sampled from a KDE fitted on the accepted parameters (details about KDE resampling in Appendix H). REJ-ABC requires the choice of the distance measure $d(\mathbf{x}, \mathbf{x}_o)$: here we used the l_2-norm.

A.2 Sequential Monte Carlo Approximate Bayesian Computation (SMC-ABC)

Algorithm 2: Population Monte Carlo ABC (ABC-PMC) as in Beaumont et al. (2009)

Set schedule $\boldsymbol{\epsilon}$ (including initial ϵ_0), population indicator $t = 0$, and population size N
Initialize weights $W_0 = 1/N$ uniformly
Sample initial population $\{\boldsymbol{\theta}_0^{(i)}\}$ using rejection sampling with ϵ_0

while *in simulation budget* **do**
  Increase population indicator $t = t + 1$
  Set particle indicator $i = 0$
  while $i < N$ **do**
    Sample $\boldsymbol{\theta}'$ from previous population $\{\boldsymbol{\theta}_{t-1}^{(i)}\}$ with weights $\{W_{t-1}^{(i)}\}$;
    Perturb $\boldsymbol{\theta}'$: $\boldsymbol{\theta}'' \sim K_t(\boldsymbol{\theta}|\boldsymbol{\theta}')$
    Simulate data x'' from $p(\mathbf{x}|\boldsymbol{\theta}'')$
    if $d(\mathbf{x}'', \mathbf{x}_o) \leq \epsilon_t$ **then**
      Set $\boldsymbol{\theta}_t^{(i)} = \boldsymbol{\theta}''$ and $W_t^i = \frac{p(\boldsymbol{\theta}_t^{(i)})}{\sum_{j=1}^N W_{t-1}^j K_t(\boldsymbol{\theta}_t^{(i)}|\boldsymbol{\theta}_{t-1}^j)}$
      Increase particle indicator $i = i + 1$
    else
      reject $\boldsymbol{\theta}''$
    end
  end
  Normalize weights so that $\sum_i W_t^{(i)} = 1$
end
return Weighted samples $\{\boldsymbol{\theta}_t^{(i)}\}$ from $\hat{p}(\boldsymbol{\theta}|d(\mathbf{x}, \mathbf{x}_o) \leq \epsilon)$

Sequential Monte Carlo Approximate Bayesian Computation (SMC-ABC) algorithms (Beaumont et al., 2002; Marjoram and Tavaré, 2006; Sisson et al., 2007; Toni et al., 2009) are an extension of the classical rejection ABC approach, inspired by importance sampling and sequential Monte Carlo sampling. Central to SMC-ABC is the idea to approach the final set of samples from the approximate posterior by constructing a series of intermediate sets of samples slowly approaching the final set through perturbations.

Several variants have been developed (e.g., Sisson et al., 2007; Beaumont et al., 2009; Toni et al., 2009; Simola et al., 2020). Here, we used the scheme ABC-PMC scheme of Beaumont et al. (2009) and refer to it as SMC-ABC in the manuscript. More formally, the description of the ABC-PMC algorithm is as follows: Given observed data $\mathbf{x}_o$, a prior $p(\boldsymbol{\theta})$ over parameters of a simulation-based model $p(\mathbf{x}|\boldsymbol{\theta})$, a distance measure $d(\mathbf{x}, \mathbf{x}_o)$, a schedule of acceptance thresholds ϵ_i, and a kernel $K(\boldsymbol{\theta}|\boldsymbol{\theta}')$ to perturb intermediate samples, weighted samples of the approximate posterior are obtained as described in Algorithm 2.

SMC-ABC can improve the sampling efficiency compared to REJ-ABC and avoids severe inefficiencies due to a mismatch between initial sampling and the target distribution. However, it comes with more hyperparameters that can require careful tuning to the problem at hand, e.g., the choice of distance measure, kernel, and ϵ-schedule. Like, REJ-ABC, SMC-ABC suffers from the curse of dimensionality.

For the benchmark, we considered the popular toolbox `pyABC` (Klinger et al., 2018). Additionally, to fully understand the details of the SMC-ABC approach, we also implemented our own version. In the main paper we report results obtained with our implementation because it yielded slightly better results. A careful comparison of the two approaches, and the optimization of hyperparameters like ϵ-schedule, population size and perturbation kernel variance across different tasks are shown in Appendix H. After optimization, the crucial parameters of SMC-ABC were set to: l_2-norm as distance metric, quantile-based epsilon decay with 0.2 quantile, population size 100 for simulation budgets 1k and 10k, population size 1000 for simulation budget 100k, Gaussian perturbation kernel with empirical covariance from previous population scaled by 0.5. We obtained 10k samples required for calculation of metrics as follows: If a population is not complete within the simulation budget we completed it with accepted particles from the last population and recalculated all weights. We then fitted a KDE on all those particles and sampled 10k samples from the KDE.

A.3 Neural Likelihood Estimation (NLE)

Algorithm 3: Single round Neural Likelihood as in Papamakarios et al. (2019b)

Set $\mathcal{D} = \{\}$
for $n = 1 : N$ **do**
  Sample $\boldsymbol{\theta}_n \sim p(\boldsymbol{\theta})$
  Simulate $\mathbf{x}_n \sim p(\mathbf{x}|\boldsymbol{\theta}_n)$
  Add $(\boldsymbol{\theta}_n, \mathbf{x}_n)$ to $\mathcal{D}$
end
Train $q_{\boldsymbol{\psi}}(\mathbf{x}|\boldsymbol{\theta})$ on $\mathcal{D}$
return Samples from $\hat{p}(\boldsymbol{\theta}|\mathbf{x}_o) \propto q_{\boldsymbol{\psi}}(\mathbf{x}_o|\boldsymbol{\theta})p(\boldsymbol{\theta})$ via MCMC; $q_{\boldsymbol{\psi}}(\mathbf{x}|\boldsymbol{\theta})$

Likelihood estimation approaches to SBI use density estimation to approximate the likelihood $p(\mathbf{x}_o|\boldsymbol{\theta})$. After learning a surrogate $q_{\boldsymbol{\psi}}$ ($\boldsymbol{\psi}$ denoting the parameters of the estimator) for the likelihood function, one can for example use Markov Chain Monte Carlo (MCMC) based sampling algorithms to obtain samples from the approximate posterior $\hat{p}(\boldsymbol{\theta}|\mathbf{x}_o)$. This idea dates back to using Gaussian approximations of the likelihood (Wood, 2010; Drovandi et al., 2018), and more recently, was extended to density estimation with neural networks (Papamakarios et al., 2019b; Lueckmann et al., 2019).

We refer to the single-round version of the (sequential) neural likelihood approach by Papamakarios et al. (2019b) as NLE, and outline it in Algorithm 3: Given a set of samples $\{\boldsymbol{\theta}_n, \mathbf{x}_n\}_{1:N}$ obtained by sampling $\boldsymbol{\theta}_n \sim p(\boldsymbol{\theta})$ from the prior and simulating $\mathbf{x}_n \sim p(\mathbf{x}|\boldsymbol{\theta}_n)$, we train a conditional neural density estimator $q_{\boldsymbol{\psi}}(\mathbf{x}|\boldsymbol{\theta})$ modelling the conditional of data given parameters on the set $\{\boldsymbol{\theta}_n, \mathbf{x}_n\}_{1:N}$. Training proceeds by maximizing the log likelihood $\sum_n \log q_{\boldsymbol{\psi}}(\mathbf{x}|\boldsymbol{\theta})$. Given enough simulations, a sufficiently flexible conditional neural density estimator approximates the likelihood in the support of the prior $p(\boldsymbol{\theta})$ (Papamakarios et al., 2019b). Once $q_{\boldsymbol{\psi}}$ is trained, samples from the approximate posterior $\hat{p}(\boldsymbol{\theta}|\mathbf{x}_o)$ are obtained using MCMC sampling based on the approximate likelihood $\hat{p}(\mathbf{x}_o|\boldsymbol{\theta})$ and the prior $p(\boldsymbol{\theta})$.

For MCMC sampling, Papamakarios et al. (2019b) suggest to use Slice Sampling (Neal, 2003) with a single chain. However, we observed that the accuracy of the obtained posterior samples can be substantially improved by changing the Slice Sampling scheme as follows: 1) Instead of a single chain, we used 100 parallel MCMC chains; 2) for initialization of the chains, we sampled 10k candidate parameters from the prior, evaluated them under the unnormalized approximate posterior, and used these values as weights to resample initial locations; 3) we transformed parameters to be unbounded as suggested e.g. in Bingham et al. (2019); Carpenter et al. (2017); Hogg and Foreman-Mackey (2018). In addition, we reimplemented the slice sampler to allow vectorized evaluations of the likelihood, which yielded significant computational speed-ups.

For the benchmark, we used as density estimator a Masked Autoregressive Flow (MAF, Papamakarios et al., 2017) with five flow transforms, each with two blocks and 50 hidden units, tanh non-linearity and batch normalization after each layer. For the MCMC step, we used the scheme as outlined above with 250 warm-up steps and ten-fold thinning, to obtain 10k samples from the approximate posterior (1k samples from each chain). In Appendix H we show results for all tasks obtained with a Neural Spline Flow (NSF, Durkan et al., 2019) for density estimation, using five flow transforms, two residual blocks of 50 hidden units each, ReLU non-linearity, and 10 bins.

A.4 Sequential Neural Likelihood Estimation (SNLE)

Algorithm 4: Sequential Neural Likelihood as in Papamakarios et al. (2019b)

Set $\hat{p}_0(\boldsymbol{\theta}|\mathbf{x}_o) = p(\boldsymbol{\theta})$ and $\mathcal{D} = \{\}$
for $r = 1 : R$ **do**
  for $n = 1 : N$ **do**
    Sample $\boldsymbol{\theta}_n \sim \hat{p}_{r-1}(\boldsymbol{\theta}|\mathbf{x}_o)$ with MCMC
    Simulate $\mathbf{x}_n \sim p(\mathbf{x}|\boldsymbol{\theta}_n)$
    Add $(\boldsymbol{\theta}_n, \mathbf{x}_n)$ to $\mathcal{D}$
  end
  (Re-)train $q_{\boldsymbol{\psi}}(\mathbf{x}|\boldsymbol{\theta})$ on $\mathcal{D}$
  Set $\hat{p}_r(\boldsymbol{\theta}|\mathbf{x}_o) \propto q_{\boldsymbol{\psi}}(\mathbf{x}_o|\boldsymbol{\theta})p(\boldsymbol{\theta})$
end
return Samples from $\hat{p}(\boldsymbol{\theta}|\mathbf{x}_o) \propto q_{\boldsymbol{\psi}}(\mathbf{x}_o|\boldsymbol{\theta})p(\boldsymbol{\theta})$ via MCMC; $q_{\boldsymbol{\psi}}(\mathbf{x}|\boldsymbol{\theta})$

Sequential Neural Likelihood estimation (SNLE or SNL, Papamakarios et al., 2019b) extends the neural likelihood estimation approach described in the previous section to be sequential.

The idea behind sequential SBI algorithms is based on the following intuition: If for a particular inference problem, there is only a single $\mathbf{x}_o$ one is interested in, then simulating data using parameters from the entire prior space might be inefficient, leading to a training set $\mathcal{D}$ that contains training data $(\boldsymbol{\theta}, \mathbf{x})$ which carries little information about the posterior $p(\boldsymbol{\theta}|\mathbf{x}_o)$. Instead, to increase sample efficiency, one may draw training data points from a proposal distribution $\tilde{p}(\boldsymbol{\theta})$, ideally obtaining $\boldsymbol{\theta}$ for which $\mathbf{x}$ is close to $\mathbf{x}_o$. One candidate that has been commonly used in the literature for such a proposal is the approximate posterior distribution itself.

SNLE is a multi-round version of NLE, where in each round new training samples are drawn from a proposal $\tilde{p}(\boldsymbol{\theta})$. The proposal is chosen to be the posterior estimate at $\mathbf{x}_o$ from the previous round $\hat{p}(\boldsymbol{\theta}|\mathbf{x}_o)$ and its samples are obtained using MCMC. The proposal controls where $q_{\boldsymbol{\psi}}(\mathbf{x}|\boldsymbol{\theta})$ is learned most accurately. Thus, by iterating over multiple rounds, a good approximation to the posterior can be learned more efficiently than by sampling all training data from the prior. SNLE is summarized in Algorithm 4.

For the benchmark, we used as density estimator a Masked Autoregressive Flow (Papamakarios et al., 2017), and MCMC to obtain posterior samples after every round, both with the same settings as described for NLE. The simulation budget was equally split across 10 rounds. In Appendix H, we show results for all tasks obtained with a Neural Spline Flow (NSF, Durkan et al., 2019) for density estimation, using five flow transforms, two residual blocks of 50 hidden units each, ReLU non-linearity, and 10 bins.

A.5 Neural Posterior Estimation (NPE)

Algorithm 5: Single round Neural Posterior Estimation as in Papamakarios and Murray (2016)

for $j = 1 : N$ **do**
  Sample $\boldsymbol{\theta}_j \sim p(\boldsymbol{\theta})$
  Simulate $\mathbf{x}_j \sim p(\mathbf{x}|\boldsymbol{\theta}_j)$
end
$\boldsymbol{\phi} \leftarrow \arg\min \sum_j^N -\log q_{F(\mathbf{x}_j,\boldsymbol{\phi})}(\boldsymbol{\theta}_j)$
Set $\hat{p}(\boldsymbol{\theta}|\mathbf{x}_o) = q_{F(\mathbf{x}_o,\boldsymbol{\phi})}(\boldsymbol{\theta})$
return Samples from $\hat{p}(\boldsymbol{\theta}|\mathbf{x}_o)$; $q_{F(\mathbf{x},\boldsymbol{\phi})}(\boldsymbol{\theta})$

NPE uses conditional density estimation to directly estimate the posterior. This idea dates back to regression adjustment approaches (Blum and François, 2010) and was extended to density estimators using neural networks (Papamakarios and Murray, 2016) more recently.

As outlined in Algorithm 5, the approach is as follows: Given a prior over parameters $p(\boldsymbol{\theta})$ and a simulator, a set of training data points $(\boldsymbol{\theta}, \mathbf{x})$ is generated. This training data is used to learn the parameters $\boldsymbol{\psi}$ of a conditional density estimator $q_{\boldsymbol{\psi}}(\boldsymbol{\theta}|x)$ using a neural network $F(\mathbf{x}, \boldsymbol{\phi})$, i.e., $\boldsymbol{\psi} = F(\mathbf{x}, \boldsymbol{\phi})$. The loss function is given by the negative log probability $-\log q_{\boldsymbol{\psi}}(\boldsymbol{\theta}|x)$. If the density estimator q is flexible enough and training data is infinite, this loss function leads to perfect recovery of the ground-truth posterior (Papamakarios and Murray, 2016).

For the benchmark, we used the approach by Papamakarios and Murray (2016) with a Neural Spline Flow (NSF, Durkan et al., 2019) as density estimator, using five flow transforms, two residual blocks of 50 hidden units each, ReLU non-linearity, and 10 bins. We sampled 10k samples from the approximate posterior $q_{F(\mathbf{x}_o,\boldsymbol{\phi})}(\boldsymbol{\theta})$. In Appendix H, we compare NSFs to Masked Autoregressive Flows (MAFs, Papamakarios et al., 2017), as used in Greenberg et al. (2019); Durkan et al. (2020), with five flow transforms, each with two blocks and 50 hidden units, tanh non-linearity and batch normalization after each layer.

A.6 Sequential Neural Posterior Estimation (SNPE)

Algorithm 6: Sequential Neural Posterior Estimation with atomic proposals (Greenberg et al., 2019)

Set $\tilde{p}_1(\boldsymbol{\theta}) = p(\boldsymbol{\theta})$
$c \leftarrow 0$
for $r = 1 : R$ **do**
  for $j = 1 : N$ **do**
    $c \leftarrow c + 1$
    Sample $\boldsymbol{\theta}_c \sim \tilde{p}_r(\boldsymbol{\theta})$
    Simulate $\mathbf{x}_c \sim p(\mathbf{x}|\boldsymbol{\theta}_c)$
  end
  $V_r(\Theta) := \begin{cases} \binom{c}{M}^{-1} & \text{if } \Theta = \{\boldsymbol{\theta}_{b_1}, \boldsymbol{\theta}_{b_1}, \ldots, \boldsymbol{\theta}_{b_M}\} \text{ and } 1 \le b_1 < b_2 < \ldots < b_M \le c \\ 0 & \text{otherwise} \end{cases}$
  $\boldsymbol{\phi} \leftarrow \arg\min_{\boldsymbol{\phi}} \mathbb{E}_{\Theta \sim V_r(\Theta)} \left[\sum_{\boldsymbol{\theta}_j \in \Theta} -\log \tilde{q}_{\mathbf{x}_j, \boldsymbol{\phi}}(\boldsymbol{\theta}_j) \right]$
  Set $\tilde{p}_{r+1}(\boldsymbol{\theta}) := q_{F(\mathbf{x}_o, \boldsymbol{\phi})}(\boldsymbol{\theta})$
end
return Samples from $\hat{p}_R(\boldsymbol{\theta}|\mathbf{x}_o)$; $q_{F(x, \boldsymbol{\phi})}(\boldsymbol{\theta})$

Sequential Neural Posterior Estimation SNPE is the sequential analog of NPE, and meant to increase sample efficiency (see also subsection A.4). When the posterior is targeted directly, using a proposal distribution $\tilde{p}(\boldsymbol{\theta})$ different from the prior requires a correction step—without it, the posterior under the proposal distribution would be inferred (Papamakarios and Murray, 2016). This so-called proposal posterior is denoted by $\tilde{p}(\boldsymbol{\theta}|\mathbf{x})$:

$$\tilde{p}(\boldsymbol{\theta}|\mathbf{x}) = p(\boldsymbol{\theta}|\mathbf{x}) \frac{\tilde{p}(\boldsymbol{\theta}) p(\mathbf{x})}{p(\boldsymbol{\theta}) \tilde{p}(\mathbf{x})},$$

where $\tilde{p}(\mathbf{x}) = \int_{\boldsymbol{\theta}} \tilde{p}(\boldsymbol{\theta}) p(\mathbf{x}|\boldsymbol{\theta})$. Note that for $\tilde{p}(\boldsymbol{\theta}) = p(\boldsymbol{\theta})$, it directly follows that $\tilde{p}(\boldsymbol{\theta}|\mathbf{x}) = p(\boldsymbol{\theta}|\mathbf{x})$.

There have been three different approaches to this correction step so far, leading to three versions of SNPE (Papamakarios and Murray, 2016; Lueckmann et al., 2017; Greenberg et al., 2019). All three algorithms have in common that they train a neural network $F(\mathbf{x}, \boldsymbol{\phi})$ to learn the parameters of a family of densities $q_{\boldsymbol{\psi}}$ to estimate the posterior. They differ in what is targeted by $q_{\boldsymbol{\psi}}$ and which loss is used for F.

SNPE-A (Papamakarios and Murray, 2016) trains F to target the proposal posterior $\tilde{p}(\boldsymbol{\theta}|\mathbf{x})$ by minimizing the log likelihood loss $-\sum_n \log q_{\boldsymbol{\psi}}(\boldsymbol{\theta}_n|\mathbf{x}_n)$, and then post-hoc solves for $p(\boldsymbol{\theta}|\mathbf{x})$. The analytical post-hoc step places restrictions on $q_{\boldsymbol{\psi}}$, the proposal, and prior. Papamakarios and Murray (2016) used Gaussian mixture density networks, single Gaussians proposals, and Gaussian or uniform priors. SNPE-B (Lueckmann et al., 2017) trains F with the importance weighted loss $-\sum_n \frac{p(\boldsymbol{\theta}_n)}{\tilde{p}(\boldsymbol{\theta}_n)} \log q_{\boldsymbol{\psi}}(\boldsymbol{\theta}_n|\mathbf{x}_n)$ to directly recover $p(\boldsymbol{\theta}|\mathbf{x})$ without the need for post-hoc correction, removing restrictions with respect to $q_{\boldsymbol{\psi}}$, the proposal, and prior. However, the importance weights can have high variance during training, leading to inaccurate inference for some tasks (Greenberg et al., 2019). SNPE-C (APT) (Greenberg et al., 2019) alleviates this issue by reparameterizing the problem such that it can infer the posterior by maximizing an estimated proposal posterior. It trains F to approximate $p(\boldsymbol{\theta}|\mathbf{x})$ with $q_{F(\mathbf{x}, \boldsymbol{\phi})}(\boldsymbol{\theta})$, using a loss defined on the approximate proposal posterior $\tilde{q}_{\mathbf{x}, \boldsymbol{\phi}}(\boldsymbol{\theta})$. Greenberg et al. (2019) introduce 'atomic' proposals to allow for arbitrary choices of the density estimator, e.g., flows (Papamakarios et al., 2019a): The loss on $\tilde{q}_{\mathbf{x}, \boldsymbol{\phi}}(\boldsymbol{\theta})$ is calculated as the expectation over proposal sets Θ sampled from a so-called 'hyperproposal' $V(\Theta)$ as outlined in Algorithm 6 (see Greenberg et al., 2019, for full details).

For the benchmark, we used the approach by Greenberg et al. (2019) with 'atomic' proposals and referred to it as SNPE. As density estimator, we used a Neural Spline Flow (Durkan et al., 2019) with the same settings as for NPE. For the 'atomic' proposals, we used $M = 10$ atoms (larger M was too demanding in terms of memory). The simulation budget was equally split across 10 rounds and for the final round, we obtained 10k samples from the approximate posterior $\hat{p}_R(\boldsymbol{\theta}|\mathbf{x}_o)$. In Appendix H, we compare NSFs to Masked Autoregressive Flows (MAFs, Papamakarios et al., 2017), as used in Greenberg et al. (2019); Durkan et al. (2020), with five flow transforms, each with two blocks and 50 hidden units, tanh non-linearity and batch normalization after each layer.

A.7 Neural Ratio Estimation (NRE)

Algorithm 7: Single round Neural Ratio Estimation as in Hermans et al. (2020)

Set optimization criterion l (e.g., BCE)
for $j = 1 : N$ **do**
  Sample $\boldsymbol{\theta}_j \sim p(\boldsymbol{\theta})$
  Sample $\boldsymbol{\theta}'_j \sim p(\boldsymbol{\theta})$
  Simulate $\mathbf{x}_j \sim p(\mathbf{x}|\boldsymbol{\theta}_j)$
end
$\boldsymbol{\phi} \leftarrow \arg\min l(d_{\boldsymbol{\phi}}(\mathbf{x}_n, \boldsymbol{\theta}_n), 1) + l(d_{\boldsymbol{\phi}}(\mathbf{x}_n, \boldsymbol{\theta}'_n), 0)$
Parameterize $d_{\boldsymbol{\phi}}(\mathbf{x}, \boldsymbol{\theta})$
return Samples from $\hat{p}(\boldsymbol{\theta}|\mathbf{x}_o)$ via MCMC; $d_{\boldsymbol{\phi}}(\mathbf{x}, \boldsymbol{\theta})$

Neural ratio estimation (NRE) uses neural-network based classifiers to approximate the posterior $p(\boldsymbol{\theta}|\mathbf{x}_o)$. While neural-network based approaches described in the previous sections use *density estimation* to either estimate the likelihood ((S)NLE) or the posterior ((S)NPE), NRE algorithms ((S)NRE) use *classification* to estimate a ratio of likelihoods. The ratio can then be used for posterior evaluation or MCMC-based sampling.

Likelihood ratio estimation can be used for SBI because it allows to perform MCMC without evaluating the intractable likelihood. In MCMC, the transition probability from a current parameter $\boldsymbol{\theta}_t$ to a proposed parameter $\boldsymbol{\theta}'$ depends on the posterior ratio and in turn on the likelihood ratio between the two parameters:

$$\frac{p(\boldsymbol{\theta}'|\mathbf{x})}{p(\boldsymbol{\theta}_t|\mathbf{x})} = \frac{p(\boldsymbol{\theta}')p(\mathbf{x}|\boldsymbol{\theta}')/p(\mathbf{x})}{p(\boldsymbol{\theta}_t)p(\mathbf{x}|\boldsymbol{\theta}_t)/p(\mathbf{x})} = \frac{p(\boldsymbol{\theta}')p(\mathbf{x}|\boldsymbol{\theta}')}{p(\boldsymbol{\theta}_t)p(\mathbf{x}|\boldsymbol{\theta}_t)}.$$

Therefore, given a ratio estimator $r(\mathbf{x}|\boldsymbol{\theta}', \boldsymbol{\theta}_t) = \frac{p(\mathbf{x}|\boldsymbol{\theta}')}{p(\mathbf{x}|\boldsymbol{\theta}_t)}$ learned from simulations, one can perform MCMC to obtain samples from the posterior, even if evaluating $p(\mathbf{x}|\boldsymbol{\theta})$ is intractable.

Hermans et al. (2020) proposed the following approach for MCMC with classifiers to approximate density ratios: A classifier is trained to distinguish samples from an arbitrary $(\boldsymbol{\theta}, \mathbf{x}) \sim p(\mathbf{x}|\boldsymbol{\theta})p(\boldsymbol{\theta})$ and samples from the marginal model $(\boldsymbol{\theta}, \mathbf{x}) \sim p(\boldsymbol{\theta})p(\mathbf{x})$. This results in a likelihood-to-evidence estimator that needs to be trained only once to be evaluated for any $\boldsymbol{\theta}$. The training of the classifier $d_{\boldsymbol{\phi}}(\mathbf{x}, \boldsymbol{\theta})$ proceeds by minimizing the binary cross-entropy loss (BCE), as outlined in Algorithm 7. Once the classifier $d_{\boldsymbol{\phi}}(\mathbf{x}, \boldsymbol{\theta})$ is parameterized, it can be used to perform MCMC to obtain samples from the posterior. The authors name their approach *Amortized Approximate Likelihood Ratio MCMC* (AALR-MCMC): It is amortized because once the likelihood ratio estimator is trained, it is possible to run MCMC for any $\mathbf{x} \sim p(\mathbf{x})$.

Earlier ratio estimation algorithms for SBI (e.g., Izbicki et al., 2014; Pham et al., 2014; Cranmer et al., 2015; Dutta et al., 2016) and their connections to recent methods are discussed in Thomas et al. (2020), as well as in Durkan et al. (2020). AALR-MCMC is closely related to LFIRE (Dutta et al., 2016) but trains an amortized classifier rather than a separate one per posterior evaluation. Durkan et al. (2020) showed that the loss of AALR-MCMC is closely related to the atomic SNPE-C/APT approach of Greenberg et al. (2019) (SNPE) and that both can be combined in a unified framework. Durkan et al. (2020) changed the formulation of the loss function for training the classifier from binary to multi-class.

For the benchmark, we used neural ratio estimation (NRE) as formulated by Durkan et al. (2020) and implemented in the `sbi` toolbox (Tejero-Cantero et al., 2020). As a classifier, we used a residual network architecture (ResNet) with two hidden layers of 50 units and ReLU non-linearity, trained with Adam (Kingma and Ba, 2015). Following the notation of Durkan et al. (2020), we used $K = 10$ as the size of the contrasting set. For the MCMC step, we followed the same procedure as described for NLE, i.e., using Slice Sampling with 100 chains, to obtain 10k samples from each approximate posterior. In Appendix H, we show results for all tasks obtained with a multi-layer perceptron (MLP) architecture with two hidden layers of 50 ReLu units, and batch normalization.

A.8 Sequential Neural Ratio Estimation (SNRE)

Algorithm 8: Sequential Neural Ratio Estimation as in Hermans et al. (2020)

Set optimization criterion l (e.g., BCE)
Set $\tilde{p}(\boldsymbol{\theta}) = p(\boldsymbol{\theta})$
for $r = 1 : R$ **do**
  for $j = 1 : N$ **do**
    Sample $\boldsymbol{\theta}_j \sim \tilde{p}(\boldsymbol{\theta})$ (via $d_{\boldsymbol{\phi}}$ and MCMC)
    Sample $\boldsymbol{\theta}'_j \sim \tilde{p}(\boldsymbol{\theta})$ (via $d_{\boldsymbol{\phi}}$ and MCMC)
    Simulate $\mathbf{x}_j \sim p(\mathbf{x}|\boldsymbol{\theta}_j)$
  end
  $\boldsymbol{\phi} \leftarrow \arg\min l(I'n, \boldsymbol{\theta}_n), 1) + l(d_{\boldsymbol{\phi}}(\mathbf{x}_n, \boldsymbol{\theta}'_n), 0)$;
  Parameterize $d_{\boldsymbol{\phi}}(\mathbf{x}, \boldsymbol{\theta})$
end
return Samples from $\hat{p}(\boldsymbol{\theta}|\mathbf{x}_o)$ via MCMC; $d_{\boldsymbol{\phi}}(\mathbf{x}, \boldsymbol{\theta})$

Sequential Neural Ratio Estimation (SNRE) is the sequential version of NRE, and meant to increase sample efficiency, at the cost of needing to train new classifiers for different $\mathbf{x}_o$.

A sequential version of neural ratio estimation was proposed by Hermans et al. (2020). As with other sequential algorithms, the idea is to replace the prior by a proposal distribution $\tilde{p}(\boldsymbol{\theta})$ that is focused on $\mathbf{x}_o$ in the sense that the sampled parameters $\boldsymbol{\theta}$ result in simulated data $\mathbf{x}$ that are informative about $\mathbf{x}_o$. The proposal for the next round is the posterior estimate from the previous round. The ratio estimator then becomes $\tilde{r}(\mathbf{x}, \boldsymbol{\theta})$ and is refined over rounds by training the underlying classifier with positive examples $(\mathbf{x}, \boldsymbol{\theta}) \sim p(\mathbf{x}|\boldsymbol{\theta})\tilde{p}(\boldsymbol{\theta})$ and negative examples $(\mathbf{x}, \boldsymbol{\theta}) \sim p(\mathbf{x})\tilde{p}(\boldsymbol{\theta})$. Exact posterior evaluation is not possible anymore, but samples can be obtained as before via MCMC. These steps are outlined in Algorithm 8.

For the benchmark, we used SNRE as formulated by Durkan et al. (2020) and implemented in the `sbi` toolbox (Tejero-Cantero et al., 2020). The classifier had the same architecture as described for NRE. For the MCMC step, we followed the same procedure as described for NLE. The simulation budget was equally split across 10 rounds. In Appendix H, we show results for all tasks obtained with a multi-layer perceptron (MLP) architecture with two hidden layers of 50 ReLu units, and batch normalization.

A.9 Random Forest Approximate Bayesian Computation (RF-ABC)

Algorithm 9: Random Forest ABC (RF-ABC) as in Raynal et al. (2019)

Set $\mathcal{D} = \{\}$ Set simulation budget N
Set number of trees B
Set minimum node size N_{min}
for $n = 1 : N$ **do**
  Sample $\boldsymbol{\theta}_n \sim p(\boldsymbol{\theta})$
  Simulate $\mathbf{x}_n \sim p(\mathbf{x}|\boldsymbol{\theta}_n)$
  Add $(\boldsymbol{\theta}_n, \mathbf{x}_n)$ to $\mathcal{D}$
end
Run random forest regression of $\mathbf{x}$ on $\boldsymbol{\theta}$ using $\mathcal{D}$, B and N_{min}
return N samples $\{\boldsymbol{\theta}^{(i)}\}$ and associated weights $\{w^{(i)}\}$ for drawing approximate posterior samples

Random forest Approximate Bayesian Computation (RF-ABC, Pudlo et al., 2016; Raynal et al., 2019) is a more recently developed ABC algorithm based on a regression approach. Similar to previous regression approaches to ABC (Beaumont et al., 2002; Blum and François, 2010), RF-ABC aims at improving classical ABC inference (REJ-ABC, SMC-ABC) in the setting of high-dimensional data.

The idea of the RF-ABC algorithm is to use random forests (RF, Breiman, 2001) to run a non-parametric regression of a set of potential summary statistics of the data on the corresponding parameters. That is, the RF regression is trained on data simulated from the model, such that the covariates are the summary statistics and the response variable is a parameter. For a detailed description of the algorithm, we refer to Raynal et al. (2019).

The only hyperparameters for the RF-ABC algorithm are the number of trees and the minimum node size for the RF regression. Following Raynal et al. (2019), we chose the default of 500 trees and a minimum of 5 nodes. The output of the algorithm is a RF weight for each of the simulated parameters. This set of weights can be used to calculate posterior quantiles or to obtain an approximate posterior density as described in Raynal et al. (2019). We obtained 10k posterior samples for the benchmark by using the random forest weights to sample from the simulated parameters. We used the implementation in the `abcranger` toolbox Collin et al. (2020).

One important property of RF-ABC is that it can only be applied in the unidimensional setting, i.e., for 1-D dimensional parameter spaces, or for multidimensional parameters spaces with the assumption that the posterior factorizes over parameters (thus ignoring potential posterior correlations). This assumptions holds only for a few tasks in our benchmark (Gaussian Linear, Gaussian Linear Uniform, Gaussian Mixture). Due to this inherent limitation, we report RF-ABC in the supplement (see Suppl. Fig. 2).

A.10 Synthetic Likelihood (SL)

Algorithm 10: Synthetic Likelihood algorithm as in Wood (2010)

Set number of simulations per step M
Set number of MCMC steps T
for $t = 1 : T$ **do**
  Get new candidate $\boldsymbol{\theta}_t$ from MCMC scheme
  Set $\mathcal{D}_t = \{\}$
  for $m = 1 : M$ **do**
    Simulate $\mathbf{x}_m \sim p(\mathbf{x}|\boldsymbol{\theta}_t)$
    Add $(\boldsymbol{\theta}_t, \mathbf{x}_m)$ to $\mathcal{D}_t$
  end
  Use $\mathcal{D}_t$ to estimate mean and covariance of a Gaussian approximation of the likelihood $\hat{L}(\mathbf{x}_o|\theta_t)$
  Perform the next MCMC step using $\hat{L}(\mathbf{x}_o|\theta_t)$
end
return N samples $\{\boldsymbol{\theta}^{(i)}\}$ from MCMC chain

The Synthetic Likelihood (SL) approach circumvents the evaluation of the intractable likelihood by estimating a *synthetic* one from simulated data or summary statistics. This approach was introduced by Wood (2010). Its main motivation is that the classical ABC approach of comparing simulated and observed data with a distance metric can be problematic if parts of the differences are entirely noise-driven. Wood (2010) instead approximated the distribution of the summary statistics (the likelihood) of a nonlinear ecological dynamic system as a Gaussian distribution, thereby capturing the underlying noise as well. The approximation of the likelihood can then be used to obtain posterior sampling via Markov Chain Monte Carlo (MCMC) (Wood, 2010).

The SL approach can be seen as the predecessor of the (S)NLE approaches: They replaced the Gaussian approximation of the likelihood with a much more flexible one that uses neural networks and normalizing flows (see A.3). Moreover, there are modern approaches from the classical ABC field that further developed SL using a Gaussian approximation (e.g., Drovandi et al., 2018; Priddle et al., 2019).

For the benchmark, we implemented our own version of the algorithm proposed by Wood (2010). We used Slice Sampling MCMC (Neal, 2003) and estimated the Gaussian likelihood from 100 samples at each sampling step. To ensure a positive definite covariance matrix, we added a small value ϵ to the diagonal of the estimated covariance matrix for some of the tasks. In particular, we used $\epsilon = 0.01$ for SIR and Bernoulli GLM Raw tasks, and we tried without success $\epsilon = [0, 0.01, 0.1, 1.0]$ for Lotka-Volterra and SLCP with distractors. For all remaining tasks, we set $\epsilon = 0$. For Slice Sampling, we used a single chain initialized with sequential importance sampling (SIR) as described for NLE, 1k warm-up steps and no thinning, in order to keep the number of required simulations tractable. This resulted in an overall simulation budget on the order of 10^8 to 10^9 simulations per run in order to generate 10k posterior samples, as new simulations are required for every MCMC step.

The high simulation budget makes it problematic to directly compare SL and other other algorithms in the benchmark. Therefore, we report SL in the supplement (see Suppl. Fig. 3).

B Benchmark

B.1 Reference posteriors

We generated 10k reference posterior samples for each observation. For the Gaussian Linear task, reference samples were obtained by using the analytic solution for the true posterior. Similarly, for Gaussian Linear Uniform and Gaussian Mixture, the analytic solution was used, combined with an additional rejection step, in order to account for the bounded support of the posterior due to the use of a uniform prior. For the Two Moons task, we devised a custom scheme based on the model equations, which samples both modes and rejects samples outside the prior bounds.

For SLCP, SIR, and Lotka-Volterra, we devised a likelihood-based procedure to ensure obtaining a valid set of reference posterior samples: First, we either used Sampling/Importance Resampling (Rubin, 1988) (for SLCP, SIR) or Slice Sampling MCMC (Neal, 2003) (for Lotka-Volterra) to obtain a set of 10k proposal samples from the unnormalized posterior $f(\boldsymbol{\theta}) = \tilde{p}(\boldsymbol{\theta}|\mathbf{x}_o) = p(\mathbf{x}_o|\boldsymbol{\theta})p(\boldsymbol{\theta})$. We used these proposal samples to train a density estimator, for which we used a neural spline flow (NSF) (Durkan et al., 2019). Next, we created a mixture composed of the NSF and the prior with weights 0.9 and 0.1, respectively, as a proposal distribution $g(\boldsymbol{\theta})$ for rejection sampling (Martino et al., 2018). Rejection sampling relies on finding a constant M such that $f(\boldsymbol{\theta}) \leq Mg(\boldsymbol{\theta})$ for all values of $\boldsymbol{\theta}$: To find this constant, we initialized $M = 1$, sampled $\boldsymbol{\theta} \sim g(\boldsymbol{\theta})$, and updated $M = 1.2f(\boldsymbol{\theta})/g(\boldsymbol{\theta})$ if $f(\boldsymbol{\theta})/g(\boldsymbol{\theta}) > M$. This loop stopped only after at least 100k samples without updating M were reached. We then used M, f, and g to generate 10k reference posterior samples. We found that the NSF-based proposal distribution resulted in high acceptance rates. We used this custom scheme rather than relying on MCMC directly, since we found that standard MCMC approaches (Slice Sampling, HMC, and NUTS) all struggled with multi-modal posteriors and wanted to avoid bias in the reference samples, e.g. due to correlations in MCMC chains.

As a sanity check, we ran this scheme twice on all tasks and observation and found that the resulting reference posterior samples were indistinguishable in terms of C2ST.

B.2 Code

We provide `sbibm`, a benchmarking framework that implements all tasks, reference posteriors, different metrics and tooling to run and analyse benchmark results at scale. The framework is available at:

github.com/sbi-benchmark/sbibm

We make benchmarking new algorithms maximally easy by providing an open, modular framework for *integration* with SBI toolboxes. We here evaluated algorithms implemented in `pyABC` (Klinger et al., 2018), `pyabcranger` (Collin et al., 2020), and `sbi` (Tejero-Cantero et al., 2020). We emphasize that the goal of `sbibm` is orthogonal to any toolbox: It could easily be used with other toolboxes, or even be used to compare results for the same algorithm implemented by different ones. There are currently several SBI toolboxes available or under active development. `elfi` (Lintusaari et al., 2018) is a general purpose toolbox, including ABC algorithms as well as BOLFI (Gutmann and Corander, 2016). There are many toolboxes for ABC algorithms, e.g., `abcpy` (Dutta et al., 2017), `astroABC` (Jennings and Madigan, 2017), `CosmoABC` (Ishida et al., 2015), see also Kousathanas et al. (2018) for an overview. `carl` (Louppe et al., 2016) implements the algorithm by Cranmer et al. (2015). `hypothesis` (Hermans, 2019), and `pydelfi` (Alsing, 2019) are SBI toolboxes under development.

B.3 Reproducibility

To ensure reproducibility of our results, we publicly released all code including instructions on how to run the benchmark on cloud-based infrastructure.

F Figures

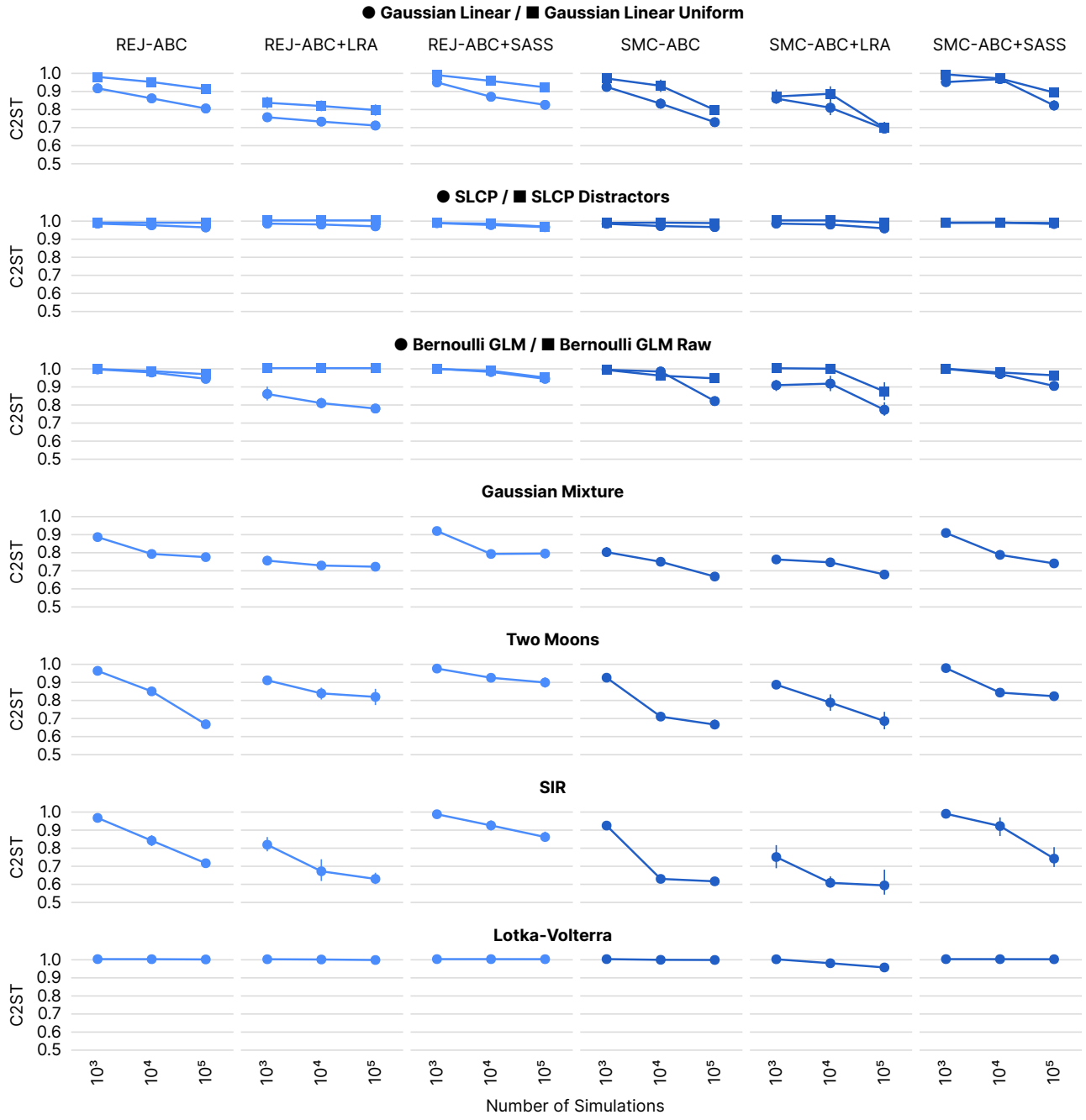

Figure 1: **Additional ABC results with linear regression adjustment (LRA) and semi-automatic summary-statistics (SASS).** We ran ABC with post-hoc LRA (Beaumont et al., 2002; Blum, 2018). On some tasks, this led to an improvement relative to versions without post-hoc adjustment. On Two Moons (bimodal posterior), linear adjustment decreased performance. We implemented our own SASS (Prangle et al., 2014b) with a third order polynomial feature expansion, and observed similar performance as with the implementation in `abcpy` toolbox (Dutta et al., 2017).

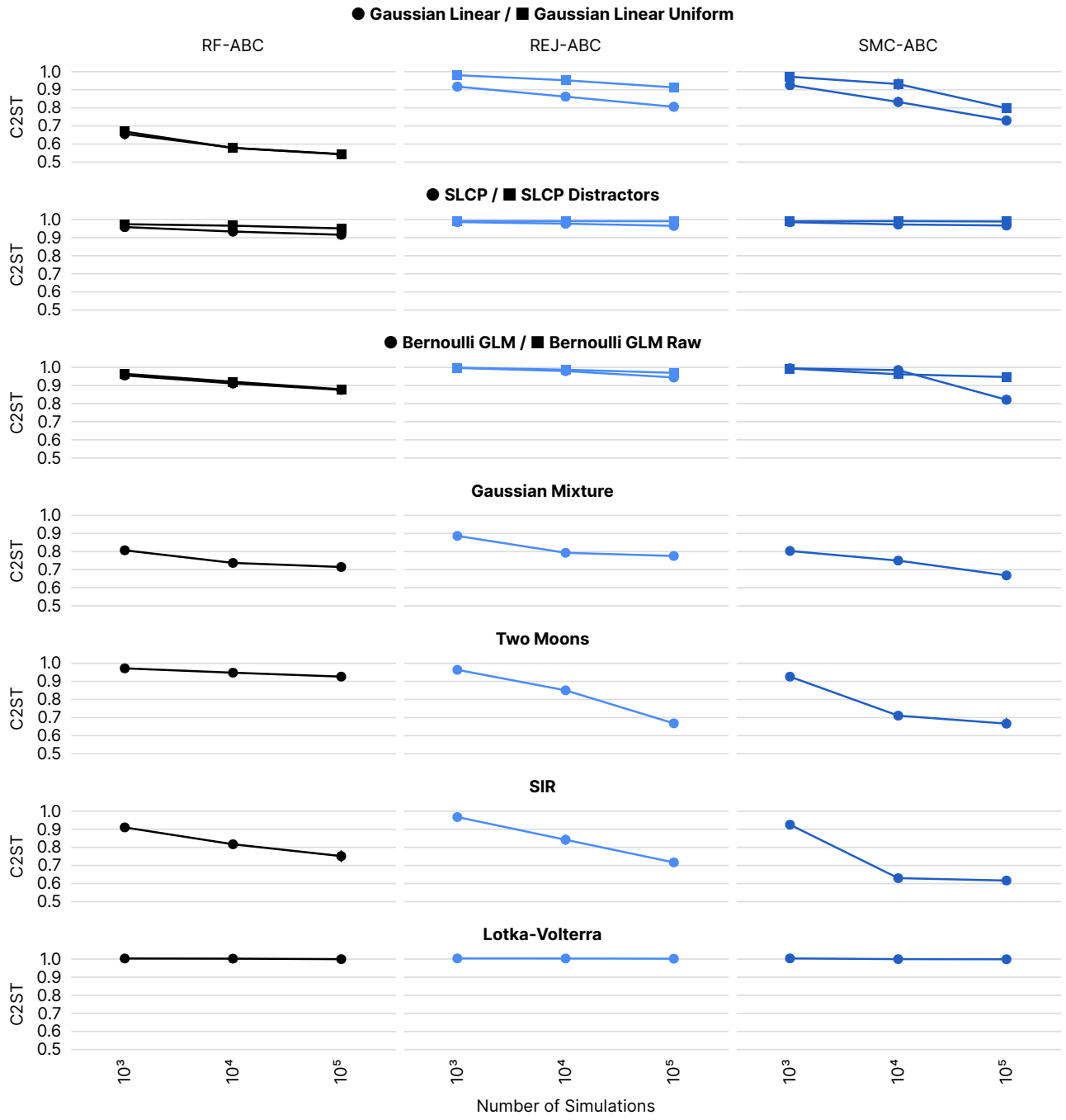

Figure 2: **RF-ABC results.** Results for RF-ABC (as described in A.9) compared to REJ-ABC and SMC-ABC on all benchmark tasks, using C2ST. Note that RF-ABC predicts each parameter individually, i.e. effectively assumes the posterior to be factorized– this is only appropriate for the Gaussian Linear, Gaussian Linear Uniform, and Gaussian Mixture tasks. On other tasks, the posterior deviates markedly from being factorized, and therefore it is to be expected that RF-ABC performance is limited, even when using many samples. Each data point corresponds to the mean and 95% confidence interval across 10 observations.

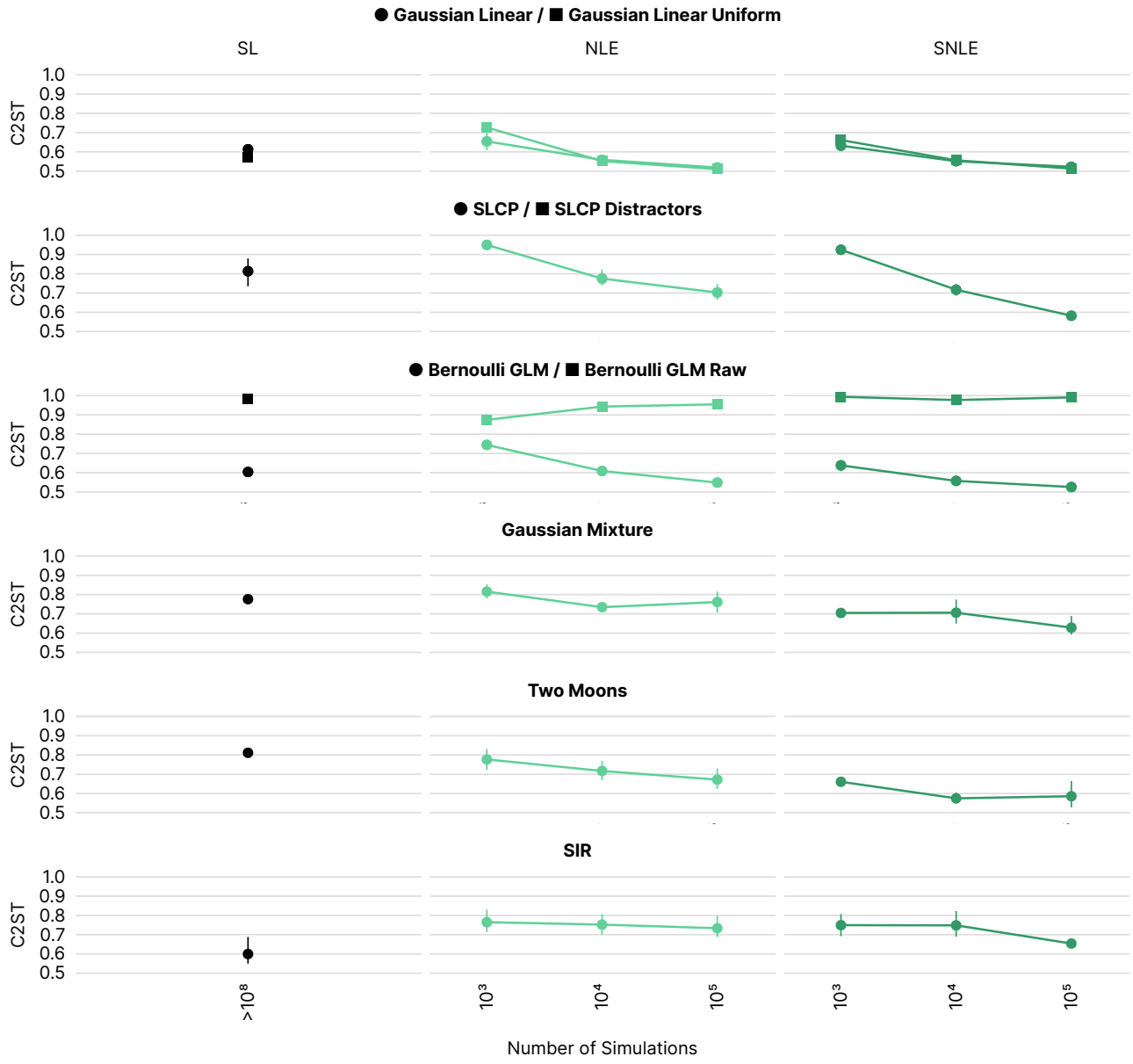

Figure 3: **SL results.** Results for SL compared to NLE and SNLE on benchmark tasks in terms of C2ST. Note that SL performs simulations at every MCMC step to approximate a Gaussian likelihood (see A.10 for details), and therefore it does not produce sensible results with the simulation budgets of other algorithms (between 1k and 100k), . In our experiments, SL required on the order of 10^8 to 10^9 simulations. For the SLCP Distractors and Lotka-Volterra stable estimation of covariances was not possible, which is why these tasks were omitted (details in A.10). We do not report SL results in the main paper, given the huge difference in simulation budget. Each data point corresponds to the mean and 95% confidence interval across 10 observations.

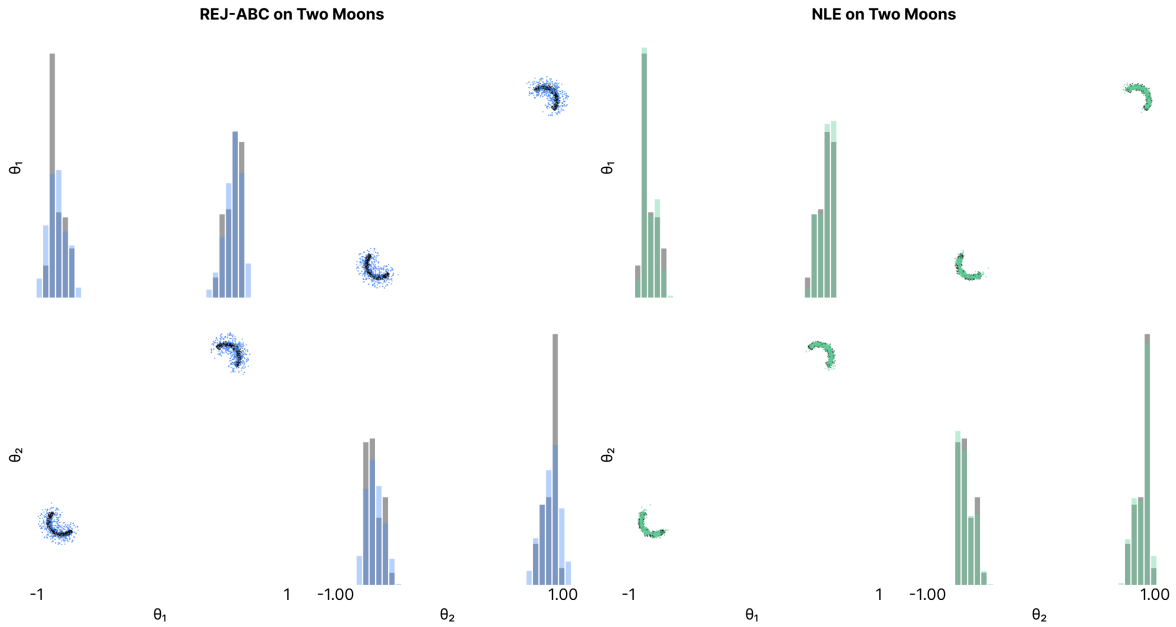

Figure 4: **MMD on Two Moons.** When using MMD with the median heuristic (as commonly done, including in SBI papers), MMD is slightly lower for the posterior obtained by REJ-ABC (left, blue samples), than for SNLE samples (right, green samples): 0.00729 (REJ-ABC) versus 0.00772 (NLE). This is at odds with the visual impression of the quality of the fit (reference samples in gray) as well as C2ST results: A classifier performed near chance level (.502) for SNLE samples while being able to tell apart REJ-ABC samples from the reference with accuracy 0.794. When manually choosing a length scale on the median distance of a *single crescent* (i.e., 0.09 instead of 1.78), MMD results were in agreement with C2ST results: 0.00738 (REJ-ABC) versus 0.00035 (SNLE), i.e., they also suggested a better fit for SNLE. In the main paper, we prefer to report C2ST because we found it less sensitive to hyperparameters: reliance on the commonly used median heuristic can be problematic on tasks with complex posterior structure, e.g., multi-modality in Two Moons, as demonstrated here. We refer the interested reader to Liu et al. (2020) for further illustrative examples of where MMD with Gaussian kernels can have limited power. We also want to point out that new kernel-based two sample tests are being actively developed which might make them easier to use on such problems in the future.

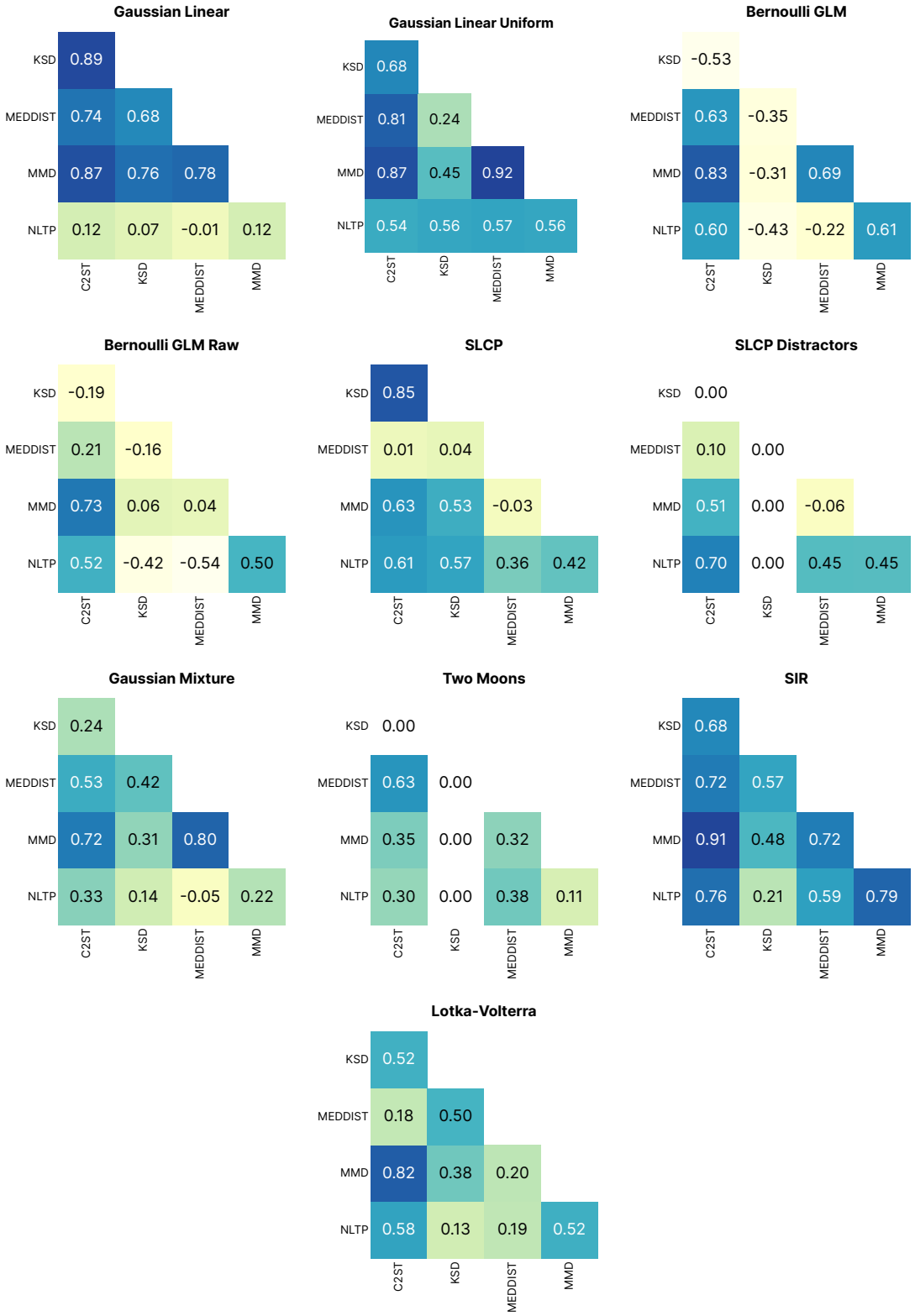

Figure 5: **Correlations between metrics for all tasks.** NLTP is the negative log probability of true parameters. Note that calculation of KSD was numerically unstable when calculating gradients for SLCP Distractors and Two Moons, resulting in correlation of zero for these tasks.

H Hyperparameter Choices

In this section, we address two central questions for any benchmark: (1) how hyperparameters are chosen and (2) how sensitive results are to the respective choices.

Rather than tuning hyperparameters on a per-task basis, we changed hyperparameters on multiple or all tasks at once and selected configurations that worked best across tasks. We wanted to avoid overfitting on individual benchmark tasks and were instead interested in settings that can generalize across multiple tasks. In practice, tuning an algorithm on a given task would typically be impossible, due to the lack of suitable metrics that can be computed without reference posteriors as well as high computational demands that SBI tasks often have.

To find good general settings, we performed more than 10 000 individual runs. We explored hyperparameter choices that have not been previously reported, and revealed substantial improvements. The benchmark offers the possibility to systematically compare different choices and design better and more robust SBI algorithms.

H.1 REJ-ABC

Classical ABC algorithms have crucial hyperparameters, most importantly, the distance metric and acceptance tolerance ϵ. We used our own implementation of REJ-ABC as it is straightforward to implement (see A.1). The distance metric was fixed to be the l_2-norm for all tasks and we varied different acceptance tolerances ϵ across tasks on which REJ-ABC performed sufficiently well. Our implementation of REJ-ABC is quantile based, i.e,. we select a quantile of the samples with the smallest distance to the observed data, which implicitly defines an ϵ. The 10k samples needed for the comparison to the reference posterior samples are then resampled from the selected samples. In order to check whether this resampling significantly impaired performance, we alternatively fit a KDE in order to obtain 10k samples.

Below, we show results for different schedules of quantiles for each simulation budget, e.g., a schedule of 0.1, 0.01, 0.001 corresponds to the 10, 1 and 0.1 percent quantile, or the top 100 samples for each simulation budget. Across tasks and budgets the 0.1, 0.01, 0.001 quantile schedule performed best (Fig. 6). Performance showed improvement by the KDE fit, especially on the Gaussian tasks. We therefore report the version using the top 100 samples and KDE in the main paper.

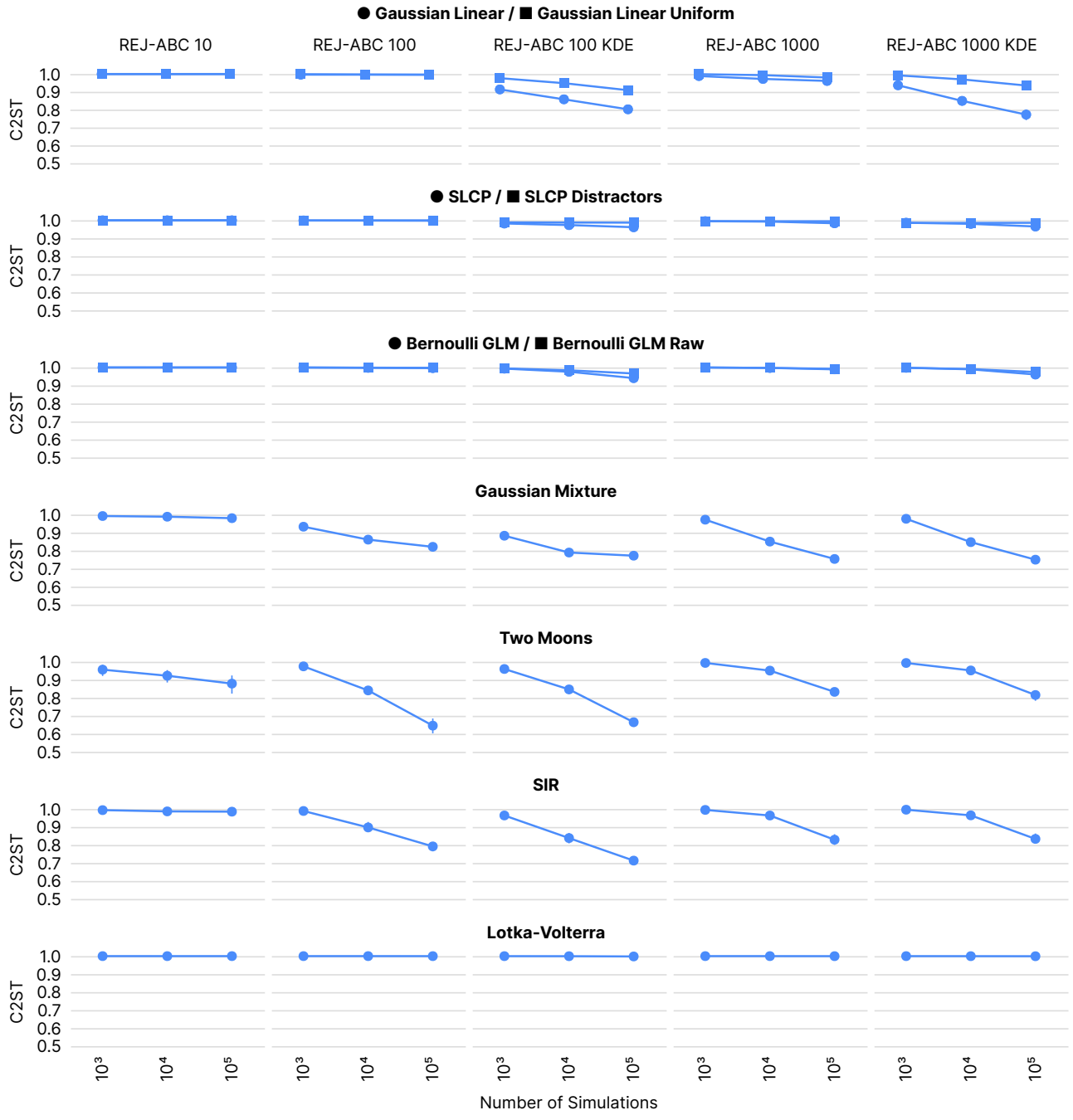

Figure 6: **Hyperparameter selection for REJ-ABC.** C2ST performance of different percentile schedules across simulation budgets (columns) for all tasks (rows). Top label for each plot column: number of samples retained, and optional KDE. Across tasks and budgets, the schedule of 0.1, 0.01, 0.001 percentiles, which corresponds to the top 100 samples closest to the observation, performed best. Each data point corresponds to the mean and 95% confidence interval across 10 observations.

H.2 SMC-ABC

SMC-ABC has several hyperparameters including the population size, the perturbation kernel, the epsilon schedule and the distance metric. In order to ensure that we report the best possible SMC-ABC results for a fair comparison, we sweeped over three hyperparameters that are especially critical: the population size, the quantile used to select the epsilon from the distances of the particles of the previous population, and the scaling factor of the covariance of the Gaussian perturbation kernel. The remaining hyperparameters were fixed to values common in the literature: Gaussian perturbation kernel and $l2$-norm distance metric.

Additionally, we compared our implementation against one from the popular pyABC toolbox (Klinger et al., 2018) to which we refer as versions A and B respectively. We sweeped over these hyperparameters and optionally added a post-hoc KDE fit for drawing the samples needed for two-sample based performance metrics.

Overall, the parameter setting with a population size of 100, a kernel covariance scale of 0.5, and an epsilon quantile 0.2 performed best. Although the results of the two different implementations were qualitatively very similar (compare Fig. 7 and Fig. 8, respectively), version A was slightly better on the Gaussian tasks. Although we tried to match the implementations and the exact settings, there are small differences between the two, which might explain the difference in the results: Implementation B constructs the Gaussian perturbation kernel using kernel density estimation on the weighted samples of the previous population, whereas A constructs it using the mean and covariance estimated from samples from the previous population. The latter could be advantageous in case of a Gaussian-like (high-dimensional) posterior (Gaussian Mixture and Gaussian linear task) and disadvantageous in a non-Gaussian-like posteriors (e.g., Two Moons). We decided to report results for SMC-ABC in the main paper using implementation A (ours) with population size 100 for simulation budgets 1k and 10k, and population size 1000 for simulation budget 100k, a kernel covariance scale of 0.5, and epsilon quantile 0.2. This choice of kernel covariance scale is different from recommendations in the literature (Sisson et al., 2007; Beaumont et al., 2009). We only found very small performance differences for different scales and note that our choice is in line with the recommendation of the `pyABC` toolbox (pyABC API Documentation, 2020), i.e., using a scale between 0 and 1. Performance showed improvement by the KDE fit, especially on the Gaussian tasks. We therefore report the version with KDE in the main paper.

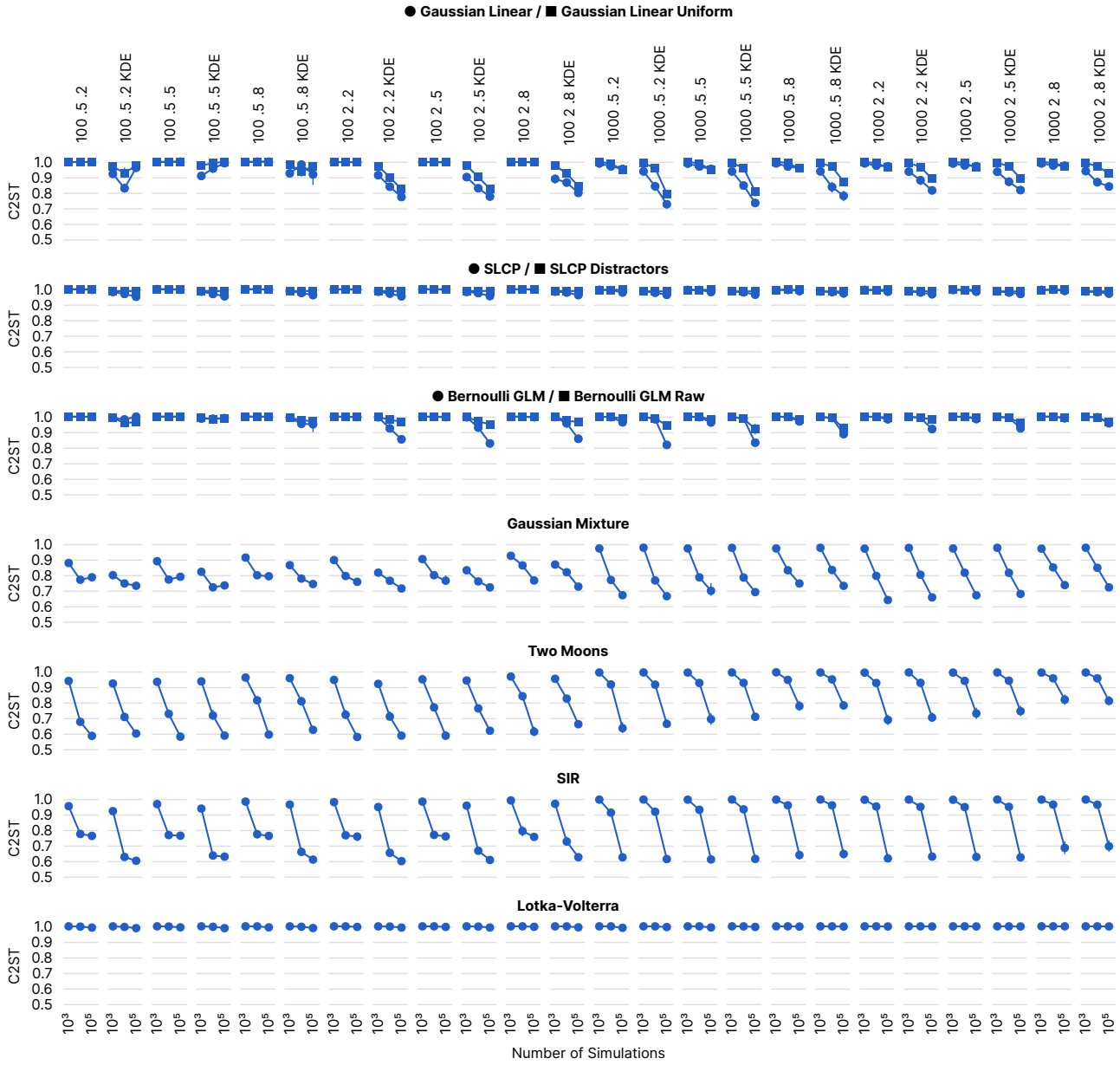

Figure 7: **Hyperparameter selection for SMC-ABC with our implementation**. Top label for each plot column: population size, kernel covariance scale, epsilon quantile/epsilon-decay parameter, and optional KDE. Each data point corresponds to the mean and 95% confidence interval across 10 observations.

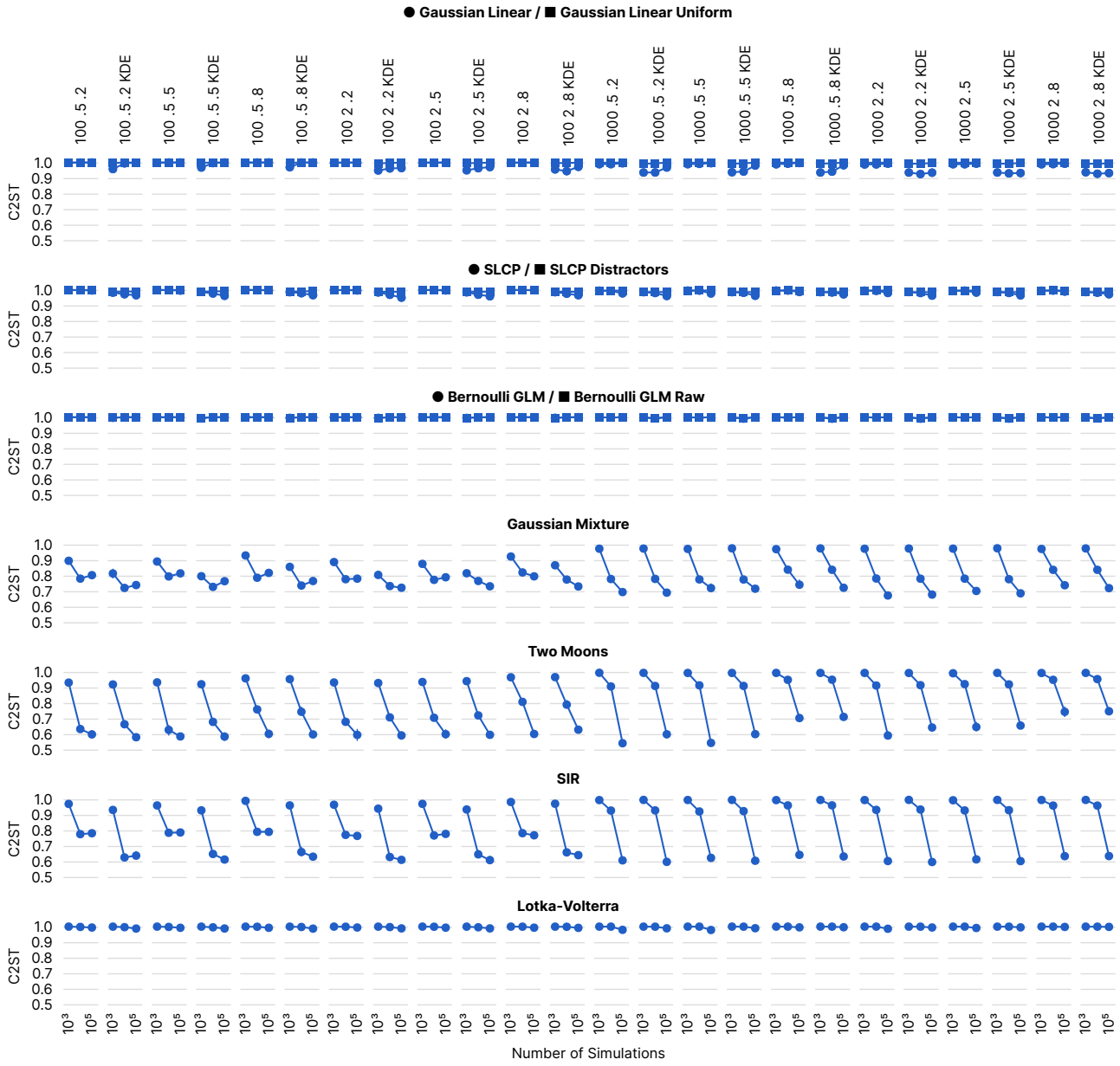

Figure 8: **Hyperparameter selection for SMC-ABC. with `pyABC` implementation**. Top label for each plot column: population size, kernel covariance scale, epsilon quantile/epsilon-decay parameter, and optional KDE. Each data point corresponds to the mean and 95% confidence interval across 10 observations.

H.3 MCMC for (S)NLE and (S)NRE

(S)NLE and (S)NRE both rely on MCMC sampling, which has several hyperparameters. In line with Papamakarios et al. (2019b) and Durkan et al. (2020), we used Slice Sampling (Neal, 2003). However, we modified the MCMC schemes used in these papers and obtained significant improvements in performance and speed.

Number of chains and initialization. While Papamakarios et al. (2019b); Durkan et al. (2020) used a single chain with axis-aligned updates, we found that on tasks with multi-modal posteriors, it can be essential to run multiple MCMC chains in order to sample all modes. Performance on Two Moons, for example, was poor with a single chain, since usually only one of the crescent shapes was sampled. Rather than initialising chains by drawing initial locations from the prior, we found the resampling scheme as described in A.3 to work better for initialisation, and used 100 chains instead of a single one.

Transformation of variables. When implementing MCMC, it is common advice to transform problems to have unbounded support (Hogg and Foreman-Mackey, 2018), although this has not been discussed in SBI papers or implemented in accompanying code. We found that without this transformation, MCMC sampling could get stuck in endless loops, e.g., on the Lotka-Volterra task. Apart from the transformation to unbounded space, we found z-scoring of parameters and data to be crucial for some tasks.

Vectorization of MCMC sampling. We reimplemented Slice Sampling so that all chains could perform likelihood evaluations in parallel. Evaluating likelihoods, e.g., in the case of (S)NLE, requires passes through a flow-based density estimator, which is significantly faster when batched. This allowed us to sample all chains in parallel rather than sequentially and yielded huge speed-ups: For example, SNLE on Gaussian Linear took more than 36 hours on average for 100k simulations without vectorization, and less than 2 hours with vectorization.

H.4 Density estimator for (S)NLE

Approaches based on neural networks (NN) tend to have many hyperparameters, including the concrete type of NN architecture and hyperparameters for training. We strove to keep our choices close to Durkan et al. (2020), which are the defaults in the toolbox we used (`sbi`, Tejero-Cantero et al., 2020).

While Papamakarios et al. (2019b); Durkan et al. (2020) used Masked Autoregressive Flows (MAFs, Papamakarios et al., 2017) for density estimation, we explored how results change when using Neural Spline Flows (NSFs, Durkan et al., 2019) for density estimation. These results are shown in Fig. 9.

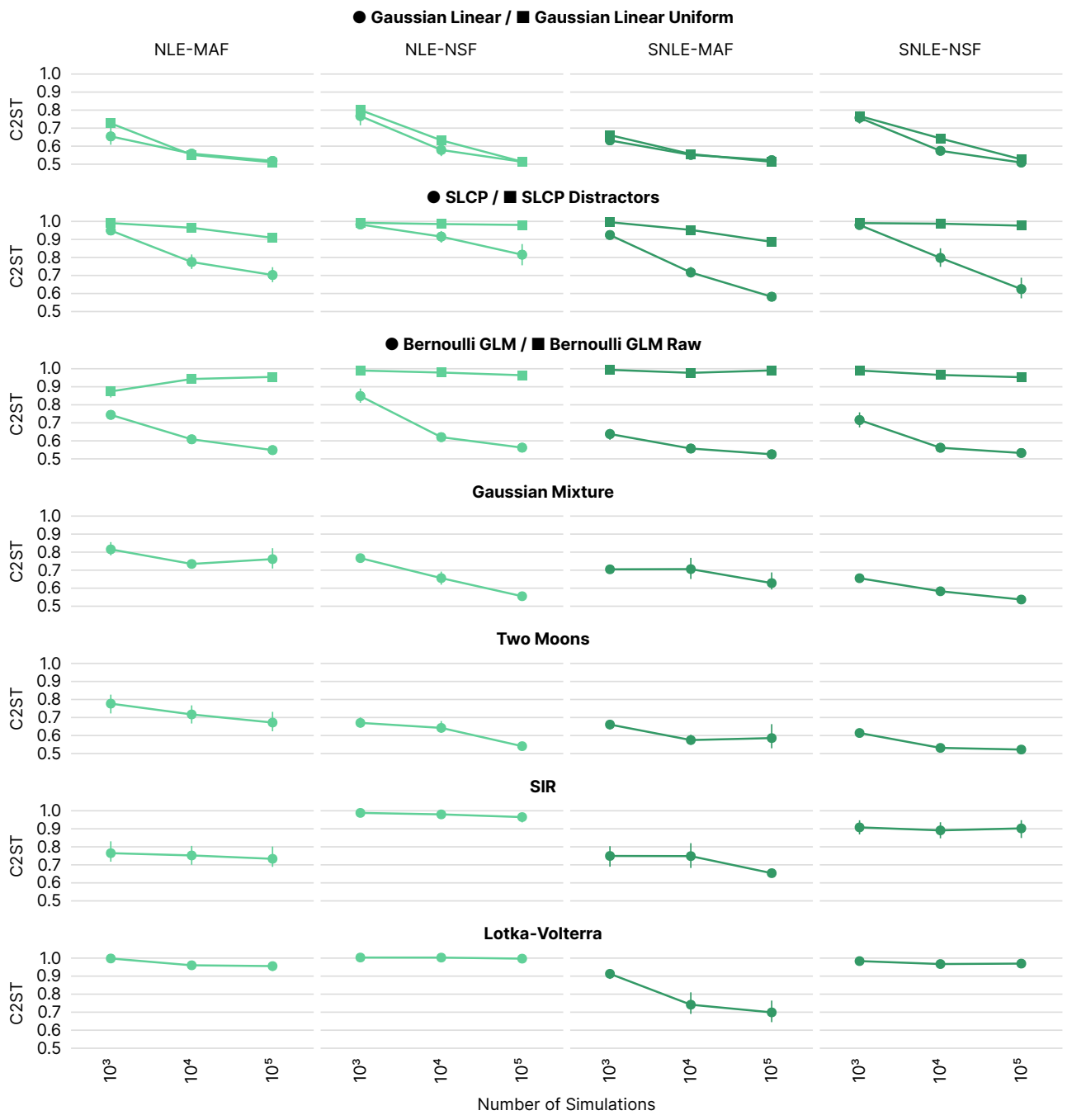

Figure 9: **Density estimator selection for (S)NLE.** Performance of (S)NLE in terms of C2ST across tasks using MAFs or NSFs for density estimation. Considering all tasks, NSFs generally performed worse, e.g., using NSFs significantly reduced performance on SIR and Lotka-Volterra, indicating that the added flexibility of NSFs was not needed for (S)NLE. We thus reported performance using MAFs in the main paper. Each data point corresponds to the mean and 95% confidence interval across 10 observations.

H.5 Density estimator for (S)NPE

We performed the analogous experiments for (S)NPE as for (S)NLE: Here, we found NSFs to increase performance relative to MAFs (Fig. 10). When directly estimating the posterior distribution, especially on tasks with complex multi-modal structure like Two Moons or SLCP, the additional flexibility offered by NSFs improved performance. With NSFs, artifacts from density transformation that were visible e.g. in Two Moons posteriors, vanished. To our knowledge, results on (S)NPE with NSFs have not been previously published.

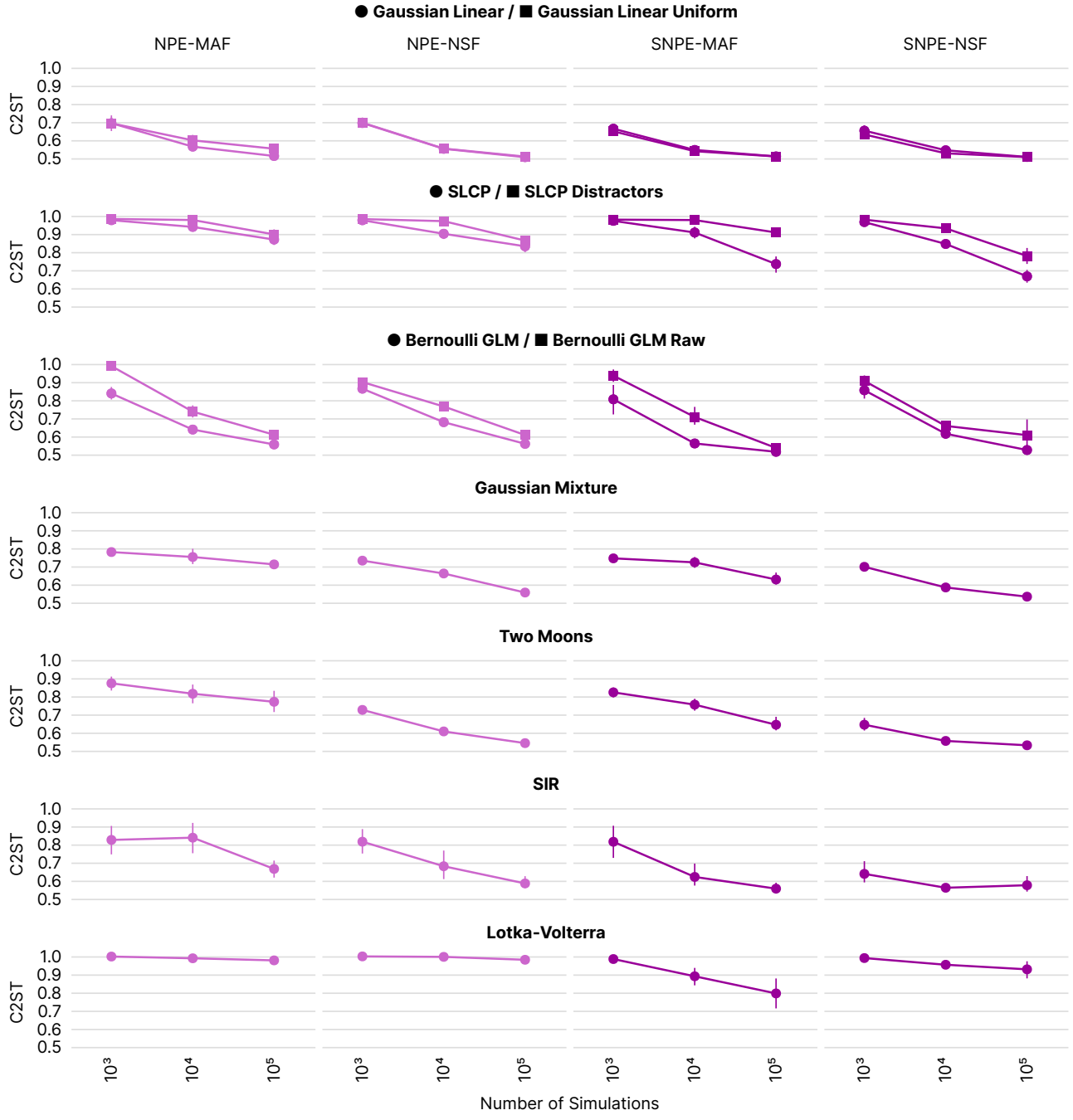

Figure 10: **Density estimator selection for (S)NPE.** Performance of (S)NPE in terms of C2ST across tasks using MAFs or NSFs for density estimation. Considering all tasks, NSFs generally performed better, especially on Gaussian Mixture, Two Moons, and SIR. We thus reported performance using NSFs in the main paper. Each data point corresponds to the mean and 95% confidence interval across 10 observations.

H.6 Classifier choice for (S)NRE

For (S)NRE, we compared two different choices of classifier architectures: an MLP and a ResNet architecture, as described in A.7. While results were similar for most tasks (Fig. 11), we decided to use the ResNet architecture in the main paper due to the better performance on Two Moons and SIR for low to medium simulation budgets.

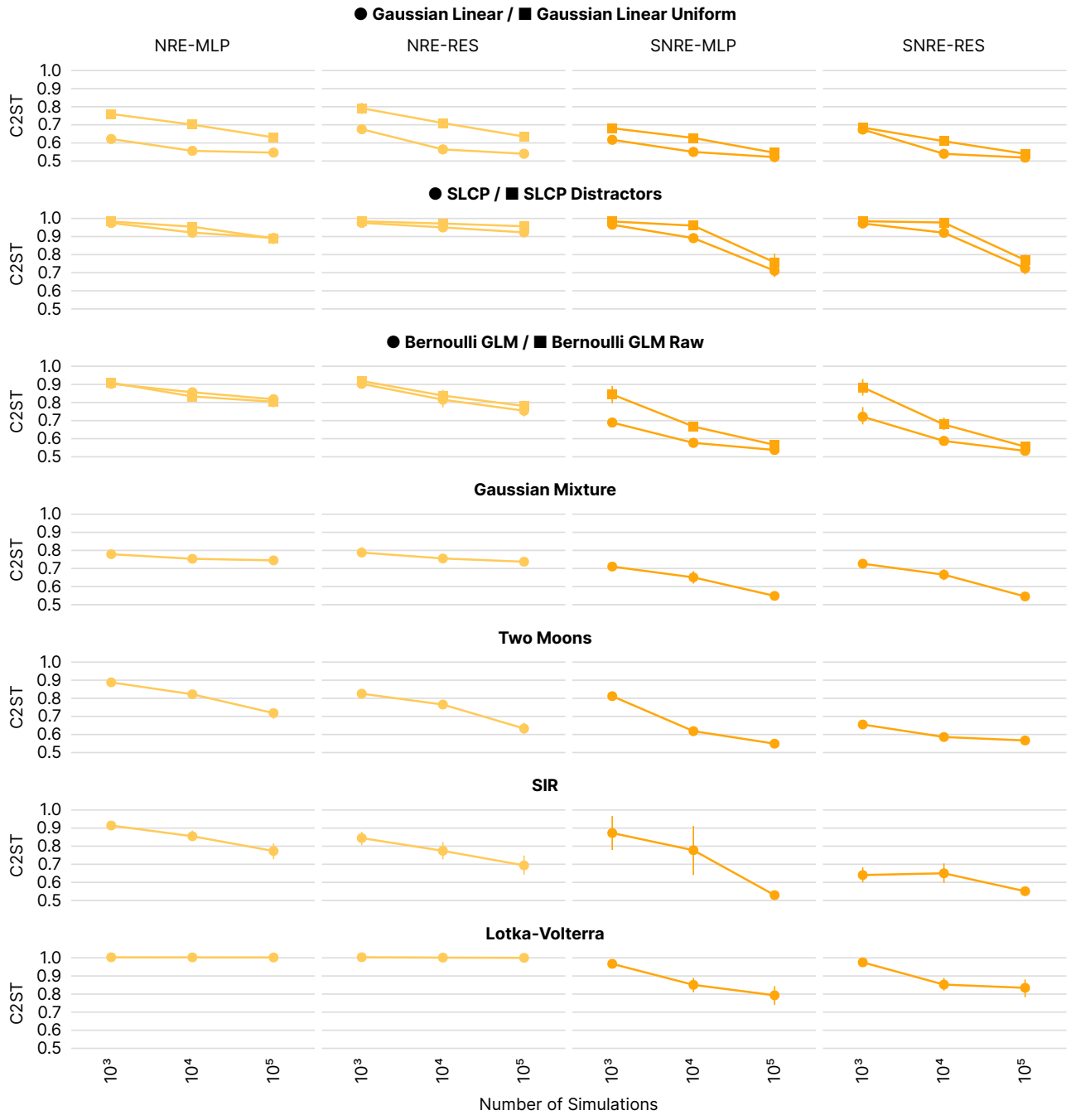

Figure 11: **Classifier architecture for (S)NRE.** Performance of (S)NRE in terms of C2ST across tasks using MLPs or ResNets for classification. Considering all tasks, ResNets generally performed better, especially on Two Moons and SIR. We thus reported performance using ResNets in the main paper. Each data point corresponds to the mean and 95% confidence interval across 10 observations.

M Metrics

M.1 Negative log probability of $\boldsymbol{\theta}_o$ (NLTP)

In simulation-based inference, the average negative log likelihood of true parameters $-\mathbb{E}[\log q(\boldsymbol{\theta}_o|\mathbf{x}_o)]$ (NLTP) is commonly reported as a performance metric in the literature (Papamakarios and Murray, 2016; Durkan et al., 2018; Papamakarios et al., 2019b; Greenberg et al., 2019; Hermans et al., 2020; Durkan et al., 2020). An attractive property of this metric is that the access to the ground-truth posterior is not required.

It is important to point out, however, that calculating this metric on a single or small number of pairs $(\boldsymbol{\theta}_o, \mathbf{x}_o)$ is problematic. To illustrate the issue, consider the following example (as discussed in Talts et al. (2018)): Consider $\theta \sim \mathcal{N}(0, 1^2), x|\theta \sim \mathcal{N}(\theta, 1^2)$, and a single pair $(\boldsymbol{\theta}_o, \mathbf{x}_o)$ with $\theta_o = 0$ and an implausible (but possible) $x_o = 2.1$. In this case, the true posterior is $\mathcal{N}(\theta|1.05, 0.5^2)$ under which the θ_o has low probability since it is more than two standard deviations away from the posterior mean. If an algorithm fitted a wrong posterior, e.g., by overestimating the standard deviation as 1 instead of 0.5, the probability of θ_o under the estimated posterior would be higher than under the true posterior.

Therefore, a large number of pairs $(\boldsymbol{\theta}_o, \mathbf{x}_o)$ should be used. Indeed, in the limit of infinite number of pairs $(\boldsymbol{\theta}_o, \mathbf{x}_o)$, the metric converges to a D_{KL}:

$$
\begin{aligned}
&\mathbb{E}_{\boldsymbol{\theta}_o \sim p(\boldsymbol{\theta})} \mathbb{E}_{\mathbf{x}_o \sim p(\mathbf{x}|\boldsymbol{\theta}_o)} \big[-\log q(\boldsymbol{\theta}_o|\mathbf{x}_o) \big] \\
&= \mathbb{E}_{\mathbf{x}_o \sim p(\mathbf{x}), \boldsymbol{\theta}_o \sim p(\boldsymbol{\theta}|\mathbf{x}_o)} \big[-\log q(\boldsymbol{\theta}_o|\mathbf{x}_o) \big] \\
&= \mathbb{E}_{\mathbf{x}_o \sim p(\mathbf{x}), \boldsymbol{\theta}_o \sim p(\boldsymbol{\theta}|\mathbf{x}_o)} \big[-\log q(\boldsymbol{\theta}_o|\mathbf{x}_o) + \log p(\boldsymbol{\theta}_o|\mathbf{x}_o) \big] - \mathbb{E}_{\mathbf{x}_o \sim p(\mathbf{x}), \boldsymbol{\theta}_o \sim p(\boldsymbol{\theta}|\mathbf{x}_o)} \big[\log p(\boldsymbol{\theta}_o|\mathbf{x}_o) \big] \\
&= \mathbb{E}_{\mathbf{x}_o \sim p(\mathbf{x})} D_{\mathrm{KL}}(p(\boldsymbol{\theta}|\mathbf{x}_o) || q(\boldsymbol{\theta}|\mathbf{x}_o)) + \mathbb{E}_{\mathbf{x}_o \sim p(\mathbf{x})} \mathbb{H}(p(\boldsymbol{\theta}|\mathbf{x}_o))
\end{aligned}
$$

The first term in the final equation is the average D_{KL} between true and approximate posteriors over all observations $\mathbf{x}_o$ that can be generated when sampling parameters $\boldsymbol{\theta}_o$ from the prior. The second term, the entropy term, would be the same for all algorithms compared.

In the context of this benchmark, we decided against using the probability of $\boldsymbol{\theta}_o$ as a metric: For all algorithms that are not amortized (all but one), evaluating posteriors at different $\mathbf{x}_o$ would require rerunning inference. As the computational requirements for running the benchmark at 10 observations per task are already high, running tasks for hundreds of observations would become prohibitively expensive.

M.2 Simulation-based calibration (SBC)

In simulation-based calibration (SBC), samples $\boldsymbol{\theta}'$ are drawn from the data-averaged posterior, i.e., the posterior obtained by running inference for many observations. When the posterior approximation is exact, $\boldsymbol{\theta}'$ is distributed according to the prior (Talts et al., 2018).

Let us briefly illustrate this: In SBC, we draw $\boldsymbol{\theta} \sim p(\boldsymbol{\theta}), \mathbf{x} \sim p(\mathbf{x}|\boldsymbol{\theta}), \boldsymbol{\theta}' \sim q(\boldsymbol{\theta}'|\mathbf{x})$, which implies a joint distribution $\pi(\boldsymbol{\theta}, \mathbf{x}, \boldsymbol{\theta}') = p(\boldsymbol{\theta})p(\mathbf{x}|\boldsymbol{\theta})q(\boldsymbol{\theta}'|\mathbf{x})$. The marginal $\pi(\boldsymbol{\theta}')$ is then:

$$
\pi(\boldsymbol{\theta}') = \int\int p(\boldsymbol{\theta})p(\mathbf{x}|\boldsymbol{\theta})q(\boldsymbol{\theta}'|\mathbf{x}) \, \mathrm{d}\mathbf{x} \, \mathrm{d}\boldsymbol{\theta} = \int\int p(\boldsymbol{\theta}, \mathbf{x})q(\boldsymbol{\theta}'|\mathbf{x}) \, \mathrm{d}\mathbf{x} \, \mathrm{d}\boldsymbol{\theta} = \int p(\mathbf{x}) \; q(\boldsymbol{\theta}'|\mathbf{x}) \, \mathrm{d}\mathbf{x}.
$$

If the approximate posterior is the true posterior, the marginal on $\boldsymbol{\theta}'$ is equal to the prior: If $q(\boldsymbol{\theta}'|\mathbf{x}) = p(\boldsymbol{\theta}'|\mathbf{x})$, then $\pi(\boldsymbol{\theta}') = \int p(\mathbf{x}, \boldsymbol{\theta}') \, \mathrm{d}\mathbf{x} = p(\boldsymbol{\theta}')$, i.e., one can set up a consistency test that is based on the distribution of $\boldsymbol{\theta}'$ samples. Talts et al. (2018) do this by using frequentist tests per dimension.

Note that SBC as described above is merely a consistency check. For example, if the approximate posterior were the prior, a calibration test as described above would not be able to detect this. This is a realistic failure mode in simulation-based inference. It could happen with rejection ABC in the limit $\epsilon \to \infty$, or when learned summary statistics have no information about $\boldsymbol{\theta}$. One way around this is issue is proposed in Prangle et al. (2014a), who propose to restrict observations to a subset of all possible $\mathcal{X}$.

SBC is similar to the average negative log likelihood of true parameters described above, in that inference needs to be carried out for many observations generated by sampling from the prior. Running inference for hundreds of observations would become prohibitively expensive in terms of compute for most algorithms, which is why we do not rely on SBC in the benchmark.

M.3 Median distance (MEDDIST)

Posterior predictive checks (PPCs) use the posterior predictive distribution to predict new data, $\mathbf{x}' \sim p(\mathbf{x}'|\mathbf{x}_o) = \int p(\mathbf{x}'|\boldsymbol{\theta})q(\boldsymbol{\theta}|\mathbf{x}_o)\,\mathrm{d}\boldsymbol{\theta}$. The observed data $\mathbf{x}_o$ should look plausible under the posterior predictive distribution (Gelman et al. (2004), chapter 6). A particular PPC, used for example in Papamakarios et al. (2019b); Greenberg et al. (2019); Durkan et al. (2020), is to assess the median L2 distance between N' posterior predictive samples $\mathbf{x}'$ and $\mathbf{x}_o$. The median is used since the mean would be more sensitive to outliers.

In the benchmark, we refer to this metric as median distance (MEDDIST) and drew $N' = 10000$ samples from each posterior predictive distribution to compute it. In contrast with other metrics considered here, the median distance is computed in the space of data $\mathbf{x}$ and requires additional simulations (which could be expensive, depending on the simulator). The median distance should be considered a mere check rather than a metric and it does not necessarily test the structure of the estimated posterior.

M.4 Maximum Mean Discrepancy (MMD)

Maximum Mean Discrepancy (MMD) is an Integral Probability Metric (IPM). Linear and quadratic time estimates for using MMD as a two-sample test were derived in Gretton et al. (2012). MMD has been commonly used in the SBI literature with Gaussian kernels (Papamakarios et al., 2019b; Greenberg et al., 2019; Hermans et al., 2020), setting a single length-scale hyperparameter by using a median heuristic (Ramdas et al., 2015). We follow the same procedure, i.e., use Gaussian kernels with length-scale determined by the median heuristic on reference samples. MMDs are calculated using 10k samples from reference and approximate posteriors.

If simple kernels are used to compare distributions with complex, multimodal structure, distinct distributions can be mapped to nearby mean embeddings, resulting in low test power. On SLCP and Two Moons, for example, we found a translation-invariant kernel to be limiting, since it cannot adapt to the local structure (see Suppl. Fig. 4). This is reflected in the low correlation of MMD and C2ST (Suppl. Fig. 5). We emphasize that these issues are strictly related to simple kernels with hyperparameters commonly used in the literature. Posteriors of the Two Moons task have a structure similar to the blobs example of Liu et al. (2020), who argue for using learned kernels to overcome the aforementioned problem.

M.5 Classifier-based tests (C2ST)

In classifier-based testing, a classifier is trained to distinguish samples of the true posterior $p(\boldsymbol{\theta}|\mathbf{x}_o)$ from samples of the estimated posterior $q(\boldsymbol{\theta}|\mathbf{x}_o)$. If the samples are indistinguishable, the classification performance should be at chance level, 0.5. Practical use and properties of classifier-based 2-sample testing (C2ST) are discussed in Lopez-Paz and Oquab (2017) (see Gutmann et al., 2018; Dalmasso et al., 2020, for examples in the context of SBI).

To compute C2ST, we trained a two-layer neural network with 10 times as many ReLU units as the dimensionality of parameters, and optimize with Adam (Kingma and Ba, 2015). Classifiers were trained on 10k z-scored samples from reference and approximate posterior each. Classification accuracy was reported using 5-fold cross-validation.

M.6 Kernelized Stein Discrepancy (KSD)

Kernelized Stein Discrepancy (KSD) is a 1-sample goodness-of-fit test proposed independently by Chwialkowski et al. (2016) and Liu et al. (2016). KSD tests samples from algorithms against the gradient of unnormalized true posterior density, $\nabla_{\boldsymbol{\theta}}\, \tilde{p}(\boldsymbol{\theta}|\mathbf{x}_o)$. We used KSD with Gaussian kernels, setting the length-scale through the median heuristic, and 10k samples from each algorithm.

R Runtimes

In applications of SBI, simulations are commonly assumed to be the dominant cost. In order to make the benchmark feasible at this scale, we focused on simple simulators and optimized runtimes, e.g. we developed a new package bridging `DifferentialEquations.jl` (Rackauckas and Nie, 2017; Bezanson et al., 2017) and `PyTorch` (Paszke et al., 2019) so that generating simulations for all implemented tasks is extremely fast. This differs from many cases in practice, where the runtime costs for an algorithm are often negligible compared to the cost of simulations. Having said that, algorithms show significant differences in runtime costs, which we measured and report here.

We recorded runtimes for all algorithms on all tasks. In principle, runtimes could be reduced by employing multi-CPU architectures, however, we decided for the single CPU setup to accurately compare runtimes across all algorithms and tasks. We did not employ GPUs for training neural-networks (NN). This is because the type of NNs used in the algorithms currently in the benchmark do not benefit much from GPU versus CPU training (e.g., no CNN architecture, rather shallow and narrow networks). In fact, running SNPE on SLCP using a GeForce GTX 1080 showed slightly longer runtimes than on CPU, due to the added overhead resulting from copying data back and forth to the device. Therefore, it was more economical and comparable to run the benchmark on CPUs.

All neural network-based algorithms were run on single 3.6 GHz CPU cores of AWS C5-instances. ABC algorithms were run on single CPU cores of an internal cluster with 2.4 GHz CPUs. We observed a difference in runtimes of less than 100ms when running ABC algorithms on the same hardware as used for neural network-based algorithms.

Figure 12 shows the recorded runtimes in minutes. We observed short runtimes for REJ-ABC and SMC-ABC, as these do not require NN training or MCMC. The sequential versions of all three NN-based algorithms yielded longer runtimes than the non-sequential versions because these involve 10 rounds of NN training. Among the sequential algorithms, SNPE showed the longest runtimes. Runtimes with MAFs instead of NSFs tend to be faster, e.g. the difference between MAFs and NSFs using SNPE on SLCP at 100k simulations was about 50 minutes on average. We also emphasize that the speed of (S)NLE reported here was only obtained after vectorizing MCMC sampling. Without vectorization, runtime on the Gaussian Linear for SNLE was more than 36 hours instead of less than 2 hours (see Appendix H).

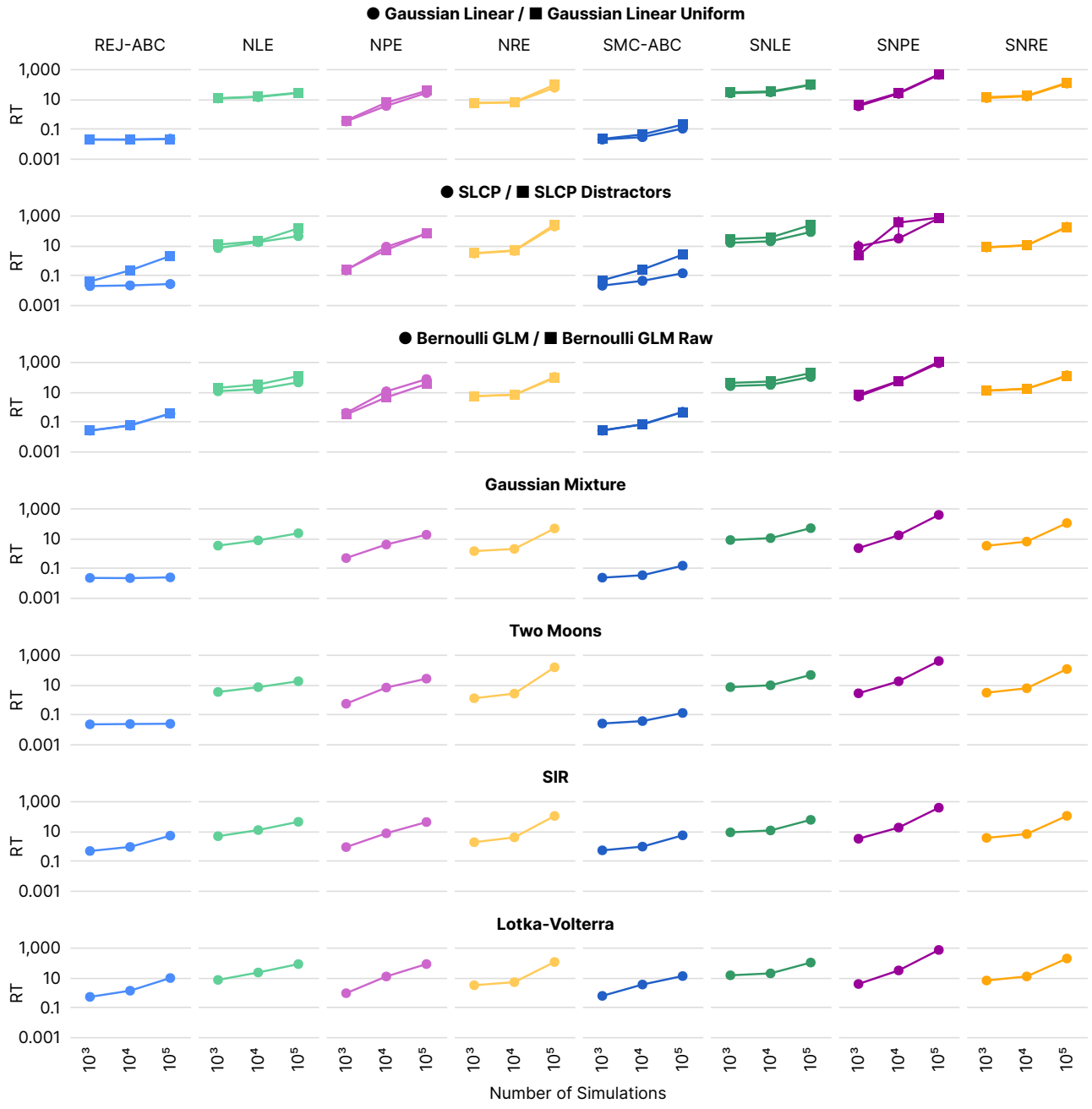

Figure 12: **Runtime on benchmark tasks.** Runtime of REJ-ABC, SMC-ABC, NLE, SNLE, NPE, SNPE, NRE, SNRE in minutes, for 10 observations each, means and 95% confidence intervals. Each run was allocated a single CPU core, see Appendix R for details.

T Tasks

T.1 Gaussian Linear

Inference of the mean of a 10-d Gaussian model, in which the covariance is fixed. The (conjugate) prior is Gaussian:

Prior $\mathcal{N}(\mathbf{0}, 0.1 \odot \mathbf{I})$

Simulator $\mathbf{x}|\boldsymbol{\theta} \sim \mathcal{N}(\mathbf{x}|\mathbf{m}_{\boldsymbol{\theta}} = \boldsymbol{\theta}, \mathbf{S} = 0.1 \odot \mathbf{I})$

Dimensionality $\boldsymbol{\theta} \in \mathbb{R}^{10}, \mathbf{x} \in \mathbb{R}^{10}$

T.2 Gaussian Linear Uniform

Inference of the mean of a 10-d Gaussian model, in which the covariance is fixed. The prior is uniform:

Prior $\mathcal{U}(-\mathbf{1}, \mathbf{1})$

Simulator $\mathbf{x}|\boldsymbol{\theta} \sim \mathcal{N}(\mathbf{x}|\mathbf{m}_{\boldsymbol{\theta}} = \boldsymbol{\theta}, \mathbf{S} = 0.1 \odot \mathbf{I})$

Dimensionality $\boldsymbol{\theta} \in \mathbb{R}^{10}, \mathbf{x} \in \mathbb{R}^{10}$

T.3 SLCP

A challenging inference task designed to have a simple likelihood and a complex posterior. The prior is uniform over five parameters $\boldsymbol{\theta}$ and the data are a set of four two-dimensional points sampled from a Gaussian likelihood whose mean and variance are nonlinear functions of $\boldsymbol{\theta}$:

Prior $\mathcal{U}(-\mathbf{3}, \mathbf{3})$

Simulator $\mathbf{x}|\boldsymbol{\theta} = (\mathbf{x}_1, \ldots, \mathbf{x}_4)$, $\mathbf{x}_i \sim \mathcal{N}(\mathbf{m}_{\boldsymbol{\theta}}, \mathbf{S}_{\boldsymbol{\theta}})$,
where $\mathbf{m}_{\boldsymbol{\theta}} = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}$, $\mathbf{S}_{\boldsymbol{\theta}} = \begin{bmatrix} s_1^2 & \rho s_1 s_2 \\ \rho s_1 s_2 & s_2^2 \end{bmatrix}$, $s_1 = \theta_3^2, s_2 = \theta_4^2, \rho = \tanh \theta_5$

Dimensionality $\boldsymbol{\theta} \in \mathbb{R}^5, \mathbf{x} \in \mathbb{R}^8$

References Papamakarios et al. (2019b); Greenberg et al. (2019); Hermans et al. (2020)
Durkan et al. (2020)

T.4 SLCP with Distractors

This task is similar to T.3, with the difference that we add uninformative dimensions (distractors) to the observation:

Prior $\mathcal{U}(-\mathbf{3}, \mathbf{3})$

Simulator $\mathbf{x}|\boldsymbol{\theta} = (\mathbf{x}_1, \ldots, \mathbf{x}_{100})$, $\mathbf{x} = p(\mathbf{y})$, where p re-orders the dimensions of $\mathbf{y}$ with a fixed random permutation,
$\mathbf{y}_{[1:8]} \sim \mathcal{N}(\mathbf{m}_{\boldsymbol{\theta}}, \mathbf{S}_{\boldsymbol{\theta}})$, $\mathbf{y}_{[9:100]} \sim \frac{1}{20} \sum_{i=1}^{20} t_2(\boldsymbol{\mu}^i, \boldsymbol{\Sigma}^i)$
where $\mathbf{m}_{\boldsymbol{\theta}} = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}$, $\mathbf{S}_{\boldsymbol{\theta}} = \begin{bmatrix} s_1^2 & \rho s_1 s_2 \\ \rho s_1 s_2 & s_2^2 \end{bmatrix}$, $s_1 = \theta_3^2, s_2 = \theta_4^2, \rho = \tanh \theta_5$,
$\boldsymbol{\mu}^i \sim \mathcal{N}(0, 15^2 \mathbf{I})$, $\boldsymbol{\Sigma}^i_{j,k} \sim \mathcal{N}(0, 9)$, for $j > k$, $\boldsymbol{\Sigma}^i_{j,j} = 3e^a$, where $a \sim \mathcal{N}(0, 1)$, $\boldsymbol{\Sigma}^i_{j,k} = 0$ otherwise

Dimensionality $\boldsymbol{\theta} \in \mathbb{R}^5, \mathbf{x} \in \mathbb{R}^{100}$

References Greenberg et al. (2019)

T.5 Bernoulli GLM

Inference of a 10-parameter Generalized linear model (GLM) with Bernoulli observations, and Gaussian prior with covariance matrix which encourages smoothness by penalizing the second-order differences in the vector of parameters (De Nicolao et al., 1997). The observations are the sufficient statistics for this GLM:

Prior	$\beta \sim \mathcal{N}(0, 2)$, $\mathbf{f} \sim \mathcal{N}(\mathbf{0}, (\mathbf{F}^\top \mathbf{F})^{-1})$,
	$\mathbf{F}_{i,i-2} = 1$, $\mathbf{F}_{i,i-1} = -2$, $\mathbf{F}_{i,i} = 1 + \sqrt{\frac{i-1}{9}}$, $\mathbf{F}_{i,j} = 0$ otherwise, $1 \leq i, j \leq 9$
Simulator	$\mathbf{x}\|\boldsymbol{\theta} = (\mathbf{x}_1, \ldots, \mathbf{x}_{10})$, $\mathbf{x}_1 = \sum_i^T z_i$, $\mathbf{x}_{2:10} = \frac{1}{\mathbf{x}_1}\mathbf{V}\mathbf{z}$,
	$z_i \sim \text{Bern}(\eta(\mathbf{v}_i^\top \mathbf{f} + \beta))$, $\eta(\cdot) = \exp(\cdot)/(1 + \exp(\cdot))$,
	frozen input between time bins $i - 8$ and i: $\mathbf{v}_i \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, $\mathbf{V} = [v_1, v_2, \ldots, v_T]$
Dimensionality	$\boldsymbol{\theta} \in \mathbb{R}^{10}, \mathbf{x} \in \mathbb{R}^{10}$
Fixed parameters	Duration of task $T = 100$.
References	Lueckmann et al. (2017); Gonçalves et al. (2020)

T.6 Bernoulli GLM Raw

This task is similar to T.5, the sole difference being that the observations are not the sufficient statistics for the Bernoulli GLM process but the raw observations:

Prior	$\beta \sim \mathcal{N}(0, 2)$, $\mathbf{f} \sim \mathcal{N}(\mathbf{0}, (\mathbf{F}^\top \mathbf{F})^{-1})$,
	$\mathbf{F}_{i,i-2} = 1$, $\mathbf{F}_{i,i-1} = -2$, $\mathbf{F}_{i,i} = 1 + \sqrt{\frac{i-1}{9}}$, $\mathbf{F}_{i,j} = 0$ otherwise $1 \leq i, j \leq 9$
Simulator	$\mathbf{x}\|\boldsymbol{\theta} = (\mathbf{x}_1, \ldots, \mathbf{x}_{100})$, $x_i \sim \text{Bern}(\eta(\mathbf{v}_i^\top \mathbf{f} + \beta))$, $\eta(\cdot) = \exp(\cdot)/(1 + \exp(\cdot))$
	frozen input between time bins $i - 8$ and i: $\mathbf{v}_i \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$,
Dimensionality	$\boldsymbol{\theta} \in \mathbb{R}^{10}, \mathbf{x} \in \mathbb{R}^{100}$
Fixed parameters	Duration of task $T = 100$.

T.7 Gaussian Mixture

This task is common in the ABC literature. It consists of inferring the common mean of a mixture of two two-dimensional Gaussian distributions, one with much broader covariance than the other:

Prior	$\mathcal{U}(-\mathbf{10}, \mathbf{10})$
Simulator	$\mathbf{x}\|\boldsymbol{\theta} \sim 0.5\,\mathcal{N}(\mathbf{x}\|\mathbf{m}_{\boldsymbol{\theta}} = \boldsymbol{\theta}, \mathbf{S} = \mathbf{I}) + 0.5\,\mathcal{N}(\mathbf{x}\|\mathbf{m}_{\boldsymbol{\theta}} = \boldsymbol{\theta}, \mathbf{S} = 0.01 \odot \mathbf{I})$
Dimensionality	$\boldsymbol{\theta} \in \mathbb{R}^2, \mathbf{x} \in \mathbb{R}^2$
References	Sisson et al. (2007); Beaumont et al. (2009); Toni et al. (2009); Simola et al. (2020)

T.8 Two Moons

A two-dimensional task with a posterior that exhibits both global (bimodality) and local (crescent shape) structure to illustrate how algorithms deal with multimodality:

Prior $\mathcal{U}(-\mathbf{1}, \mathbf{1})$

Simulator $\boldsymbol{x}|\boldsymbol{\theta} = \begin{bmatrix} r\cos(\alpha) + 0.25 \\ r\sin(\alpha) \end{bmatrix} + \begin{bmatrix} -|\theta_1 + \theta_2|/\sqrt{2} \\ (-\theta_1 + \theta_2)/\sqrt{2} \end{bmatrix}$, where $\alpha \sim \mathcal{U}(-\pi/2, \pi/2)$, $r \sim \mathcal{N}(0.1, 0.01^2)$

Dimensionality $\boldsymbol{\theta} \in \mathbb{R}^2, \mathbf{x} \in \mathbb{R}^2$

References Greenberg et al. (2019)

T.9 SIR

The SIR model is an epidemiological model describing the dynamics of the number of individuals in three possible states: susceptible S, infectious I, and recovered or deceased R.

The SIR task consists in inferring the contact rate β and the mean recovery rate γ, given a sampled number of individuals in the infectious group I in 10 evenly-spaced points in time:

Prior $\beta \sim \text{LogNormal}(\log(0.4), 0.5)$ $\gamma \sim \text{LogNormal}(\log(1/8), 0.2)$

Simulator $\mathbf{x}|\boldsymbol{\theta} = (x_1, \ldots, x_{10})$, $x_i \sim \mathcal{B}(1000, \frac{I}{N})$, where I is simulated from

$\frac{dS}{dt} = -\beta\frac{SI}{N}$

$\frac{dI}{dt} = \beta\frac{SI}{N} - \gamma I$

$\frac{dR}{dt} = \gamma I$

Dimensionality $\boldsymbol{\theta} \in \mathbb{R}^2, \mathbf{x} \in \mathbb{R}^{10}$

Fixed parameters Population size $N = 1000000$ and duration of task $T = 160$.

Initial conditions: $(S(0), I(0), R(0)) = (N - 1, 1, 0)$

References Kermack and McKendrick (1927)

T.10 Lotka-Volterra

This is an influential model in ecology describing the dynamics of two interacting species, most commonly prey and predator interactions. Our task consists in the inference of four parameters $\boldsymbol{\theta}$ related to species interaction, given 20 summary statistics consisting of the number of individuals in both populations in 10 evenly-spaced points in time:

Prior $\alpha \sim \text{LogNormal}(-0.125, 0.5)$, $\beta \sim \text{LogNormal}(-3, 0.5)$,

$\gamma \sim \text{LogNormal}(-0.125, 0.5)$, $\delta \sim \text{LogNormal}(-3, 0.5)$

Simulator $\mathbf{x}|\boldsymbol{\theta} = (\mathbf{x}_1, \ldots, \mathbf{x}_{10})$, $\mathbf{x}_{1,i} \sim \text{LogNormal}(\log(X), 0.1)$, $\mathbf{x}_{2,i} \sim \text{LogNormal}(\log(Y), 0.1)$,

X and Y are simulated from

$\frac{dX}{dt} = \alpha X - \beta XY$

$\frac{dY}{dt} = -\gamma Y + \delta XY$

Dimensionality $\boldsymbol{\theta} \in \mathbb{R}^4, \mathbf{x} \in \mathbb{R}^{20}$

Fixed parameters Duration of task $T = 20$. Initial conditions: $(X(0), Y(0)) = (30, 1)$

References Lotka (1920)

W Website

The companion website (sbi-benchmark.github.io) allows interactive comparisons in terms of all metrics. It also allows inspection of posterior samples of all runs, which we found insightful when choosing hyperparameters and diagnosing implementation issues. Two screenshots are provided in Fig. 13.

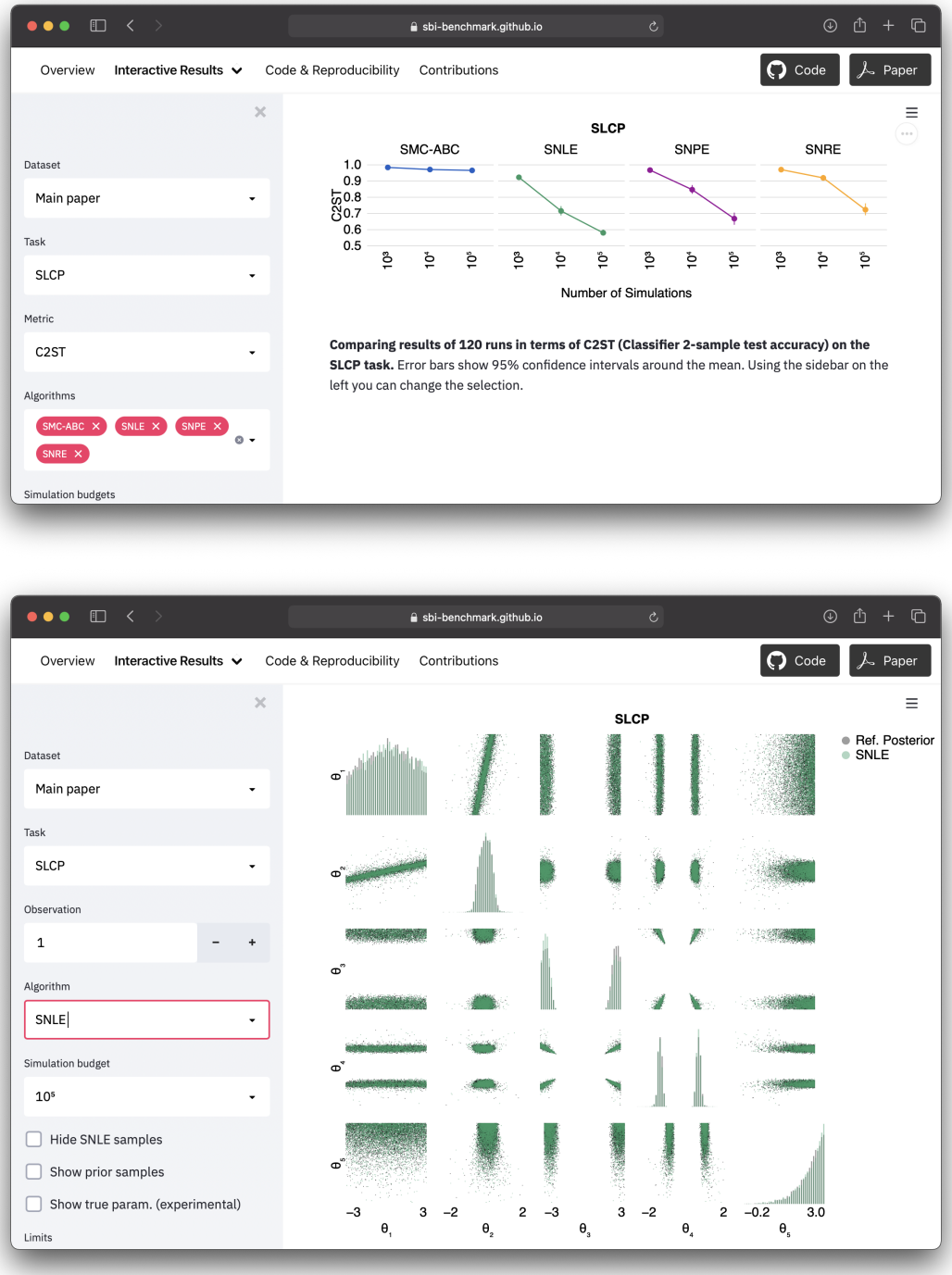

Figure 13: **Screenshots from the companion website**. Top: Classification accuracy (C2ST) for a subset of sequential algorithms on the SLCP task. Bottom: SNLE posterior on SLCP for $\mathbf{x}_o^{(1)}$ at 100k simulations.

References

Alsing, J.
2019. pydelfi: Density estimation likelihood-free inference. `https://github.com/justinalsing/pydelfi`.

Beaumont, M. A., J.-M. Cornuet, J.-M. Marin, and C. P. Robert
2009. Adaptive approximate bayesian computation. *Biometrika*, 96(4):983–990.

Beaumont, M. A., W. Zhang, and D. J. Balding
2002. Approximate bayesian computation in population genetics. *Genetics*, 162(4):2025–2035.

Bezanson, J., A. Edelman, S. Karpinski, and V. B. Shah
2017. Julia: A fresh approach to numerical computing. *SIAM review*, 59(1):65–98.

Bingham, E., J. P. Chen, M. Jankowiak, F. Obermeyer, N. Pradhan, T. Karaletsos, R. Singh, P. Szerlip, P. Horsfall, and N. D. Goodman
2019. Pyro: Deep universal probabilistic programming. *Journal of Machine Learning Research*, 20(1):973–978.

Blum, M. G.
2018. Regression approaches for abc. In *Handbook of Approximate Bayesian Computation*, chapter 3. CRC Press, Taylor & Francis Group.

Blum, M. G. and O. François
2010. Non-linear regression models for approximate bayesian computation. *Statistics and Computing*, 20(1):63–73.

Breiman, L.
2001. Random forests. *Machine Learning*, 45(1):5–32.

Carpenter, B., A. Gelman, M. Hoffman, D. Lee, B. Goodrich, M. Betancourt, M. Brubaker, J. Guo, P. Li, and A. Riddell
2017. Stan: A probabilistic programming language. *Journal of Statistical Software, Articles*, 76(1):1–32.

Chwialkowski, K., H. Strathmann, and A. Gretton
2016. A kernel test of goodness of fit. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, Pp. 2606–2615. PMLR.

Collin, F.-D., A. Estoup, J.-M. Marin, and L. Raynal
2020. Bringing abc inference to the machine learning realm: Abcranger, an optimized random forests library for abc. In *JOBIM 2020*, volume 2020.

Cranmer, K., J. Pavez, and G. Louppe
2015. Approximating likelihood ratios with calibrated discriminative classifiers. *arXiv preprint arXiv:1506.02169*.

Dalmasso, N., A. B. Lee, R. Izbicki, T. Pospisil, and C.-A. Lin
2020. Validation of approximate likelihood and emulator models for computationally intensive simulations. In *Proceedings of The 23rd International Conference on Artificial Intelligence and Statistics (AISTATS)*.

De Nicolao, G., G. Sparacino, and C. Cobelli
1997. Nonparametric input estimation in physiological systems: problems, methods, and case studies. *Automatica*, 33(5).

Drovandi, C. C., C. Grazian, K. Mengersen, and C. Robert
2018. Approximating the likelihood in approximate bayesian computation. In *Handbook of Approximate Bayesian Computation*, S. Sisson, Y. Fan, and M. Beaumont, eds., chapter 12. CRC Press, Taylor & Francis Group.

Durkan, C., A. Bekasov, I. Murray, and G. Papamakarios
2019. Neural spline flows. In *Advances in Neural Information Processing Systems*, Pp. 7509–7520. Curran Associates, Inc.

Durkan, C., I. Murray, and G. Papamakarios
2020. On contrastive learning for likelihood-free inference. In *Proceedings of the 36th International Conference on Machine Learning*, volume 98 of *Proceedings of Machine Learning Research*. PMLR.

Durkan, C., G. Papamakarios, and I. Murray
2018. Sequential neural methods for likelihood-free inference. *Bayesian Deep Learning Workshop at Neural Information Processing Systems*.

Dutta, R., J. Corander, S. Kaski, and M. U. Gutmann
2016. Likelihood-free inference by ratio estimation. *arXiv preprint arXiv:1611.10242*.

Dutta, R., M. Schoengens, J.-P. Onnela, and A. Mira
2017. Abcpy: A user-friendly, extensible, and parallel library for approximate bayesian computation. In *Proceedings of the Platform for Advanced Scientific Computing Conference*, PASC '17.

Gelman, A., J. B. Carlin, H. S. Stern, and D. B. Rubin
2004. *Bayesian Data Analysis*, 2nd ed. edition. Chapman and Hall/CRC.

Gonçalves, P. J., J.-M. Lueckmann, M. Deistler, M. Nonnenmacher, K. Öcal, G. Bassetto, C. Chintaluri, W. F. Podlaski, S. A. Haddad, T. P. Vogels, D. S. Greenberg, and J. H. Macke
2020. Training deep neural density estimators to identify mechanistic models of neural dynamics. *eLife*.

Greenberg, D., M. Nonnenmacher, and J. Macke
2019. Automatic posterior transformation for likelihood-free inference. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, Pp. 2404–2414. PMLR.

Gretton, A., K. M. Borgwardt, M. J. Rasch, B. Schölkopf, and A. Smola
2012. A kernel two-sample test. *The Journal of Machine Learning Research*, 13(Mar):723–773.

Gutmann, M. U. and J. Corander
2016. Bayesian optimization for likelihood-free inference of simulator-based statistical models. *The Journal of Machine Learning Research*, 17(1):4256–4302.

Gutmann, M. U., R. Dutta, S. Kaski, and J. Corander
2018. Likelihood-free inference via classification. *Statistics and Computing*, 28(2):411–425.

Hermans, J.
2019. Hypothesis. `https://github.com/montefiore-ai/hypothesis`.

Hermans, J., V. Begy, and G. Louppe
2020. Likelihood-free mcmc with approximate likelihood ratios. In *Proceedings of the 37th International Conference on Machine Learning*, volume 98 of *Proceedings of Machine Learning Research*. PMLR.

Hogg, D. W. and D. Foreman-Mackey
2018. Data analysis recipes: Using markov chain monte carlo. *The Astrophysical Journal Supplement Series*, 236(1):11.

Ishida, E., S. Vitenti, M. Penna-Lima, J. Cisewski, R. de Souza, A. Trindade, E. Cameron, V. Busti, C. collaboration, et al.
2015. Cosmoabc: likelihood-free inference via population monte carlo approximate bayesian computation. *Astronomy and Computing*, 13:1–11.

Izbicki, R., A. Lee, and C. Schafer
2014. High-dimensional density ratio estimation with extensions to approximate likelihood computation. In *Artificial Intelligence and Statistics*, Pp. 420–429.

Jennings, E. and M. Madigan
2017. astroabc: an approximate bayesian computation sequential monte carlo sampler for cosmological parameter estimation. *Astronomy and computing*, 19:16–22.

Kermack, W. O. and A. G. McKendrick
1927. A contribution to the mathematical theory of epidemics. *Proceedings of the royal society of London. Series A, Containing papers of a mathematical and physical character*, 115(772):700–721.

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

Klinger, E., D. Rickert, and J. Hasenauer
2018. pyabc: distributed, likelihood-free inference. *Bioinformatics*, 34(20):3591–3593.

Kousathanas, A., P. Duchen, and D. Wegmann
2018. A guide to general-purpose abc software. In *Handbook of Approximate Bayesian Computation*, chapter 13. CRC Press, Taylor & Francis Group.

Lintusaari, J., H. Vuollekoski, A. Kangasrääsiö, K. Skytén, M. Järvenpää, P. Marttinen, M. U. Gutmann, A. Vehtari, J. Corander, and S. Kaski
2018. Elfi: engine for likelihood-free inference. *The Journal of Machine Learning Research*, 19(1):643–649.

Liu, F., W. Xu, J. Lu, G. Zhang, A. Gretton, and D. J. Sutherland
2020. Learning deep kernels for non-parametric two-sample tests. In *Proceedings of the 37th International Conference on Machine Learning*, volume 98 of *Proceedings of Machine Learning Research*. PMLR.

Liu, Q., J. Lee, and M. Jordan
2016. A kernelized stein discrepancy for goodness-of-fit tests. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, Pp. 276–284. PMLR.

Lopez-Paz, D. and M. Oquab
2017. Revisiting classifier two-sample tests. In *5th International Conference on Learning Representations, ICLR*.

Lotka, A. J.
1920. Analytical note on certain rhythmic relations in organic systems. *Proceedings of the National Academy of Sciences*, 6(7):410–415.

Louppe, G., K. Cranmer, and J. Pavez
2016. carl: a likelihood-free inference toolbox. *Journal of Open Source Software*, 1(1):11.

Lueckmann, J.-M., G. Bassetto, T. Karaletsos, and J. H. Macke
2019. Likelihood-free inference with emulator networks. In *Proceedings of The 1st Symposium on Advances in Approximate Bayesian Inference*, volume 96 of *Proceedings of Machine Learning Research*, Pp. 32–53. PMLR.

Lueckmann, J.-M., P. J. Goncalves, G. Bassetto, K. Öcal, M. Nonnenmacher, and J. H. Macke
2017. Flexible statistical inference for mechanistic models of neural dynamics. In *Advances in Neural Information Processing Systems 30*, Pp. 1289–1299. Curran Associates, Inc.

Marjoram, P. and S. Tavaré
2006. Modern computational approaches for analysing molecular genetic variation data. *Nature Reviews Genetics*, 7(10):759–770.

Martino, L., D. Luengo, and J. Míguez
2018. Accept–reject methods. In *Independent Random Sampling Methods*, Pp. 65–113. Springer.

Neal, R. M.
2003. Slice sampling. *Annals of Statistics*, Pp. 705–741.

Papamakarios, G. and I. Murray
2016. Fast ϵ-free inference of simulation models with bayesian conditional density estimation. In *Advances in Neural Information Processing Systems 29*, Pp. 1028–1036. Curran Associates, Inc.

Papamakarios, G., E. Nalisnick, D. J. Rezende, S. Mohamed, and B. Lakshminarayanan
2019a. Normalizing flows for probabilistic modeling and inference. *arXiv preprint arXiv:1912.02762*.

Papamakarios, G., T. Pavlakou, and I. Murray
2017. Masked autoregressive flow for density estimation. In *Advances in Neural Information Processing Systems 30*, Pp. 2338–2347. Curran Associates, Inc.

Papamakarios, G., D. Sterratt, and I. Murray
2019b. 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*, Pp. 837–848. PMLR.

Paszke, A., S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala
2019. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, Pp. 8024–8035. Curran Associates, Inc.

Pham, K. C., D. J. Nott, and S. Chaudhuri
2014. A note on approximating abc-mcmc using flexible classifiers. *Stat*, 3(1):218–227.

Prangle, D., M. G. B. Blum, G. Popovic, and S. A. Sisson
2014a. Diagnostic tools of approximate bayesian computation using the coverage property. *Australian & New Zealand Journal of Statistics*, 56(4):309–329.

Prangle, D., P. Fearnhead, M. P. Cox, P. J. Biggs, and N. P. French
2014b. Semi-automatic selection of summary statistics for abc model choice. *Statistical applications in genetics and molecular biology*, 13(1):67–82.

Priddle, J. W., S. A. Sisson, and C. Drovandi
2019. Efficient bayesian synthetic likelihood with whitening transformations. *arXiv preprint arXiv:1909.04857*.

Pritchard, J. K., M. T. Seielstad, A. Perez-Lezaun, and M. W. Feldman
1999. Population growth of human y chromosomes: a study of y chromosome microsatellites. *Molecular Biology and Evolution*, 16(12):1791–1798.

Pudlo, P., J.-M. Marin, A. Estoup, J.-M. Cornuet, M. Gautier, and C. P. Robert
2016. Reliable abc model choice via random forests. *Bioinformatics*, 32(6):859–866.

pyABC API Documentation
2020. Multivariate normal transition. `https://pyabc.readthedocs.io/en/latest/api_transition.html#pyabc.transition.MultivariateNormalTransition`. Accessed: 2020-10-20.

Rackauckas, C. and Q. Nie
2017. Differentialequations.jl – a performant and feature-rich ecosystem for solving differential equations in julia. *The Journal of Open Research Software*, 5(1).

Ramdas, A., S. J. Reddi, B. Poczos, A. Singh, and L. Wasserman
2015. On the decreasing power of kernel and distance based nonparametric hypothesis tests in high dimensions. *AAAI Conference on Artificial Intelligence*.

Raynal, L., J.-M. Marin, P. Pudlo, M. Ribatet, C. P. Robert, and A. Estoup
2019. Abc random forests for bayesian parameter inference. *Bioinformatics*, 35(10):1720–1728.

Rubin, D. B.
1988. Using the sir algorithm to simulate posterior distributions. *Bayesian statistics*, 3:395–402.

Simola, U., J. Cisewski-Kehe, M. U. Gutmann, J. Corander, et al.
2020. Adaptive approximate bayesian computation tolerance selection. *Bayesian Analysis*.

Sisson, S. A., Y. Fan, and M. M. Tanaka
2007. Sequential monte carlo without likelihoods. *Proceedings of the National Academy of Sciences*, 104(6):1760–1765.

Talts, S., M. Betancourt, D. Simpson, A. Vehtari, and A. Gelman
2018. Validating bayesian inference algorithms with simulation-based calibration. *arXiv preprint arXiv:1804.06788*.

Tavaré, S., D. J. Balding, R. C. Griffiths, and P. Donnelly
1997. Inferring coalescence times from dna sequence data. *Genetics*, 145(2).

Tejero-Cantero, A., J. Boelts, M. Deistler, J.-M. Lueckmann, C. Durkan, P. J. Gonçalves, D. S. Greenberg, and J. H. Macke
2020. sbi: A toolkit for simulation-based inference. *Journal of Open Source Software*, 5(52):2505.

Thomas, O., R. Dutta, J. Corander, S. Kaski, and M. U. Gutmann
2020. Likelihood-free inference by ratio estimation. *Bayesian Analysis*.

Toni, T., D. Welch, N. Strelkowa, A. Ipsen, and M. P. Stumpf
2009. Approximate bayesian computation scheme for parameter inference and model selection in dynamical systems. *Journal of the Royal Society Interface*, 6(31):187–202.

Wood, S. N.
2010. Statistical inference for noisy nonlinear ecological dynamic systems. *Nature*, 466(7310):1102–1104.