Fast and Robust Simulation-Based Inference With Optimization Monte Carlo

Vasilis Gkolemis
Harokopio University of Athens

Christos Diou
Harokopio University of Athens

Michael Gutmann
University of Edinburgh

Abstract

Bayesian parameter inference for complex stochastic simulators is challenging due to intractable likelihood functions. Existing simulation-based inference methods often require large number of simulations and become costly to use in high-dimensional parameter spaces or in problems with partially uninformative outputs. We propose a new method for differentiable simulators that delivers accurate posterior inference with substantially reduced runtimes. Building on the Optimization Monte Carlo framework, our approach reformulates stochastic simulation as deterministic optimization problems. Gradient-based methods are then applied to efficiently navigate toward high-density posterior regions and avoid wasteful simulations in low-probability areas. A JAX-based implementation further enhances the performance through vectorization of key method components. Extensive experiments, including high-dimensional parameter spaces, uninformative outputs, multiple observations and multimodal posteriors show that our method consistently matches, and often exceeds, the accuracy of state-of-the-art approaches, while reducing the runtime by a substantial margin.

1 Introduction

Parametric stochastic simulators are widely used to model complex processes across biology, physics, health sciences, and many other disciplines. As simulators become more realistic, they also become more complex, often involving many free parameters and high-dimensional outputs. This complexity, in turn, poses significant challenges for parameter inference.

A variety of methods, collectively referred to as Likelihood-Free (LFI) or Simulation-Based (SBI) Inference (Lintusaari et al., 2017; Sisson et al., 2018; Cranmer et al., 2020; Deistler et al., 2025), have been developed to infer simulator parameters. These include sampling-based techniques, like Approximate Bayesian Computation (ABC, Marin et al., 2012), and approaches based on summary statistics (Chen et al., 2021, 2023), parametric modeling (Wood, 2010; Price et al., 2018), and ratio estimation (Thomas et al., 2022; Durkan et al., 2020). More recently, substantial progress has been driven by neural methods, which use deep networks to estimate the posterior (Greenberg et al., 2019; Papamakarios and Murray, 2016; Radev et al., 2020; Wildberger et al., 2023) or the likelihood (Papamakarios et al., 2019) function.

Neural-based approaches have greatly broadened the applicability of SBI to complex problems. However, they are data-intensive, as they require large volumes of simulated datasets to train their neural estimators. Since both data generationand neural network training are computationally costly, these methods often incur significant runtimes, which in turn restricts experimental flexibility and hinders practical deployment.

The computational cost of SBI methods is particularly pronounced in two scenarios. First, in high-dimensional parameter spaces, they face the curse of dimensionality, as they rely on densely populating both parameter and data spaces with samples (Lintusaari et al., 2017; Sisson et al., 2018). Second, when a subset of the output dimensions depends weakly, or not at all, on the simulator parameters (Saltelli et al., 2008; Monsalve-Bravo et al., 2022), these uninformative outputs act as "distractors", complicating inference (Lueckmann et al., 2021). As illustrated in Figure 1, in both cases existing neural-based approaches demand large simulation budgets, resulting in substantial runtime for an accurate posterior estimate.

We propose R2OMC, a novel LFI method that overcomes these challenges (see Figure 1). Building on the Robust Optimization Monte Carlo framework (ROMC, Ikonomov and Gutmann, 2020), it reformulates the inference problem in terms of deterministic optimization problems and uses gradient descent to obtain posterior

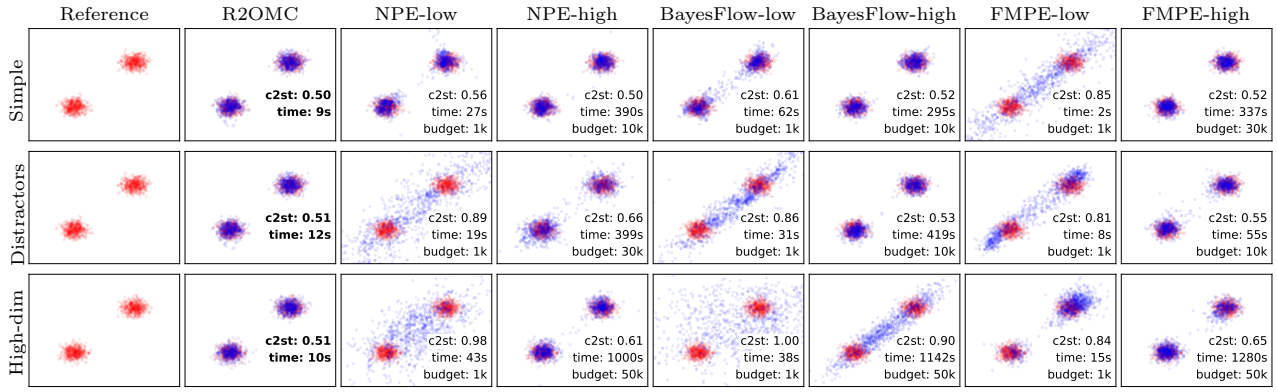


Figure 1: Posterior inference on a two-Gaussian mixture simulator. Rows correspond to simple ($D = D_y = 2$), distractor ($D = 2, D_y = 18$), and high-dimensional ($D = D_y = 10$) settings. Columns show the reference posterior, and samples obtained from R2OMC (proposed), and three established neural-based methods—Neural Posterior Estimator (NPE, Greenberg et al., 2019), BayesFlow (Radev et al., 2020), and Flow Matching Posterior Estimator (FMPE, Wildberger et al., 2023)—each evaluated at low and high runtime. Unlike the neural methods, which require high runtime in the distractor and high-dimensional settings, R2OMC produces accurate posterior samples with low runtime. See Appendix C for details on the experimental setup.

samples. Gradients, as in standard optimization, provide efficient navigation in high-dimensional parameter spaces, so they rapidly guide us toward high-density posterior regions. Moreover, they enable sensitivity analyses which allows the identification and filtering of distractor dimensions, improving inference quality.

ROMC, the foundation of our method, establishes the theory for casting inference as deterministic optimization but has key limitations: it lacks a strategy for defining an appropriate distance function in problems with distractors or multiple iid observations and its latest implementation (Gkolemis et al., 2023) does not exploit gradients or other computational accelerations, restricting its applicability to low-dimensional problems. R2OMC overcomes these limitations, providing an SBI method for differentiable simulators that:

- enables posterior sampling in high-dimensional parameter spaces with efficient runtimes,
- automatically identifies and filters out uninformative dimensions (distractors),
- handles multiple iid observations effectively,
- runs efficiently through a `JAX` implementation that uses automatic differentiation, vectorization, and just-in-time compilation.

R2OMC is systematically compared to state-of-the-art SBI methods, demonstrating accurate posterior inference with substantially lower runtime across a wide range of scenarios.

Code is available at github.com/givasile/lfi.

2 Background and Motivation

Neural-based SBI methods are data-intensive, with their accuracy depending on access to large simulated datasets. This incurs computational costs on two fronts: generating massive datasets from complex simulators and training deep neural estimators on such volumes. Prior work (Lueckmann et al., 2021) has shown that it is the latter that often dominates runtime. Moreover, modern frameworks, like `JAX`, are used in SBI (Dirmeier et al., 2024) to accelerate simulation through vectorization, which can massively reduce the first cost but not the second. Appendix A expands on both points. This motivates us to develop a method that avoids the cost of training an external neural estimator.

2.1 ROMC framework

ROMC, introduced by Ikonomov and Gutmann (2020), improves upon Optimization Monte Carlo Meeds and Welling (2015) by addressing its robustness issues. It belongs to the larger family of ABC methods that approximate the likelihood function as the probability of simulating an output ϵ-close to the observation,

$$L_\epsilon(\boldsymbol{\theta}) = \Pr(d(\mathbf{y}, \mathbf{y}^o) \leq \epsilon \mid \boldsymbol{\theta}). \tag{1}$$

Here, $d(\cdot, \cdot)$ is a distance, $\mathbf{y}$ the simulator output, and $\mathbf{y}^o$ the observed data. In the following, we denote by $g(\boldsymbol{\theta}, \mathbf{u})$ the simulator that maps noise (nuisance) variables $\mathbf{u} \sim p(\mathbf{u})$ and parameters $\boldsymbol{\theta}$ to multivariate data $\mathbf{y} = g(\boldsymbol{\theta}, \mathbf{u})$. In a computer program, sampling $\mathbf{u}$ corresponds to sampling a random seed. Introducing the acceptance region $C_\epsilon = \{(\boldsymbol{\theta}, \mathbf{u}) : d(g(\boldsymbol{\theta}, \mathbf{u}), \mathbf{y}^o) \leq \epsilon\}$

and using the law of the unconscious statistician, (1) can be written as

$$L_\epsilon(\boldsymbol{\theta}) = \mathbb{E}_{p(\mathbf{u})}\left[\mathbb{1}_{C_\epsilon}(\boldsymbol{\theta}, \mathbf{u})\right], \tag{2}$$

where $\mathbb{1}_{C_\epsilon}$ denotes the indicator function that is one if $(\boldsymbol{\theta}, \mathbf{u}) \in C_\epsilon$ and zero otherwise. When approximating the expectation as a sample average, we obtain

$$L_\epsilon(\boldsymbol{\theta}) \approx \frac{1}{S}\sum_{i=1}^{S} \mathbb{1}_{C_\epsilon^i}(\boldsymbol{\theta}), \tag{3}$$

which is the proportion of acceptance regions $C_\epsilon^i = \{\boldsymbol{\theta} : d(g(\boldsymbol{\theta}, \mathbf{u}_i), \mathbf{y}^o) \leq \epsilon\}$ that contain $\boldsymbol{\theta}$.

ROMC takes (3) as starting point and focuses on characterizing the acceptance regions C_ϵ^i, because once the C_ϵ^i are known, we can not only evaluate the approximate likelihood function, but also obtain posterior samples efficiently. ROMC introduces uniform proposal distributions q_i with support on C_ϵ^i only. Samples from $\boldsymbol{\theta}_{ij} \sim q_i$ for $j \in 1, \ldots, M$, are then equal to posterior samples if weighted with the (unnormalized) weights w_{ij}, $i = 1, \ldots, S, j = 1, \ldots, M$,

$$w_{ij} = \frac{p(\boldsymbol{\theta}_{ij})}{q_i(\boldsymbol{\theta}_{ij})}\mathbb{1}_{C_\epsilon^i}(\boldsymbol{\theta}_{ij}) \tag{4}$$

where $p(\boldsymbol{\theta})$ is the prior. For the characterization of the acceptance regions C_ϵ^i, ROMC first determines the minima $\boldsymbol{\theta}_i^*$,

$$\boldsymbol{\theta}_i^* = \text{argmin}_{\boldsymbol{\theta}} d_i(\boldsymbol{\theta}), \quad d_i(\boldsymbol{\theta}) = d(g(\boldsymbol{\theta}, \mathbf{u}_i), \mathbf{y}^o), \tag{5}$$

where $d_i(\boldsymbol{\theta})$ are deterministic distance functions because the sources of randomness (seeds) are set to $\mathbf{u}_i$ and kept fixed. Next, it starts from the optimization end points $\boldsymbol{\theta}_i^*$ to discover C_ϵ^i selecting ϵ as the value derived from the quantiles of the minimum distance functions, $d_i^* = d_i(\boldsymbol{\theta}_i^*), i = 1, \ldots, S$ (see Ikonomov and Gutmann, 2020, for details). While several options are possible (Ikonomov and Gutmann, 2020), here we use hyperboxes centered at $\boldsymbol{\theta}_i^*$, which can be constructed with a line search algorithm for each parameter dimension. For details, please see Appendix B.

The power of the ROMC framework stems from the possibility to use optimization, in particular gradient-based methods, to determine the optimization end points $\boldsymbol{\theta}_i^*$, and hence the acceptance regions C_ϵ^i and the proposal distributions q_i. However, ROMC has important limitations, which we analyze in the next subsection.

2.2 ROMC challenges and limitations

ROMC suffers from three main limitations; (a) it fails in the presence of distractors, (b) it does not provide a strategy for handling multiple iid observations, and (c) its existing implementation does not use gradients and other computational accelerations, so it can only be applied to low-dimensional problems.

Uninformative dimensions (distractors). Distractors are output dimensions that do not depend on the parameters of interest. For instance, consider a graphics renderer where the parameter of interest controls a scene element, e.g. the appearance of vase, so that most pixel values (output dimensions) are unrelated to the parameter and thus serve as distractors. Formally, consider an observation vector $\mathbf{y}^o = (\mathbf{y}^{o,(1)}, \mathbf{y}^{o,(2)})$, where only $\mathbf{y}^{o,(1)}$ depends on $\boldsymbol{\theta}$. Posterior inference depends solely on $\mathbf{y}^{o,(1)}$, so the elements of $\mathbf{y}^{o,(2)}$ act as distractor dimensions.

ROMC is vulnerable to failure in this setting because typical distance functions d consider all output dimensions equally. Therefore, the minimal distances d_i^* will generally be large for seeds that generate $\mathbf{y}^{(2)}$ significantly different from $\mathbf{y}^{o,(2)}$. This results in an inflated ϵ, and in turn to a much broader posterior than the ground truth.

Multiple iid observations. In many applications, we have access to multiple iid observations $\{\mathbf{y}_n^o\}_{n=1}^N$. For instance, in neuroscience, researchers often collect several recordings of the same neural response to a stimulus. Properly incorporating these repeated measurements is crucial for accurate inference, however, most simulation-based methods, including ROMC, are designed for single observations, making it non-trivial to aggregate data and estimate $p(\boldsymbol{\theta}; \{\mathbf{y}_n^o\}_{n=1}^N)$ correctly.

Naive approaches are prone to failure. For example, simulating N outputs with fixed seeds $\{\mathbf{u}_{i,n}\}$ and averaging pairwise distances, i.e., $d_i(\boldsymbol{\theta}) = 1/N \sum_n \tilde{d}(g(\boldsymbol{\theta}, \mathbf{u}_{i,n}), \mathbf{y}_n^o)$—or concatenating all observations into a single vector—are sensitive to the arbitrary ordering of data points, violating permutation invariance. Such artificial sensitivity inflate the minimal achievable distances d_i^*, forcing an excessively large ϵ and yielding posteriors that are much broader than the ground truth. Similarly, simple summary statistics, like computing the mean of each feature across observations, discard higher-order information and variability, producing biased or overconfident posteriors.

Computational efficiency. Current ROMC implementations (Gkolemis et al., 2023) suffer from significant computational bottlenecks that restrict their applicability to low-dimensional problems. First, optimization relies on gradient-free methods, e.g., Bayesian optimization, which scale poorly with the number of parameters. Second, the construction of the acceptance regions C_ϵ^i with a line-search algorithm (see Appendix B), requires repeated simulator evaluations per seed

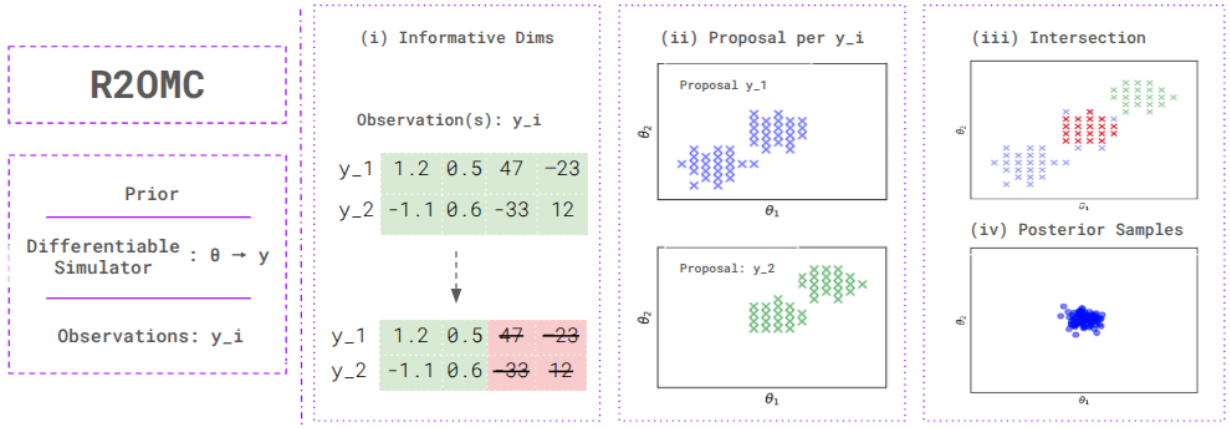


Figure 2: R2OMC overview: Distractors are automatically filtered out and proposal distributions q_i^n are created for each observation (here two): the blue-dotted region indicates where the proposal distributions for the first observation are nonzero, and the green-dotted region indicates the same for the second observation. Only samples that fall within the intersection of both regions are given a positive weight and become posterior samples.

and per dimension, which also scales poorly with the parameter dimensions.

3 Proposed method: R2OMC

We here present our new method for Bayesian parameter inference in simulator models. The method uses gradients to obtain posterior samples even in high-dimensional parameter spaces. The method belongs to the ROMC framework so that we start with discussing how we deal with distractors and iid observations before discussing its implementation.

3.1 Uninformative dimensions (distractors)

Distractors can negatively affect the quality of the posterior approximation. To counteract their influence, we measure the sensitivity of the output dimensions with respect to the parameters $\boldsymbol{\theta}$ by computing the average norm of their derivative with respect to $\boldsymbol{\theta}$, and checking that the value is above a minimal value (threshold) τ,

$$I_i = \mathbb{E}_{p(\boldsymbol{\theta})p(\mathbf{u})} \left[||\nabla_{\boldsymbol{\theta}} g_i(\boldsymbol{\theta}, \mathbf{u})|| \right] > \tau. \tag{6}$$

The expectation equals the expectation with respect to the joint distribution of the prior and the i-th dimension of the simulator output, $g_i(\boldsymbol{\theta}, \mathbf{u})$. In our experiments, we approximate the expectation using 50 samples from $p(\boldsymbol{\theta})$ and $p(\mathbf{u})$ each. Collecting all binary indicator variables I_i into the vector $\mathbf{m}_\tau$, we can filter out uninformative dimensions by computing the distance between observed and simulated data for informative dimensions only. To compute $\boldsymbol{\theta}^*_{i,n}$ in (11), we minimize

$$d_i^n(\boldsymbol{\theta}) = d(g(\boldsymbol{\theta}, \mathbf{u}_{i,n}) \odot \mathbf{m}_\tau, \mathbf{y}_n^o \odot \mathbf{m}_\tau). \tag{7}$$

The procedure is illustrated in Figure 2, step (i).

The threshold τ controls the sensitivity of the test. In our experiments, we set τ to machine precision; more sophisticated approaches consist in monitoring the distribution of the expected norm of the gradient, or by assessing false vs true positives under control conditions. This is possible since the mask $\mathbf{m}_\tau$ can be computed prior to receiving observed data.

3.2 Multiple iid observations

The likelihood for multiple iid observations $\{\mathbf{y}_n^o\}_{n=1}^N$ is the product of the likelihoods for each data point. Approximating the likelihood for each data point as in (3) and multiplying-in the prior $p(\boldsymbol{\theta})$, we can obtain a formal expression for the approximate posterior

$$p_\epsilon(\boldsymbol{\theta}|\{\mathbf{y}_n^o\}) \propto p(\boldsymbol{\theta}) \prod_n^N \sum_i^S \mathbb{1}_{C_\epsilon^{i,n}}(\boldsymbol{\theta}), \tag{8}$$

where $C_\epsilon^{i,n}$ is the acceptance region for data point n and source of randomness (seed) $\mathbf{u}_{i,n}$,

$$C_\epsilon^{i,n} = \{\boldsymbol{\theta} : d(g(\boldsymbol{\theta}, \mathbf{u}_{i,n}), \mathbf{y}_n^o) \leq \epsilon\}. \tag{9}$$

We have S sources of randomness for each data point $\mathbf{y}_n^o$, and so in total $S * N$ acceptance regions $C_\epsilon^{i,n}$.

We now address the question how to efficiently sample from $p_\epsilon(\boldsymbol{\theta}|\{\mathbf{y}_n^o\})$. To achieve this, we consider the task of computing the posterior expectation $\bar{h} = \mathbb{E}_{p_\epsilon}[h(\boldsymbol{\theta})]$ under $p_\epsilon(\boldsymbol{\theta}|\{\mathbf{y}_n^o\})$. Plugging in the expression in (8) gives, after normalization,

$$\bar{h} = \frac{\int h(\boldsymbol{\theta}) p(\boldsymbol{\theta}) \prod_n^N \sum_i^S \mathbb{1}_{C_\epsilon^{i,n}}(\boldsymbol{\theta}) d\boldsymbol{\theta}}{\int p(\boldsymbol{\theta}) \prod_n^N \sum_i^S \mathbb{1}_{C_\epsilon^{i,n}}(\boldsymbol{\theta}) d\boldsymbol{\theta}}. \tag{10}$$

The integrals in the equation are typically intractable. But they correspond to expectations with respect to the prior $p(\boldsymbol{\theta})$ and hence could be approximated with a sample average. However, this is very inefficient. Few (or no) prior samples will generate outputs ϵ-close to all observations, so most (or all) will get zero weight, leading to a very low effective sample size. We thus employ importance sampling with an adaptive proposal distribution obtained via optimization.

We first treat each data point $\mathbf{y}_n^o$ separately and construct proposal distributions $q_i^n(\boldsymbol{\theta})$, $i = 1, \dots, S$, as in Section 2.1: We use optimization to find the minimizers $\boldsymbol{\theta}_{i,n}^*$ of the deterministic functions $d_i^n(\boldsymbol{\theta}) = d(g(\boldsymbol{\theta}, \mathbf{u}_{i,n}), \mathbf{y}_n^o)$,

$$\boldsymbol{\theta}_{i,n}^* = \text{argmin}_{\boldsymbol{\theta}} d_i^n(\boldsymbol{\theta}), \tag{11}$$

and then build a hyperbox around each acceptance region $C_\epsilon^{i,n}$ and define $q_i^n(\boldsymbol{\theta})$ to be the uniform distribution over it. The optimization problem in (11) is deterministic and we solve it with gradient-based methods to enable scalability to large parameter spaces. Note that treating each data point $\mathbf{y}_n^o$ separately allows us to avoid the issues explained in Section 2.2.

We propose to combine the per-data point proposal distributions $q_i^n(\boldsymbol{\theta})$ in form of a mixture distribution,

$$q(\boldsymbol{\theta}) = \frac{1}{N}\frac{1}{S}\sum_{n=1}^{N}\sum_{i=1}^{S} q_i^n(\boldsymbol{\theta}). \tag{12}$$

The posterior expectation in (10) can then be approximated in form of a weighted average

$$\bar{h} = \frac{\sum_p w_p h(\boldsymbol{\theta}_p)}{\sum_p w_p}, \quad \boldsymbol{\theta}_p \sim q(\boldsymbol{\theta}) \tag{13}$$

where the weights w_p, $p = 1, \dots, P$, are given by

$$w_p = \frac{p(\boldsymbol{\theta}_p)}{q(\boldsymbol{\theta}_p)} \prod_{n=1}^{N}\sum_{i=1}^{S} \mathbb{1}_{C_\epsilon^{i,n}}(\boldsymbol{\theta}_p). \tag{14}$$

This means that the $\boldsymbol{\theta}_p \sim q(\boldsymbol{\theta})$, weighted by w_p, represent samples from the posterior $p_\epsilon(\boldsymbol{\theta}|\{\mathbf{y}_n^o\})$ in (8).

The rationale for choosing $q(\boldsymbol{\theta})$ in (12) as proposal distribution is as follows: it is a mixture of uniform distributions, so sampling and evaluating is easy. It seamlessly combines the contribution from multiple data points, allowing future data points to be included without re-computation. If each q_i^n encloses the corresponding acceptance region $C_\epsilon^{i,n}$, then $q(\boldsymbol{\theta}) > 0$ in areas of the parameter space where $\prod_n^N \sum_i^S \mathbb{1}_{C_\epsilon^{i,n}}(\boldsymbol{\theta}) > 0$ and no posterior mass is discarded in the weighting in (14). Finally, regions of high likelihood are characterized by regions where multiple $C_\epsilon^{i,n}$ overlap. As the proposal distribution $q(\boldsymbol{\theta})$ consists of a mixture of the $q_i^n(\boldsymbol{\theta})$ with equal weights, areas where multiple $C_\epsilon^{i,n}$ overlap will be sampled more often, so that areas of high likelihood will be represented with more samples.

Figure 2, steps (ii-iii), illustrates the proposed procedure to obtain posterior samples. We have two observed data points and the $\boldsymbol{\theta}_p$ come from regions that match at least one observation; the blue-dotted area for $\mathbf{y}_1^o$ and the green-dotted region for $\mathbf{y}_2^o$. However, only the ones that match both will get a positive weight (red dots) and become posterior samples.

3.3 `JAX` implementation

Algorithms 1 summarizes the key steps of R2OMC: dealing with distractors (line 2); constructing per-data point proposal distributions $\{q_i^n\}_{i=1}^S$ (lines 3-6); combining them to form the overall proposal distribution $q(\boldsymbol{\theta})$ (line 7) and generating weighted samples (line 8).

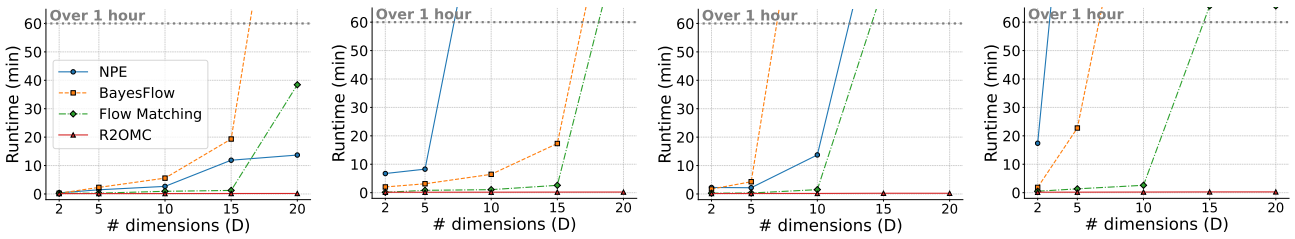
Algorithm 1 `R2OMC`. Requires: $p(\boldsymbol{\theta}), g(\boldsymbol{\theta}, \mathbf{u}), \{\mathbf{y}_n^o\}_{n=1}^N$

1: **procedure** R2OMC(τ, S, P)
2: $\quad\mathbf{m}_\tau = (I_1, \dots, I_{D_y})$ using (6) with τ
3: $\quad$**for** $n \leftarrow 1$ to N **do**
4: $\quad\quad\boldsymbol{\theta}_{i,n}^* = \text{argmin}_{\boldsymbol{\theta}} d_i^n(\boldsymbol{\theta})$ using (7) for all i
5: $\quad\quad$Compute hyperboxes and $\{q_i^n\}_{i=1}^S$
6: $\quad$**end for**
7: $\quad\boldsymbol{\theta}_p \sim q(\boldsymbol{\theta})$, where $q(\boldsymbol{\theta}) = \frac{1}{N}\frac{1}{S}\sum_n \sum_i q_i^n(\boldsymbol{\theta})$
8: $\quad$Compute $\{w_p\}_p^P$ using (14)
9: $\quad$**return** $\{w_p, \boldsymbol{\theta}_p\}_{p=1}^P$
10: **end procedure**

We offer an efficient `JAX` implementation benefiting from automatic differentiation (`grad`), vectorization (`vmap`), and just-in-time compilation (`jit`). In many cases, R2OMC executes in seconds where most SBI methods require minutes (see Section 4). This happens because, R2OMC can run using (approximately) only $N*S$ unique calls to the vectorized `JAX`-based simulator, as we explain below.

For the distractors (line 2), `grad` computes $\partial g_i/\partial\theta_j$ for all i and j in one call, and `vmap` evaluates multiple $\boldsymbol{\theta}$ per seed, at once. Repeating for all seeds costs S.

Obtaining the per-data point proposal distributions $\{q_i^n\}_{i=1}^S$ (lines 3-6) consists of optimization (line 4) and hyperbox building (line 5) for each data point. Each of S optimizations requires T gradient descent steps, where T is a user-defined hyperparameter. `jit` allows compiling multiple consecutive steps to optimize execution, in many cases achieving cost similar to a single execution, so the cost reduces to $\sim S$. Hyperbox building also requires of the order of S steps when the required line searches are parallelized over each param-



(a) $S_{\texttt{base}}$ (b) $S^{\texttt{dist}}_{\texttt{base}}$ (c) $S_{\texttt{MoG}}$ (d) $S^{\texttt{dist}}_{\texttt{MoG}}$

Figure 3: MoG benchmark: Success Frontier plots showing the minimum runtime required to achieve a mean C2ST score ≤ 0.75 across varying D. Corresponding Budget vs. Dimension plots are provided in Appendix D.

eter dimensions using `vmap` (see Appendix B). Thus the total cost of obtaining the proposal distributions is of the order of $N * S$.

For generating the weighted samples, (14) is evaluated on P samples (line 8). The simulator is only used for computing $\prod_n^N \sum_i^S \mathbb{1}_{C_\epsilon^{i,n}}(\boldsymbol{\theta}_p)$ which requires evaluating $S * N$ different distance functions on P points each. Using `vmap` in each distance function, the total cost is $S * N$. Evaluating $q(\cdot)$ and $p(\cdot)$ does not require calling the simulator. Finally, if the simulator is expensive, the $S * N$ cost of checking whether the samples are in the acceptance regions can be avoided by using the hyperbox or another model for the acceptance region, see Ikonomov and Gutmann (2020).

3.4 Limitations and Discussion

While R2OMC enables efficient inference in many scenarios, it has several limitations. First, it requires differentiable simulators, making it a gray-box rather than a fully black-box method. Second, its performance depends critically on the success of the underlying optimization: if optimization fails, proposal samples will poorly approximate the posterior. In that case, increasing the budget (S in Algorithm 1) will not necessarily improve accuracy as is the case with most neural-based methods.

In the multiple-observations setting, R2OMC optimizes each observation independently. As a result, each proposed samples is ϵ-close to at least one observation but not equally-close to others, and this cannot be improved since optimization is per observation. In such cases, accepting enough proposal samples ($w_p > 0$ in Eq. (14)) requires increasing the ϵ-threshold, which can broaden the posterior beyond the ground truth. However, under the assumption that all observations originate from a single (unknown) parameter configuration, many proposal samples are expected to render all observations plausible, so ϵ need not become large.

Finally, the optimization does not explicitly incorporate the prior. This enables cheap prior sensitivity analysis but proposal regions can occasionally fall in areas with low or no prior support, which may degrade inference quality.

4 Experiments

We assess R2OMC on: (a) a Mixture of Gaussians (MoG) benchmark, (b) the synthetic SBI benchmark (Lueckmann et al., 2021) and (c) two image-based inference tasks. Together, they span diverse challenges, including high-dimensionality, distractors, multiple observations, and complex posteriors.

Methods. We evaluate R2OMC across all benchmarks. For MoG, it is compared against three established neural methods, NPE (Greenberg et al., 2019), BayesFlow (Radev et al., 2020) and Flow-Matching (Wildberger et al., 2023), using their official implementations provided by Tejero-Cantero et al. (2020). For SBI benchmark, it is compared to all methods reported by Lueckmann et al. (2021). For image-based tasks, we assess its applicability to high-dimensional problems without baseline comparisons. All experiments were run on a 12-core Dell XPS laptop (2.60GHz).

Metrics. We evaluate all methods using the `C2ST` score (Gutmann et al., 2018; Lueckmann et al., 2021), which measures similarity between two sets: samples from the approximate posterior and samples from the ground truth. A score of 0.5 indicates indistinguishable sets (best), while 1.0 indicates perfect separation (worst). As a classifier, we use a fully connected neural network and we report the mean `C2ST` score over, at least, five independent runs.

Appendix D provides additional details on the setup, methods, and metrics.

4.1 Mixture of Gaussians (MoG) benchmark

The MoG benchmark systematically evaluates the trade-off between accuracy and runtime for SBI meth-

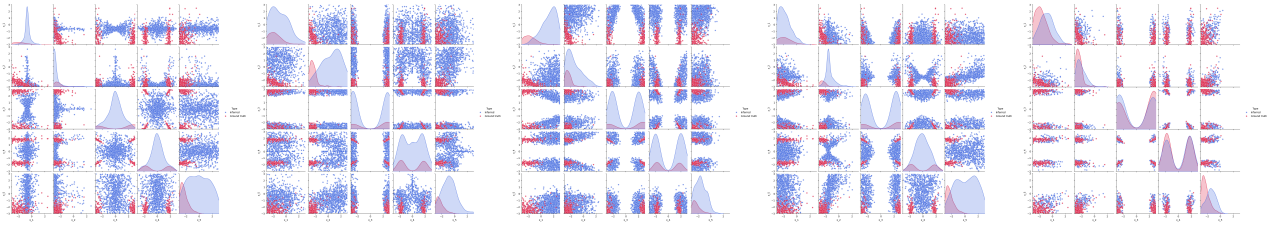


Figure 4: SLCP pairwise posteriors for multiple observations. The four left panels show proposal samples per observation within ϵ-distance, while the rightmost panel shows the subset of samples selected as closest across all observations using the R2OMC weighting scheme (Eq. 14).

ods as the parameter dimensionality D increases, while keeping the underlying problem structure fixed.

We consider two simulators:

- $S_{\texttt{base}} : \mathbf{y} \sim \mathcal{N}(\boldsymbol{\theta}, \sigma^2 \mathbf{I})$
- $S_{\texttt{MoG}} : \mathbf{y} \sim \frac{1}{2} \sum_{s \in \{+1,-1\}} \mathcal{N}(\boldsymbol{\theta} + s\boldsymbol{\mu}, \sigma^2 \mathbf{I})$

which yield uni-modal and bi-modal posteriors, respectively. For each simulator, we consider (i) a simple setting, where the output dimensionality matches the parameter vector ($D_y = D$), and (ii) a distractor setting, where 18 uninformative dimensions drawn from $\mathcal{U}(-3, 3)$ are appended to the output ($D_y = D + 18$), resulting in four simulator configurations, in total.

Observations are vectors of zeros with small noise, matching the simulator's output dimension (D or $D + 18$). For each simulator, we evaluate R2OMC against the three neural baselines introduced above, varying D from 2 to 20 and the simulation budget from 1,000 to 100,000. A uniform prior $\mathcal{U}(-\mathbf{3}, \mathbf{3})$ is used in all cases.

Figure 3 summarizes the results using success frontier plots, which show the minimum runtime required for successful inference across D. Inference is considered successful when the mean C2ST score is ≤ 0.75.

All neural-based methods follow a consistent trend: as D increases, the required simulation budget grows rapidly, often exceeding 100,000 simulations, resulting in runtimes of over an hour (see Appendix D for corresponding Budget vs. Dimension plots). Runtimes increase more steeply for $S_{\texttt{MoG}}$ than for $S_{\texttt{base}}$ and more in the distractor settings than in the simple ones. In contrast, R2OMC achieves successful inference in a few seconds across all D and simulators.

4.2 SBI benchmark

The SBI benchmark (Lueckmann et al., 2021) is widely used to evaluate SBI methods. We select: SLCP (T.3) to test inference with multiple observations, SLCP with distractors (T.4) to assess robustness to uninformative

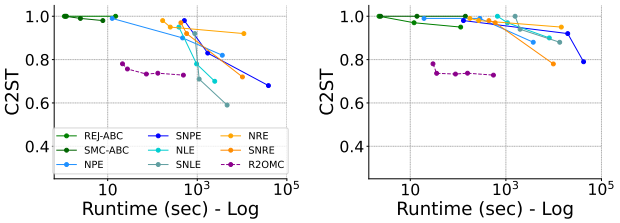


Figure 5: SLCP (left) and SLCP with distractors (right): C2ST score vs. runtime.

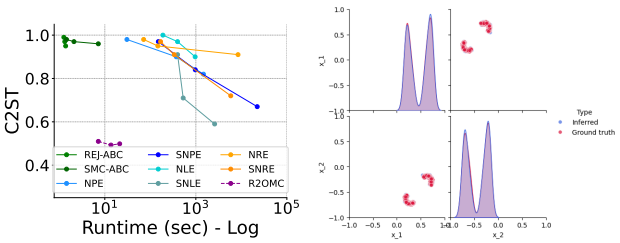


Figure 6: 2-moons task. Left: C2ST score vs. runtime. Right: pairwise posterior.

dimensions and 2-Moons (T.8) for a bimodal posterior with narrowly-concentrated modes.

4.2.1 SLCP (T.3) and plus distractors (T.4)

The SLCP (Simple Likelihood, Complex Posterior) task has a uniform prior over five parameters $\boldsymbol{\theta} \in \mathcal{U}(-3, 3)^5$ and four two-dimensional observations drawn from a Gaussian distribution whose mean and variance are nonlinear functions of $\boldsymbol{\theta}$. This results in a posterior characterized by four symmetric modes and sharp vertical boundaries. T.4 augments the original SLCP task by appending 23 non-informative dimensions to each observation.

Figure 5 reports the C2ST score versus runtime for R2OMC and the methods of Lueckmann et al. (2021) on both SLCP variants. R2OMC approximates the posterior accurately (C2ST score 0.7–0.8) in some seconds, whereas other methods require substantially longer runtimes (often hours), because they require budgets

of at least 10^4 (often 10^5) samples for similar performance. The gap is especially pronounced in the presence of distractors, where most methods fail to achieve C2ST scores below 0.8 regardless of budget.

Figure 4 illustrates how R2OMC handles multiple observations. Proposal samples are first selected per observation (within ϵ-distance) and then weighted according to Eq. 14 to retain those closest with respect to all observations. We speculate that this is why C2ST scores never reach the optimum of 0.5 and starts to flatten out: increasing the budget ensures samples are ϵ-close to individual observations, but not necessarily to the joint set. In our case, all proposed samples satisfy $\epsilon < 1$ per observation, yet the maximum distance across all observations reaches $\epsilon = 5$.

4.2.2 Two-moons (T.8)

The 2-moons evaluates SBI methods for bimodal, crescent-shaped posteriors. The simulator is $\mathbf{y}|\boldsymbol{\theta} \sim (r\cos(\alpha) + 0.25, r\sin(\alpha)) + (-|\theta_1 + \theta_2|/\sqrt{2}, (-\theta_1 + \theta_2)/\sqrt{2})$ with $\alpha \sim \mathcal{U}(-\pi/2, \pi/2)$ and $r \sim \mathcal{N}(0.1, 0.01^2)$. The problem is two-dimensional with a uniform prior $\boldsymbol{\theta} \sim \mathcal{U}(-\mathbf{1}, \mathbf{1})$, producing a posterior with two half-moon modes.

Figure 6 compares R2OMC to the methods of Lueckmann et al. (2021). R2OMC achieves a near-optimal C2STscore (≈ 0.5) in a few-seconds runtime, whereas competing methods require minutes to hours (budgets of 10^5 samples) to reach comparable performance. The pairwise posterior further demonstrates that R2OMC accurately captures both crescent-shaped modes. Additional results versus budget are reported in Appendix D.

4.3 Image-based inference

We evaluate R2OMC on high-dimensional parameter spaces (784 dimensions) using image-based inference in noisy camera models. Two simulators are considered on MNIST observations (LeCun et al., 1998): the first applies pixel-wise intensity changes, $y_{ij} = a\theta_{ij} + b + \boldsymbol{\epsilon}$, with a, b controlling contrast and brightness; the second applies edge-detection filtering, $\mathbf{y} = \text{filter}(\boldsymbol{\theta}) + \boldsymbol{\epsilon}$, via convolution with a checkerboard filter. Both use Gaussian noise $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, 0.1^2\mathbf{I})$ and uninformative prior $\boldsymbol{\theta} \sim \mathcal{U}(0,1)^{28\times 28}$.

Figure 7 shows that R2OMC recovers posterior means that visually match the clean images, indicating the posterior mode is close to the true generating parameters $\boldsymbol{\theta}^*$, despite the completely uninformative prior.

While direct comparisons are not performed here, prior work highlights that SBI on images is challenging due to the curse of dimensionality. Methods like GATSBI (Ramesh et al., 2022) can succeed but require complex generative models with long, often unstable, training. In contrast, R2OMC achieves accurate inference with just 100 simulations and a runtime of a few seconds.

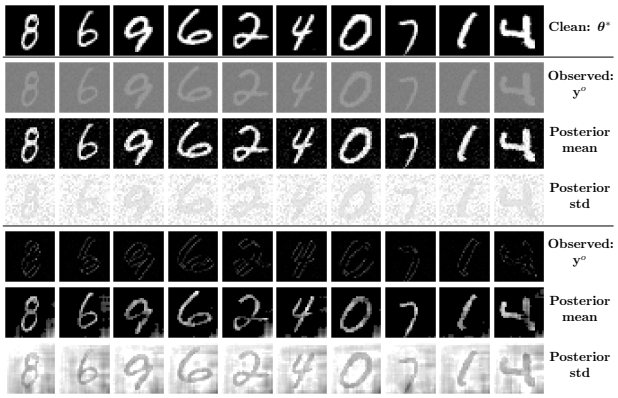


Figure 7: Image-based inference with R2OMC. Top row: clean ground truth image $\boldsymbol{\theta}^*$. Rows 2–4: distorted observation, posterior mean, and posterior standard deviation for pixel-wise intensity distortion. Rows 5–7: the same results for edge-detection filter distortion.

5 Conclusion

We have presented R2OMC, a new Bayesian inference method for differentiable simulator models that addresses two major challenges in simulation-based (likelihood-free) inference: scalability to high-dimensional parameter spaces and robustness to uninformative dimensions (distractors), all within fast runtimes.

Building on the ROMC framework, R2OMC recasts inference as a set of deterministic optimization problems solvable via gradient descent, enabling efficient inference in high-dimensional spaces. A gradient-based filtering step further masks distractors, while a novel adaptive importance sampling scheme selects posterior samples consistent with *all* available i.i.d. observations.

Our JAX-based implementation leverages auto-differentiation and vectorization to accelerate simulator evaluations, enabling many parameter–seed combinations to be executed at the cost of a single call. Across both novel experimental settings and the comprehensive SBI benchmark, R2OMC consistently achieves high accuracy—often approaching the optimal C2ST score of 0.5—while requiring only a fraction of the computational time of state-of-the-art methods.

Acknowledgments

This research was funded from the European Union's Horizon Europe research and innovation program under Grant Agreement No: 101135826 (ai-dapt.eu).

References

Jarno Lintusaari, Michael U Gutmann, Ritabrata Dutta, Samuel Kaski, and Jukka Corander. Fundamentals and recent developments in approximate bayesian computation. *Systematic biology*, 66(1): e66–e82, 2017.

Scott A Sisson, Yanan Fan, and Mark Beaumont. *Handbook of approximate Bayesian computation*. CRC press, 2018.

Kyle Cranmer, Johann Brehmer, and Gilles Louppe. The frontier of simulation-based inference. *Proceedings of the National Academy of Sciences*, 117(48): 30055–30062, 2020.

Michael Deistler, Jan Boelts, Peter Steinbach, Guy Moss, Thomas Moreau, Manuel Gloeckler, Pedro LC Rodrigues, Julia Linhart, Janne K Lappalainen, Benjamin Kurt Miller, et al. Simulation-based inference: A practical guide. *arXiv preprint arXiv:2508.12939*, 2025.

Jean-Michel Marin, Pierre Pudlo, Christian P. Robert, and Robin J. Ryder. Approximate bayesian computational methods. *Statistics and Computing*, 22(6):1167–1180, 2012. ISSN 1573-1375. doi:10.1007/s11222-011-9288-2. URL https://doi.org/10.1007/s11222-011-9288-2.

Y. Chen, D. Zhang, M. U. Gutmann, A. Courville, and Z. Zhu. Neural approximate sufficient statistics for implicit models. In *International Conference on Learning Representations (ICLR)*, 2021. URL https://openreview.net/forum?id=SRDuJssQud.

Yanzhi Chen, Michael U. Gutmann, and Adrian Weller. Is learning summary statistics necessary for likelihood-free inference? In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 4529–4544. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/chen23h.html.

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

Leah F Price, Christopher C Drovandi, Anthony Lee, and David J Nott. Bayesian synthetic likelihood. *Journal of Computational and Graphical Statistics*, 27(1):1–11, 2018.

Owen Thomas, Ritabrata Dutta, Jukka Corander, Samuel Kaski, and Michael U Gutmann. Likelihood-free inference by ratio estimation. *Bayesian Analysis*, 17(1):1–31, 2022.

Conor Durkan, Iain Murray, and George Papamakarios. On contrastive learning for likelihood-free inference. In *International conference on machine learning*, pages 2771–2781. PMLR, 2020.

David Greenberg, Marcel Nonnenmacher, and Jakob Macke. Automatic posterior transformation for likelihood-free inference. In *International Conference on Machine Learning*, pages 2404–2414. PMLR, 2019.

George Papamakarios and Iain Murray. Fast ε-free inference of simulation models with bayesian conditional density estimation. *Advances in neural information processing systems*, 29, 2016.

Stefan T Radev, Ulf K Mertens, Andreas Voss, Lynton Ardizzone, and Ullrich Köthe. Bayesflow: Learning complex stochastic models with invertible neural networks. *IEEE transactions on neural networks and learning systems*, 33(4):1452–1466, 2020.

Jonas Wildberger, Maximilian Dax, Simon Buchholz, Stephen Green, Jakob H Macke, and Bernhard Schölkopf. Flow matching for scalable simulation-based inference. *Advances in Neural Information Processing Systems*, 36:16837–16864, 2023.

George Papamakarios, David Sterratt, and Iain Murray. Sequential neural likelihood: Fast likelihood-free inference with autoregressive flows. In *The 22nd international conference on artificial intelligence and statistics*, pages 837–848. PMLR, 2019.

A. Saltelli, M. Ratto, T. Andres, F. Campolongo, J. Cariboni, D. Gatelli, M. Saisana, and S. Tarantola. *Global Sensitivity Analysis: The Primer*. John Wiley & Sons, 2008.

Gloria M. Monsalve-Bravo, Brodie A. J. Lawson, Christopher Drovandi, Kevin Burrage, Kevin S. Brown, Christopher M. Baker, Sarah A. Vollert, Kerrie Mengersen, Eve McDonald-Madden, and Matthew P. Adams. Analysis of sloppiness in model simulations: Unveiling parameter uncertainty when mathematical models are fitted to data. *Science Advances*, 8(38):eabm5952, 2022. doi:10.1126/sciadv.abm5952. URL https://www.science.org/doi/abs/10.1126/sciadv.abm5952.

Jan-Matthis Lueckmann, Jan Boelts, David Greenberg, Pedro Goncalves, and Jakob Macke. Benchmarking simulation-based inference. In *International conference on artificial intelligence and statistics*, pages 343–351. PMLR, 2021.

Borislav Ikonomov and Michael U Gutmann. Robust optimisation monte carlo. In *International Conference on Artificial Intelligence and Statistics*, pages 2819–2829. PMLR, 2020.

Vasilis Gkolemis, Michael Gutmann, and Henri Pesonen. An extendable python implementation of

robust optimisation monte carlo. *arXiv preprint arXiv:2309.10612*, 2023.

Simon Dirmeier, Simone Ulzega, Antonietta Mira, and Carlo Albert. Simulation-based inference with the python package sbijax. *arXiv preprint arXiv:2409.19435*, 2024.

Ted Meeds and Max Welling. Optimization monte carlo: Efficient and embarrassingly parallel likelihood-free inference. *Advances in Neural Information Processing Systems*, 28, 2015.

Alvaro Tejero-Cantero, Jan Boelts, Michael Deistler, Jan-Matthis Lueckmann, Conor Durkan, Pedro J. Gonçalves, David S. Greenberg, and Jakob H. Macke. sbi: A toolkit for simulation-based inference. *Journal of Open Source Software*, 5(52):2505, 2020. doi:10.21105/joss.02505. URL https://doi.org/10.21105/joss.02505.

Michael U Gutmann, Ritabrata Dutta, Samuel Kaski, and Jukka Corander. Likelihood-free inference via classification. *Statistics and Computing*, 28:411–425, 2018.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11): 2278–2324, 1998.

Poornima Ramesh, Jan-Matthis Lueckmann, Jan Boelts, Álvaro Tejero-Cantero, David S Greenberg, Pedro J Gonçalves, and Jakob H Macke. Gatsbi: Generative adversarial training for simulation-based inference. *arXiv preprint arXiv:2203.06481*, 2022.

A Simulation Budget

In simulation-based inference (SBI), an important consideration is the ability of a method to infer the posterior distribution under a restricted simulation budget. The term *simulation budget* typically refers to the number of allowed simulator evaluations — that is, the number of times the simulator is called to generate synthetic data. Each simulator call produces a pair $(\boldsymbol{\theta}_i, \mathbf{y}_i)$, where $\mathbf{y}_i$ denotes the simulated observation corresponding to parameters $\boldsymbol{\theta}_i$.

Recently, with the advent of neural-based SBI methods, the simulation budget is often associated with the number of unique parameter–observation pairs used to train the neural density estimator. In multi-round or active learning approaches, simulations from previous rounds can be reused for training, so the total number of simulator calls and the size of the training dataset are not necessarily identical. Nevertheless, the number of unique simulations remains limited by the simulation budget, and neural estimators rely on these unique instances to learn an accurate posterior.

The simulation budget is an important consideration because it directly relates to computational cost. This arises for two main reasons: (a) each simulator call can be computationally expensive, and (b) training the neural density estimator on the resulting simulated dataset requires additional computation, which grows with dataset size. A common third cost is the time required to draw samples from the trained estimator; however, this cost is typically independent of the simulation budget and is therefore not counted here. Therefore, simulation budget typically affects costs incured by (a) and/or (b).

For neural-based SBI methods, the simulation budget impacts both sources of computational cost: (a) the number of simulator calls and (b) the cost of training the neural density estimator. In contrast, for more traditional likelihood-free inference methods, such as approximate Bayesian computation (ABC), the simulation budget primarily affects (a), since these methods do not explicitly train a separate estimator.

Lately, modern computational frameworks, such as `JAX`, allow simulators to be implemented in a way that accelerates computation through vectorization. For example, a batch of parameters $\boldsymbol{\theta}_i$ for $i = 1, \ldots, N$ can be evaluated on the same random seed s, producing outputs $\mathbf{y}ii = 1^N$ at roughly the cost of a single simulator evaluation. More generally, a grid of parameter–seed pairs $\boldsymbol{\theta}_i, s_j$ for $i = 1, \ldots, N$ and $j = 1, \ldots, S$ can also be computed efficiently, effectively generating $N \times S$ simulated pairs $(\boldsymbol{\theta}, \mathbf{y})$ at the cost of a 'vectorized' evaluation, i.e., in significantly reduced computational overhead compared to $N \times S$ sequential evaluations.

These advancements can substantially reduce computational cost (a), as a single simulator call can now produce multiple $(\boldsymbol{\theta}, \mathbf{y})$ pairs through vectorized evaluation. However, they do not reduce cost (b), since training the neural density estimator still scales with the total number of simulated instances. Therefore, although, this advancements benefits neural-based methods in terms of cost incured by (a), does not help them for (b). This motivates the development of a method that avoids the computational cost of training an external neural estimator.

Therefore, we clarify that for R2OMC, the term simulation budget is not directly applicable as a metric and, when we use it for comparison purposes, it refers to the number of independent calls to a vectorized simulator, where each call can produce multiple $(\boldsymbol{\theta}, \mathbf{y})$ pairs. This definition emphasizes that for R2OMC, the budget counts independent simulator evaluations rather than the total number of resulting dataset instances.

B Methods

Below, we provide information and the default hypeparameter setting that we use in the four methods we compare in the Introductory Example (Figure 1) and the Mixture of Gaussians (MoG) benchmark 4.1.

R2OMC. A detailed description of the method can be found in Algorithm 1.

In Step 2 of Algorithm 1, i.e., computing the uninformative dimensions, we typically use around 50 samples from the distribution $p(u)$ and another 50 samples from $p(\boldsymbol{\theta})$, with the parameter τ set to machine precision, i.e., $\tau \approx 0$. Similar results can be obtained with fewer samples, such as 10 or 20.

For Step 3, i.e., computing the optimal points $\boldsymbol{\theta}^*_{i,n}$, we use the Adam optimizer, with a learning rate between 0.01 and 0.2, depending on the specific problem. Our default choice is 0.01. The number of optimization steps varies between 50 and 400, with 50 as the default. The distance function $d(\boldsymbol{\theta})$ is defined as the squared Euclidean distance between the simulator output and the observation, i.e., $d(\boldsymbol{\theta}) = \|\mathbf{y} - \mathbf{y}^o_n\|_2^2$. The number of seeds S used ranges from 100 to 10,000, with 1000 as the default. The number of seeds corresponds to the simulation budget; for example, if a budget of 10,000 runs is available, we use (at most) 10,000 seeds. Typically, we select 80% of the seeds based on the best $d^n_i(\boldsymbol{\theta}^*_{i,n})$. However, the percentage of accepted seeds can vary between 50% and 100%, depending on the absolute values of $d^n_i(\boldsymbol{\theta}^*_{i,n})$ for $i = 1, \ldots, S$.

For Step 5, i.e., constructing the hyperboxes, we use Algorithm 2. We normally set ϵ to be twice the distance of the 'worst' accepted seed, i.e., $\epsilon = 2 \max_i d^n_i(\boldsymbol{\theta}^*_{i,n})$, where i is an index over the accepted seeds. Alternatively, ϵ can be set to a fixed value such as 0.1. The typical settings for step size, number of steps, and refinements are $\eta = 0.1$, $L = 100$, and $R = 1$, respectively.

For Step 7, i.e., sampling from the proposal distribution, we generate between 1000 and 100,000 candidate samples, with the default being 2000. From these candidates, we select 1000 via weighted sampling. For computing the weights, $\{w_p\}_{p=1}^P$, we set ϵ to a value about twice the distance of the 'worst' accepted seed, i.e., $\epsilon = 2 \max_{i,n} d^n_i(\boldsymbol{\theta}^*_{i,n})$. However, this can vary depending on the problem, and by testing different values.

Algorithm 2 Hyperbox Computation
Input: Optimal point $\boldsymbol{\theta}^*$, distance function $d(\boldsymbol{\theta})$, Jacobian $\mathbf{J}$ at $\boldsymbol{\theta}^*$
Parameters: Step size η, number of refinements R, number of steps L, distance threshold ϵ

Compute eigenvectors $\mathbf{v}_d$ of $\mathbf{J}^T\mathbf{J}$ $(d = 1, \ldots, D)$
for $d = 1$ to D **do**
  Initialize endpoint $\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta}^*$
  Set refinement counter $r \leftarrow 0$
  repeat
    Set step counter $l \leftarrow 0$
    repeat
      Increment step counter $l \leftarrow l + 1$
      Update $\tilde{\boldsymbol{\theta}} \leftarrow \tilde{\boldsymbol{\theta}} + \eta \mathbf{v}_d$
    until $d(\boldsymbol{\theta}) > \epsilon$ or $l = L$
    Move back one step $\tilde{\boldsymbol{\theta}} \leftarrow \tilde{\boldsymbol{\theta}} - \eta \mathbf{v}_d$
    Half the step size $\eta \leftarrow \eta/2$
    Increment refinement counter $r \leftarrow r + 1$
  until $r = R$
  Set $\tilde{\boldsymbol{\theta}}$ as the endpoint
  Repeat steps 3-15 for $\mathbf{v}_d = -\mathbf{v}_d$ and set $\tilde{\boldsymbol{\theta}}$ as the negative endpoint
end for
Define a uniform distribution q over the hyperbox
return q

NPE For Neural Posterior Estimation (NPE), we use the variant denoted as `NPE_C` (Greenberg et al., 2019) in the SBI benchmark (Lueckmann et al., 2021). `NPE_C` estimates the posterior by training a neural network $F(\mathbf{y}, \boldsymbol{\phi})$ to approximate $p(\boldsymbol{\theta} \mid \mathbf{y})$ through a density estimator $q_F(\mathbf{y}, \boldsymbol{\phi})(\boldsymbol{\theta})$. In the experiments, we use a neural spline flow

as density estimator (`sbi.utils.get_nn_models.posterior_nn()`). The NPE configuration is summarized in Table 1.

Table 1: Configuration of the Neural Posterior Estimation (NPE_C) method.

Component	Parameter	Value
Density estimator (`posterior_nn`)	`model`	`'nsf'`
	`hidden_features`	100
	`num_transforms`	8
	`num_bins`	10
	`z_score_x`	`'independent'`
	`z_score_theta`	`'independent'`
Training configuration (`.train()`)	`training_batch_size`	500
	`max_num_epochs`	1000
	`force_first_round`	`True`

BayesFlow. BayesFlow (Radev et al., 2020) is equivalent to Neural Posterior Estimation (NPE) with the key contribution of adding an embedding network that automatically learns from the data appropriate summary statistics. The `sbi` package provides access to these embeddings through the `sbi.neural_nets.embedding_nets()` interface. In our experiments, we use the fully connected embedding network (`FCEmbedding`) and the default BayesFlow density estimator. The corresponding configurations are summarized in Table 2.

Table 2: Configuration of the BayesFlow embedding network and density estimator.

Component	Parameter	Value
Embedding network (`FCEmbedding`)	`output_dim`	20
	`num_layers`	2
	`num_hiddens`	50
Density estimator (`posterior_nn`)	`model`	`'nsf'`
	`hidden_features`	100
	`num_transforms`	8
	`z_score_x`	`'independent'`
	`z_score_theta`	`'independent'`
Training configuration (`.train()`)	`training_batch_size`	500
	`max_num_epochs`	1000
	`force_first_round`	`True`

FMPE. Flow Matching Posterior Estimation (FMPE) (Wildberger et al., 2023) builds upon advances in generative modeling with continuous normalizing flows. Similar in spirit to diffusion models, FMPE replaces discrete flow transformations with a continuous-time formulation based on flow matching. In our experiments, we use the `sbi` implementation (`.FMPE()`) with a multilayer perceptron (`mlp`) as vector field estimator (`vf_estimator`). The default configuration is summarized in Table 3.

Table 3: Configuration of the Flow Matching Posterior Estimation (FMPE) method.

Component	Parameter	Value
Vector field estimator	`vf_estimator`	`'mlp'`
Training configuration (`.train()`)	`training_batch_size`	500
	`max_num_epochs`	1000
	`force_first_round_loss`	`True`

C Introductory Example

C.1 Problem Definition

We provide further details on the introductory example of Figure 1. The example comes with three variants, that we refer to as: (1) **Simple**, (2) **High-dimensional**, and (3) **Distractors**. Simple (1) and High-dimensional (2) share the same setup. Between them, the only difference is the parameter dimension D and the output dimension D_y which are: $D = D_y = 2$ in (1) and $D = D_y = 10$ in (2).

In both (1) and (2), the simulator is:

$$\mathbf{y} \sim \frac{1}{2} \sum_{s \in \{+1,-1\}} \mathcal{N}(\boldsymbol{\theta} + s\boldsymbol{\mu}, \sigma^2 \mathbf{I}), \quad \boldsymbol{\mu} = \mathbf{1},\ \sigma = 0.2. \tag{15}$$

In the Distractors (3) variant, the parameter space is $D = 2$, but we add $D_{\text{dist}} = 18$ distractor dimensions to the output, yielding $D_y = 20$. Thus, in (3), the simulator becomes:

$$\mathbf{y} = (\mathbf{y}^{(1)}, \mathbf{y}^{(2)}), \quad \mathbf{y}^{(1)} \sim \frac{1}{2} \sum_{s \in \{+1,-1\}} \mathcal{N}(\boldsymbol{\theta} + s\boldsymbol{\mu}, \sigma^2 \mathbf{I}), \quad \mathbf{y}^{(2)} \sim \mathcal{U}(-\mathbf{3}, \mathbf{3}), \tag{16}$$

where $\boldsymbol{\mu} = \mathbf{1}$ and $\sigma = 0.2$. For all variants, the prior is $\mathcal{U}(-\mathbf{3}, \mathbf{3})$ and the observation is $\mathbf{y}^o = \mathbf{0} \in \mathbb{R}^{D_y}$.

C.2 Ground Truth Posterior

Simple and High-dimensional Case. The posterior is given by

$$\begin{aligned}
p(\boldsymbol{\theta} \mid \mathbf{y}^o) &\propto p(\boldsymbol{\theta}) p(\mathbf{y}^o \mid \boldsymbol{\theta}) && (17)\\
&\propto p(\boldsymbol{\theta}) \left[\mathcal{N}(\mathbf{y}^o; \boldsymbol{\theta} - \boldsymbol{\mu}, \sigma^2 \mathbf{I}) + \mathcal{N}(\mathbf{y}^o; \boldsymbol{\theta} + \boldsymbol{\mu}, \sigma^2 \mathbf{I})\right] && (18)\\
&\propto p(\boldsymbol{\theta}) \left[\mathcal{N}(\boldsymbol{\theta}; \boldsymbol{\mu}, \sigma^2 \mathbf{I}) + \mathcal{N}(\boldsymbol{\theta}; -\boldsymbol{\mu}, \sigma^2 \mathbf{I})\right] && (19)\\
&\propto \begin{cases} \mathcal{N}(\boldsymbol{\theta}; \boldsymbol{\mu}, \sigma^2 \mathbf{I}) + \mathcal{N}(\boldsymbol{\theta}; -\boldsymbol{\mu}, \sigma^2 \mathbf{I}), & \boldsymbol{\theta} \in [-3, 3]^D, \\ 0, & \text{otherwise.} \end{cases} && (20)
\end{aligned}$$

From (18) to (19), we use the property: $\mathcal{N}(\mathbf{y}; \boldsymbol{\theta} - \boldsymbol{\mu}, \Sigma) = \mathcal{N}(\boldsymbol{\theta}; \mathbf{y} + \boldsymbol{\mu}, \Sigma)$.

Distractors Case. In the distractors variant, the posterior over $\boldsymbol{\theta}$ remains unchanged. Formally:

$$\begin{aligned}
p(\boldsymbol{\theta} \mid \mathbf{y}^o) &\propto p(\boldsymbol{\theta}) p(\mathbf{y}^o \mid \boldsymbol{\theta}) && (21)\\
&\propto p(\boldsymbol{\theta}) p((\mathbf{y}^{o,(1)}, \mathbf{y}^{o,(2)}) \mid \boldsymbol{\theta}) && (22)\\
&\propto p(\boldsymbol{\theta})\, p(\mathbf{y}^{o,(1)} \mid \boldsymbol{\theta})\, p(\mathbf{y}^{o,(2)} \mid \boldsymbol{\theta}) && (23)\\
&\propto p(\boldsymbol{\theta})\, p(\mathbf{y}^{o,(1)} \mid \boldsymbol{\theta})\, p(\mathbf{y}^{o,(2)}) && (24)\\
&\propto \begin{cases} \mathcal{N}(\boldsymbol{\theta}; \boldsymbol{\mu}, \sigma^2 \mathbf{I}) + \mathcal{N}(\boldsymbol{\theta}; -\boldsymbol{\mu}, \sigma^2 \mathbf{I}), & \boldsymbol{\theta} \in [-3, 3]^D, \\ 0, & \text{otherwise.} \end{cases} && (25)
\end{aligned}$$

Steps (22) to (23) use independence between $\mathbf{y}^{o,(1)}$ and $\mathbf{y}^{o,(2)}$, and steps (23) to (24) use that $\mathbf{y}^{o,(2)}$ is independent of $\boldsymbol{\theta}$.

C.3 Experimental Setup and Hyperparameter Configuration

All methods were evaluated for simulation budgets of $[1{,}000, 10{,}000, 30{,}000, 40{,}000, 50{,}000]$. For each method, we drew 1,000 samples from the approximate posterior and compared them against 1,000 ground-truth samples using the C2ST score. Each experiment was repeated across 5 independent random seeds, so we report the average C2ST scores among the independent runs.

The experiment goal is to analyze the performance of each method in two regimes. The **low-budget/low-runtime** regime, where the budget is always 1,000 and the **high-budget/high-runtime** regime, where the budget is between 10,000 and 50,000. In the low-runtime regime, we simply report the mean C2ST score and runtime for a budget of 1,000. In the high-runtime regime, we select the minimum budget that achieves an average C2ST below 0.65. If no budget meets this threshold, we select the budget with the lowest C2ST (which is always 50,000 in this case). We report the mean C2ST and runtime of the selected budget

Below, we detail the hyperparameter configuration for each method, highlighting the deviations from the default settings.

ROMC For R2OMC, we used at most a maximum of $S = 1{,}000$ seeds, corresponding to roughly 1,000 independent simulator calls, as this budget was enough for a near-optimal C2ST score. The modified fitting parameters are: `{"pcg_to_keep":1.}`, i.e., we keep all optimization ending points $\{\boldsymbol{\theta}_i^*\}_{i=1}^S$.

NPE-C The modified fitting parameters are: `{"batch_size":100, "training_batch_size":100}`

BayesFlow The modified fitting parameters are: `{"embedding_net_output_dim": 2}` for the Simple and Distractor variants, and `10` for the High-dimensional variant.

Flow-Matching All arguments were default.

C.4 Conclusions

Figure 1 illustrates the key-findings presented below.

In the simple scenario (1), most methods perform well across both low- and high-budget regimes. Only FMPE requires a substantial simulation budget and runtime to succeed, whereas the remaining methods already achieve good performance under low-budget conditions.

However, the picture changes as the task becomes more challenging—either through the addition of distractors (2) or by increasing dimensionality (3). In these settings, all neural-based methods fail to perform adequately with a low simulation budget and require significantly higher budgets, and consequently higher runtimes, to succeed. BayesFlow, in particular, fails to converge even under the high-budget regime. In contrast, R2OMC achieves competitive performance using only 1,000 simulations, resulting in substantially lower runtime requirements.

Overall, the results highlight an important trend: as dimensionality increases or distractor dimensions are introduced, neural-based methods demand increasingly larger simulation budgets to achieve reliable performance. While a budget of 50,000 simulations is not prohibitive—modern infrastructure can handle even 100,000 simulations within reasonable runtimes—the observed trend is clear. With increases in dimensionality, additional distractors, or more complex simulators (beyond simple Gaussian noise), the required number of simulations grows rapidly. Consequently, under strict budget and runtime constraints, these neural-based approaches are likely to face significant limitations.

D Experiments: Additional Information

We here provide some additional information on the experiments presented in the main text.

D.1 Mixture of Gaussians (MoG) Benchmark

The Mixture of Gaussians (MoG) benchmark evaluates how simulation-based inference (SBI) methods scale with increasing problem dimensionality. Specifically, it assesses whether higher-dimensional problems require larger simulation budgets to achieve accurate inference, i.e., whether there exists a correlation between dimensionality and the required number of simulations.

To this end, we consider four simulator variants: $S_{\texttt{base}}$ (1), $S^{\texttt{dist}}_{\texttt{base}}$ (2), $S_{\texttt{MoG}}$ (3), and $S^{\texttt{dist}}_{\texttt{MoG}}$ (4). Each simulator is tested across different dimensionalities, $D = 2, 5, 10, 15, 20$. For each configuration, we evaluate every method's performance across multiple simulation budgets using the C2ST metric.

D.1.1 Problem definition and ground truth posterior

Base simulator - without distractors (1):

Prior $\quad \boldsymbol{\theta} \sim \mathcal{U}(-\mathbf{3}, \mathbf{3})$

Simulator $\quad \mathbf{y} \sim \mathcal{N}(\boldsymbol{\theta} + \boldsymbol{\mu}, \sigma^2\mathbf{I})$, with $\boldsymbol{\mu} = \mathbf{1}, \sigma^2 = 0.2^2$

Dimensionality $\quad \boldsymbol{\theta} \in \mathbb{R}^D$, $\mathbf{y} \in \mathbb{R}^D$

Observation $\quad \mathbf{y}^o = (0, \dots, 0)$ where $\mathbf{y}^o \in \mathbb{R}^D$

Posterior $\quad p(\boldsymbol{\theta}|\mathbf{y}^o) \propto \begin{cases} \mathcal{N}(\boldsymbol{\theta}; -\boldsymbol{\mu}, \sigma^2\mathbf{I}), & \text{if } \boldsymbol{\theta} \in [-3, 3]^D, \\ 0, & \text{otherwise.} \end{cases}$

Proof of the posterior:

$$
\begin{aligned}
p(\boldsymbol{\theta}|\mathbf{y}^o) &\propto p(\boldsymbol{\theta}) p(\mathbf{y}^o|\boldsymbol{\theta}) && (26)\\
&\propto p(\boldsymbol{\theta}) \mathcal{N}(\mathbf{y}^o; \boldsymbol{\theta} + \boldsymbol{\mu}, \sigma^2\mathbf{I}) && (27)\\
&\propto p(\boldsymbol{\theta}) \mathcal{N}(\boldsymbol{\theta}; \mathbf{y}^o - \boldsymbol{\mu}, \sigma^2\mathbf{I}) && (28)\\
&\propto \begin{cases} \mathcal{N}(\boldsymbol{\theta}; -\boldsymbol{\mu}, \sigma^2\mathbf{I}), & \text{if } \boldsymbol{\theta} \in [-3, 3]^D, \\ 0, & \text{otherwise.} \end{cases} && (29)
\end{aligned}
$$

From (27) to (28), we apply the property: $\mathcal{N}(\mathbf{y}; \boldsymbol{\theta} + \boldsymbol{\mu}, \Sigma) = \mathcal{N}(\boldsymbol{\theta}; \mathbf{y} - \boldsymbol{\mu}, \Sigma)$.

Base simulator - with distractors (2):

Prior $\quad \boldsymbol{\theta} \sim \mathcal{U}(-\mathbf{3}, \mathbf{3})$

Simulator $\quad \mathbf{y} = (\mathbf{y}^{(\mathbf{1})}, \mathbf{y}^{(\mathbf{2})})$,
$\mathbf{y}^{(\mathbf{1})} \sim \mathcal{N}(\boldsymbol{\theta} + \boldsymbol{\mu}, \sigma^2\mathbf{I})$, $\boldsymbol{\mu} = \mathbf{1}$, $\sigma^2 = 0.2^2$,
$\mathbf{y}^{(\mathbf{2})} \sim \mathcal{U}(-\mathbf{3}, \mathbf{3})$

Dimensionality $\quad \boldsymbol{\theta} \in \mathbb{R}^D, \mathbf{y} \in \mathbb{R}^{D+18}$, $\mathbf{y}^{(1)} \in \mathbb{R}^D$, $\mathbf{y}^{(2)} \in \mathbb{R}^{18}$

Observation $\quad \mathbf{y}^o = (\mathbf{y}^{\mathbf{o},(\mathbf{1})}, \mathbf{y}^{\mathbf{o},(\mathbf{2})})$, where both $\mathbf{y}^{\mathbf{o},(\mathbf{1})}$ and $\mathbf{y}^{\mathbf{o},(\mathbf{2})}$ are $(0, \dots, 0)$

Posterior $\quad p(\boldsymbol{\theta}|\mathbf{y}^o) \propto \begin{cases} \mathcal{N}(\boldsymbol{\theta}; -\boldsymbol{\mu}, \sigma^2\mathbf{I}), & \text{if } \boldsymbol{\theta} \in [-3, 3]^D, \\ 0, & \text{otherwise.} \end{cases}$

The ground truth posterior is the same as in (29), as the distractors do not affect the posterior, see (25).

MoG simulator - without distractors (3):

Prior	$\boldsymbol{\theta} \sim \mathcal{U}(-\mathbf{3}, \mathbf{3})$
Simulator	$\mathbf{y} \sim \frac{1}{2}\sum_{s\in\{-1,+1\}} \mathcal{N}(\boldsymbol{\theta} + s\boldsymbol{\mu}, \sigma^2\mathbf{I})$, $\boldsymbol{\mu} = \mathbf{1}$ and $\sigma^2 = 0.2^2$
Dimensionality	$\boldsymbol{\theta} \in \mathbb{R}^D$, $\mathbf{y} \in \mathbb{R}^D$
Observation	$\mathbf{y}^o = (0, \dots, 0)$ where $\mathbf{y}^o \in \mathbb{R}^D$
Posterior	$p(\boldsymbol{\theta}\|\mathbf{y}^o) \propto \begin{cases} \mathcal{N}(\boldsymbol{\theta}; \sum_{s\in\{-1,+1\}} \mathcal{N}(s\boldsymbol{\mu}, \sigma^2\mathbf{I}), & \text{if } \boldsymbol{\theta} \in [-3,3]^D, \\ 0, & \text{otherwise.} \end{cases}$

The MoG simulator is the same as in the introductory example, so the proof is equivalent to Section C.

MoG simulator - with distractors (4):

Prior	$\boldsymbol{\theta} \sim \mathcal{U}(-\mathbf{3}, \mathbf{3})$
Simulator	$\mathbf{y} = (\mathbf{y^{(1)}}, \mathbf{y^{(2)}})$, $\mathbf{y^{(1)}} \sim \frac{1}{2}\sum_{s\in\{-1,+1\}} \mathcal{N}(\boldsymbol{\theta} + s\boldsymbol{\mu}, \sigma^2\mathbf{I})$, $\boldsymbol{\mu} = \mathbf{1}$ and $\sigma^2 = 0.2^2$ $\mathbf{y^{(2)}} \sim \mathcal{U}(-\mathbf{3}, \mathbf{3})$
Dimensionality	$\boldsymbol{\theta} \in \mathbb{R}^D$, $\mathbf{y} \in \mathbb{R}^{D+18}$, $\mathbf{y}^{(1)} \in \mathbb{R}^D$, $\mathbf{y}^{(2)} \in \mathbb{R}^{18}$
Observation	$\mathbf{y}^o = (\mathbf{y^{o,(1)}}, \mathbf{y^{o,(2)}})$, where both $\mathbf{y^{o,(1)}}$ and $\mathbf{y^{o,(2)}}$ are $(0, \dots, 0)$
Posterior	$p(\boldsymbol{\theta}\|\mathbf{y}^o) \propto \begin{cases} \mathcal{N}(\boldsymbol{\theta}; \sum_{s\in\{-1,+1\}} \mathcal{N}(s\boldsymbol{\mu}, \sigma^2\mathbf{I}), & \text{if } \boldsymbol{\theta} \in [-3,3]^D, \\ 0, & \text{otherwise.} \end{cases}$

The MoG simulator is the same as in the introductory example, so the proof is equivalent to Section C.

D.1.2 Experimental Setup and Hyperparameter Configuration

For each of the four problems (1)–(4) and across all dimensionalities $D \in \{2, 5, 10, 15, 20\}$, we evaluated all methods under simulation budgets of [1,000, 5,000, 10,000, 50,000, 100,000], recording both the C2ST score and the runtime.

To reduce experimental cost, we adopted an early-stopping criterion: if a method achieved a C2ST score below 0.75 for a given dimensionality D at a certain budget, we did not evaluate it at higher budgets. Each experiment was repeated with three independent random seeds, and we report the average C2ST scores across these runs. For each method, we drew 1,000 samples from the approximate posterior and compared them against 1,000 ground-truth samples using the C2ST metric.

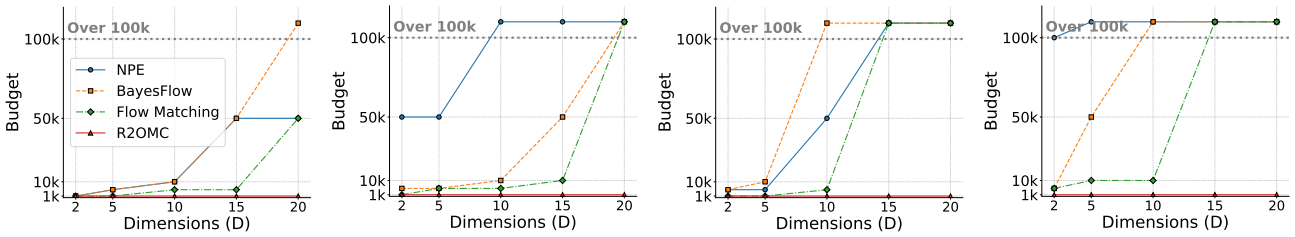
The goal of this experiment is to assess whether the required simulation budget—and consequently, the runtime—increase with problem dimensionality. To this end, we identify the minimum budget at which each method achieves successful inference, defined as an average C2ST score below 0.75. We report both the mean C2ST and mean runtime across the three runs.

Below, we detail the hyperparameter configuration for each method, emphasizing deviations from the default settings.

ROMC For R2OMC, we used at most a maximum of $S = 1{,}000$ seeds, corresponding to roughly 1,000 independent simulator calls, as this budget was enough for a near-optimal C2ST score. The modified fitting parameters are: `{"pcg_to_keep":1., "epochs": 10, "alpha": 0.1}` and the modified sampling parameters are: `{"samples_per_region": 2}`.

NPE-C The modified fitting parameters are: `{"num_transforms":16, "num_bins":16 "hidden_features":200, "training_batch_size": 500}`

BayesFlow The modified fitting parameters are: `{ "batch_size": 500, "training_batch_size": 500, "embedding_net_num_layers": 2, "embedding_net_num_hiddens": 32 }`. Finally, the parameter `{"embedding_net_output_dim": dim}` is for the non-distractor variants ((1), (3)) and `dim + 18` for the distractor variants ((2), (4)).



(a) $S_{\texttt{base}}$ (b) $S^{\texttt{dist}}_{\texttt{base}}$ (c) $S_{\texttt{MoG}}$ (d) $S^{\texttt{dist}}_{\texttt{MoG}}$

Figure 8: MoG benchmark: Success Frontier plots showing the lowest budget required to reach a C2ST score ≤ 0.75 for varying D.

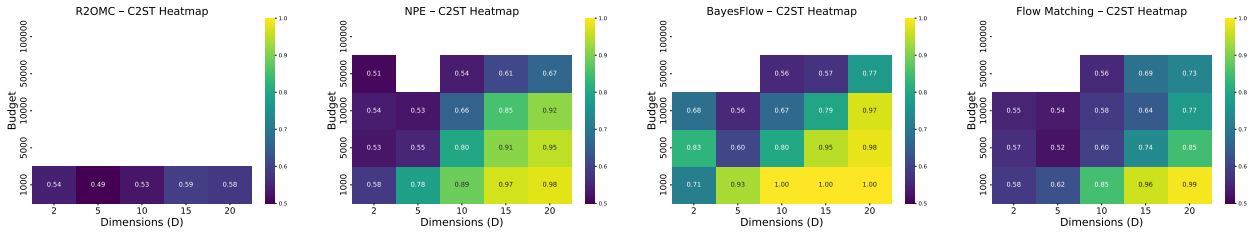


Figure 9: MoG Benchmark - Base simulator: C2ST heatmaps for R2OMC, NPE, BayesFlow, and Flow Matching.

Flow-Matching The modified fitting parameters are: `{"batch_size": 100, "training_batch_size": 100}`.

D.1.3 Results

Figure 3 in the main paper summarizes the results using success frontier plots, which depict the minimum runtime required to achieve successful inference across different dimensions D. Inference is deemed successful when the mean `C2ST` score is less than or equal to 0.75. The corresponding plots using the simulation budget (instead of runtime) are shown in Figure 8. This figure complements Figure 3 validating that the factor that incurs longer runtimes is the increased simulation budget required at higher dimensionalities.

For a more detailed view, Figures 9 ($\mathcal{S}_{\texttt{base}}$), 10 ($\mathcal{S}^{\texttt{dist}}_{\texttt{base}}$), 11 ($\mathcal{S}_{\texttt{MoG}}$), and 12 ($\mathcal{S}^{\texttt{dist}}_{\texttt{MoG}}$) report the mean `C2ST` score for each method across all dimensionalities and budgets.

All neural-based methods exhibit a consistent trend: as D increases, the simulation budget required for accurate posterior inference grows rapidly, often exceeding 100,000 simulations and resulting in runtimes that often exceed one hour. Runtimes increase more sharply for $S_{\texttt{MoG}}$ compared to $S_{\texttt{base}}$, and are higher in the distractor settings than in the simpler ones.

In contrast, R2OMC achieves successful inference within a few seconds across all dimensionalities and simulator configurations.

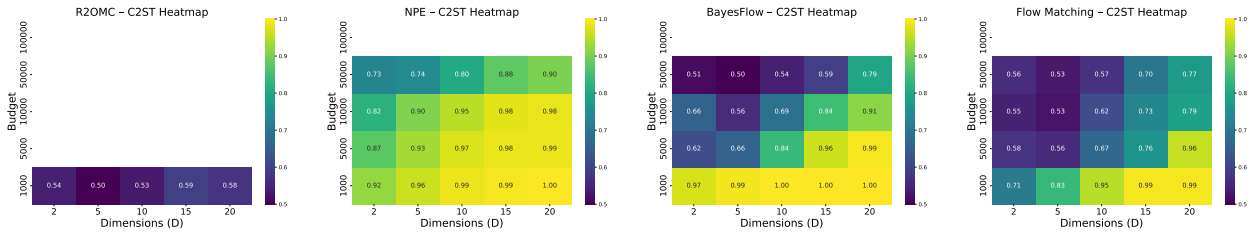


Figure 10: MoG Benchmark - Base simulator with distractors: C2ST heatmaps for R2OMC, NPE, BayesFlow, and Flow Matching.

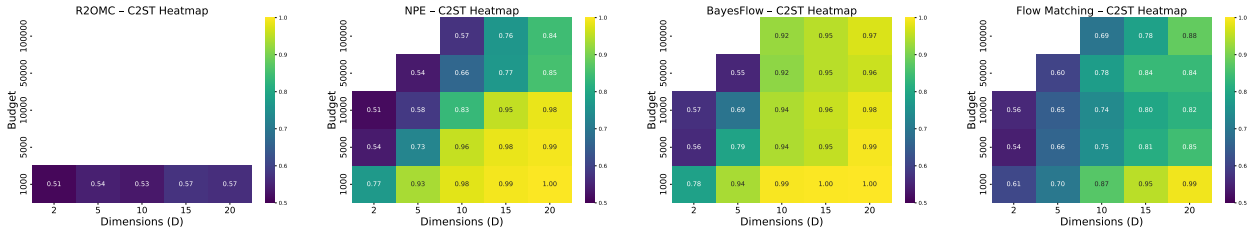


Figure 11: MoG Benchmark - MoG simulator: C2ST heatmaps for R2OMC, NPE, BayesFlow, and Flow Matching.

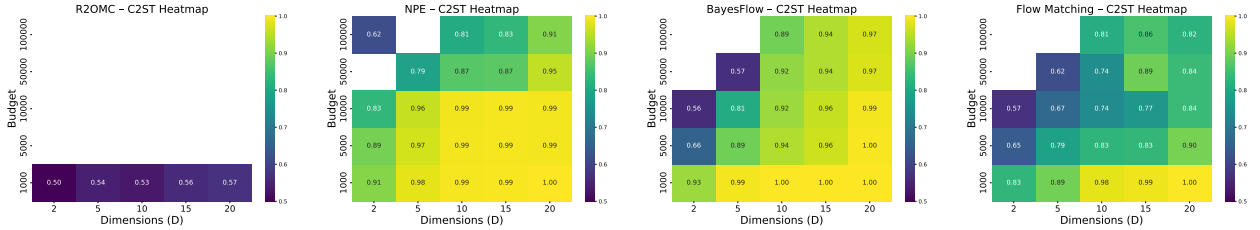


Figure 12: MoG Benchmark - MoG simulator with distractors: C2ST heatmaps for R2OMC, NPE, BayesFlow, and Flow Matching.

D.2 SBI - SLCP (T.3) and SLCP Distractors (T.4)

We here provide additional information on example T.3 and T.4 of the SBI benchmark Lueckmann et al. (2021).

SLCP (T.3):

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

Simulator $\mathbf{y} = (\boldsymbol{y_1}, \ldots, \boldsymbol{y_4}), \boldsymbol{y_i} \sim \mathcal{N}(\boldsymbol{m_\theta}, \boldsymbol{S_\theta}) \in \mathbb{R}^2$

where: $\boldsymbol{m_\theta} \sim \begin{bmatrix}\theta_1 \\ \theta_2\end{bmatrix}, \boldsymbol{S_\theta} = \begin{bmatrix}s_1 & \rho s_1 s_2 \\ \rho s_1 s_2 & s_2^2\end{bmatrix}$, where $s_1 = \theta_3^2$, $s_2 = \theta_4^2$ and $\rho = \tanh \theta_5$

Dimensionality $\boldsymbol{\theta} \in \mathbb{R}^5$, $\boldsymbol{y_i} \in \mathbb{R}^2$, $\mathbf{y} \in \mathbb{R}^8$

SLCP with distractors (T.4):

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

Simulator $\mathbf{y} = (\boldsymbol{y_1}, \ldots, \boldsymbol{y_4})$, $\boldsymbol{y_i} = (\boldsymbol{y_i^{(0)}}, \boldsymbol{y_i^{(1)}})$ where $\boldsymbol{y_i^{(0)}} \sim \mathcal{N}(\boldsymbol{m_\theta}, \boldsymbol{S_\theta})$

where: $\boldsymbol{m_\theta} \sim \begin{bmatrix}\theta_1 \\ \theta_2\end{bmatrix}, \boldsymbol{S_\theta} = \begin{bmatrix}s_1 & \rho s_1 s_2 \\ \rho s_1 s_2 & s_2^2\end{bmatrix}$, $s_1 = \theta_3^2$, $s_2 = \theta_4^2$ and $\rho = \tanh \theta_5$

and $\boldsymbol{y_i^{(1)}}$ comes from a distribution (independent of $\boldsymbol{\theta}$) analyzed in (Lueckmann et al., 2021).

Dimensionality $\boldsymbol{\theta} \in \mathbb{R}^5$, $\boldsymbol{y_i} \in \mathbb{R}^{100}$, $\boldsymbol{y_i} \in \mathbb{R}^{25}$, $\boldsymbol{y_i^{(0)}} \in \mathbb{R}^2$, $\boldsymbol{y_i^{(0)}} \in \mathbb{R}^{23}$

The ground truth posterior is not available in closed form. Instead a set of 10,000 posterior samples is used

D.2.1 Experimental Setup and Results

We evaluate R2OMC using simulation budgets (i.e., numbers of seeds) of 1,000, 1,500, 5,000, 10,000, and 30,000. The SBI benchmark provides 10,000 ground-truth posterior samples for each of eight distinct observations. We run R2OMC on all observations and report the mean `C2ST` score and mean runtime across them.

Figure 5 in the main paper presents the `C2ST`–runtime plots for both T.3 and T.4 tasks. To complement these, Figure 13 shows the `C2ST`–budget and runtime–budget relationships for the same tasks. These results confirm that the increase in runtime is primarily driven by the larger simulation budgets required for accurate inference.

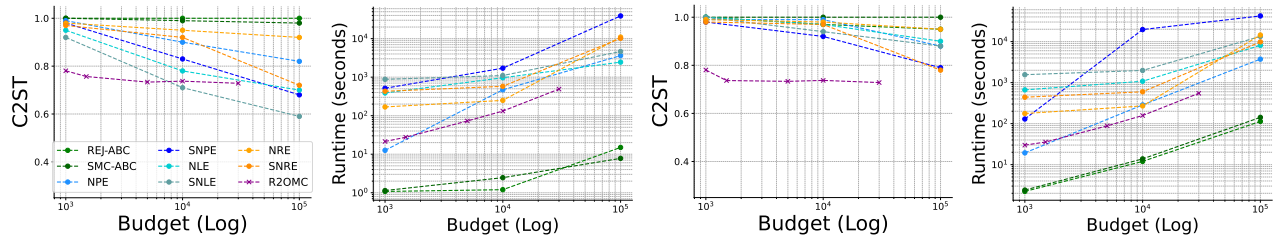


Figure 13: From left to right: `C2ST` vs. budget, runtime vs. budget. for T.3: SLCP and same figures for (T.4: SLCP Distractors)

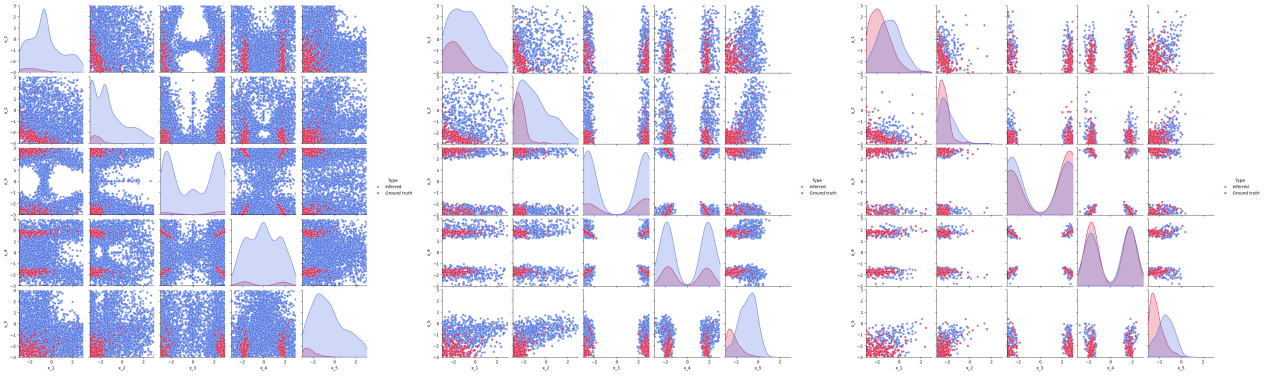


Figure 14: T.3: SLCP. From left to right: total samples, accepted samples, and selected samples by R2OMC.

Figure 4 in the main paper illustrates how R2OMC handles multiple observations. Proposal samples are first selected per observation (within an ϵ-distance) and then reweighted according to Eq. 14 to retain those most consistent across all observations. In Figure 14, we visualize all proposal samples from all observations (left panel), the subset that receives positive weights (middle panel), and the final accepted samples (right panel).

Overall, results demonstrate that R2OMC accurately approximates the posterior, achieving mean `C2ST` scores in the range of 0.7–0.8 within only a few seconds. In contrast, competing methods require substantially higher budgets (typically 10^4–10^5 simulations) and correspondingly longer runtimes—often several hours—to reach comparable performance. The performance gap becomes particularly pronounced in distractor settings, where most competing methods fail to achieve `C2ST` scores below 0.8 regardless of the simulation budget.

D.3 SBI - Two Moons (T.8)

We here provide additional information on example T.8 of the SBI benchmark Lueckmann et al. (2021). It tests inference when the posterior exhibits both global (bimodality) and local (crescent shape) structure to illustrate how algorithms deal with multimodality:

Prior $\quad \boldsymbol{\theta} \sim \mathcal{U}(-\mathbf{1}, \mathbf{1})$

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

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

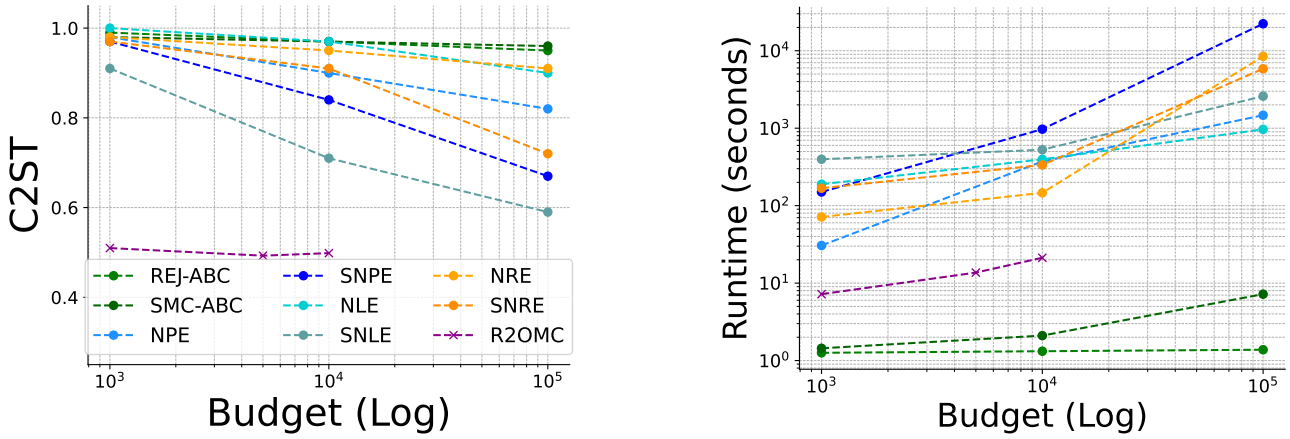
The ground truth posterior is not available in closed form. Instead a set of 10,000 posterior samples is used

D.3.1 Experimental Setup and Results

We test ROMC for simulation budget (number of seeds) equal to 1000, 5000, 10000. SBI benchamrk provides a set of 10,000 gound-truth posterior samples for each of the 8 different observations. We run ROMC in all observations and we report the mean C2ST score and the mean runtime across themm. Figure 6 in the main paper shows the C2ST vs. runtime plot and the pairwise posterior plot. To compliment that, in Figure 15 we

provide the C2T vs. budget plot and the Budget vs. Rutime. These plot confirm that the reason behind higher duntimes is the increase in the budget.

R2OMC achieves a near-optimal `C2ST`score (≈ 0.5) in a few-seconds runtime as 1000 different seeds S are already enough for a good posterior approximaiont. Competing methods require minutes to hours (budgets of 10^5 samples) to reach comparable performance (which never becomes that good) The pairwise posterior further demonstrates that R2OMC accurately captures both crescent-shaped modes.



(a) `C2ST` score

(b) Ground truth samples

Figure 15: T.8: Two Moons. From left to right: `C2ST` score, ground truth samples, and samples by R2OMC.