

Simulation-based Inference with the Python Package sbijax

Simon Dirmeier^{1,2}, Antonietta Mira^{3,4}, and Carlo Albert⁵

1 Swiss Data Science Center, Zurich, Switzerland 2 ETH Zurich, Zurich, Switzerland 3 Università della Svizzera italiana, Switzerland 4 University of Insubria, Italy 5 Swiss Federal Institute of Aquatic Science and Technology, Switzerland

DOI: [10.21105/joss.11195](https://doi.org/10.21105/joss.11195)

Software

- [Review](#) ↗
- [Repository](#) ↗
- [Archive](#) ↗

Editor: [Richard Littauer](#) ↗ 

Reviewers:

- [@aneeshnaik](#)
- [@dosvk](#)

Submitted: 19 March 2026

Published: 30 October 2026

License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](#)).

Summary

In natural sciences like astrophysics, biology or neuroscience, models can be frequently formulated as simulator functions, i.e., computer programs that stochastically map a parameter vector θ to synthetic data x . Even though simulators of this kind can represent realistic descriptions of real-world data-generating processes, they often emit intractable probability density functions. However, statistical inference of the parameters of a simulator requires the likelihood $p(x|\theta)$ to be tractable, such that inference algorithms like Markov chain Monte Carlo can be applied. Simulation-based inference (SBI) circumvents this by only requiring the ability to sample from the simulator: parameters θ are drawn from the prior distribution $p(\theta)$, an observable x is drawn from the simulator $p(x|\theta)$, and the resulting pairs are used to approximate the posterior distribution. In computational statistics, neural SBI describes an emerging family of methods for Bayesian inference for simulator models that use neural networks as surrogate models. Here we introduce sbijax, a Python package that implements a wide variety of state-of-the-art methods in neural SBI using a user-friendly programming interface. Targeted at domain scientists, e.g., in computational physics or computational biology, and SBI researchers, sbijax offers a compact functional interface to construct SBI estimators and compute posterior distributions with a few lines of code. In addition, the package provides functionality for conventional approximate Bayesian computation, to compute model diagnostics, and to automatically estimate summary statistics. By virtue of being entirely written in JAX, sbijax is extremely computationally efficient, allowing rapid training of neural networks and executing code automatically in parallel on both CPU and GPU.

Statement of Need

Modern approaches to neural SBI utilize recent developments in neural density estimation or score-based generative modelling to build surrogate models to approximate Bayesian posterior distributions. Similarly to conventional methods, such as approximate Bayesian computation (ABC, Sisson et al. (2018)) and its sequential (SMC-ABC; Sisson et al. (2007), Beaumont et al. (2009)) and annealing-based (e.g., SABC, Albert et al. (2015)) variants, neural SBI methods infer this posterior distribution by first simulating synthetic data and then numerically constructing an appropriate approximation to this pseudo data set. SBI methods are attractive for a couple of reasons. On the one hand, this family of methods has been shown to be more computationally efficient and often more accurate than ABC methods, in particular for smaller simulation budgets. On the other hand, SBI allows to easily amortize inference, i.e., to infer the posterior distribution for multiple different observations once a neural model has been trained.

Here we propose sbijax, a Python package implementing state-of-the-art methodology of neural simulation-based inference. While the main focus of the package is the implementation

of recent algorithms to make them available to practitioners, e.g., Albert et al. (2025) or Gloeckler et al. (2024), `sbijax` also implements common methods from approximate Bayesian computation, e.g., SMC-ABC, to have the entire SBI toolbox in one efficient package (see Table 1 for an overview). In addition, `sbijax` provides functionality for model diagnostics and Markov chain Monte Carlo (MCMC) sampling.

The package uses the high-performance computing framework JAX as a backend (Bradbury et al., 2018). Using JAX has several advantages, including a) that it uses the same syntax as `numpy` (Harris et al., 2020) which enables a seamless transition for applied scientists who already are familiar with it, and b) that empirical evaluations have shown that JAX can be significantly faster than PyTorch (see, e.g., Phan et al. (2019)). Our package heavily builds on libraries from the JAX-verse and common Bayesian inference tools. Specifically, we use Haiku (Hennigan et al., 2020) to construct and train neural networks, surjectors for normalizing flow based density estimation (Dirmeier, 2024), TensorFlow Probability (Dillon et al., 2017) to define statistical distributions, and BlackJAX (Cabezas et al., 2024) for posterior sampling using Markov chain Monte Carlo.

Table 1: Implemented SBI methods in `sbijax`.

Method	Function	Reference
Sequential Monte Carlo ABC	<code>smcab</code>	Beaumont et al. (2009)
Simulated annealing ABC	<code>sabc</code>	Albert et al. (2025)
Neural likelihood estimation	<code>nle</code>	Papamakarios et al. (2019)
Surjective neural likelihood estimation	<code>snle</code>	Dirmeier et al. (2025)
Neural posterior estimation	<code>npe</code>	Greenberg et al. (2019)
Contrastive neural ratio estimation	<code>nre</code>	Miller et al. (2022)
Flow matching posterior estimation	<code>fmpe</code>	Wildberger et al. (2023)
Neural posterior score estimation	<code>npse</code>	Sharrock et al. (2024)
All-in-one posterior estimation	<code>experimental.aio</code>	Gloeckler et al. (2024)
Consistency model posterior estimation	<code>experimental.cmpe</code>	Schmitt et al. (2024)
Neural approximate sufficient statistics	<code>nass</code>	Chen et al. (2021)
Neural approximate slice sufficient statistics	<code>nasss</code>	Chen et al. (2023)

State of the field

While a plethora of different models has been proposed in the recent literature, the development of adequate software packages has not followed at the same pace, and few packages exist that allow modelers to use these methods. Most prominently, the Python package `sbi` (Tejero-Cantero et al., 2020) implements several approaches for neural simulation-based inference, such as a neural posterior, likelihood-ratio, and likelihood estimation (Cranmer et al., 2020) utilizing a PyTorch backend (Paszke et al., 2019). The package additionally provides an API for model diagnostics, such as posterior predictive checks, effective sample size computations and simulation-based calibration. However, the package lacks implementations of recent developments which pose the state-of-the-art in the field, such as by Chen et al. (2023), Dirmeier et al. (2025) or Albert et al. (2025). BayesFlow (Radev et al., 2023) offers a wide variety of neural methods, but is similarly limited in the set of estimators it is implementing. For approximate Bayesian computation, several Python packages are available. In particular `abcpy` (Dutta et al., 2021) implements a multitude of different ABC algorithms. However, none of these packages implement modern (neural) SBI methods.

Software design

`sbijax` is designed as a modular and extensible toolbox for SBI in JAX, combining a functional programming philosophy with a flexible interface for both expert users and domain scientists. Its design is guided by the following principles:

- 1) Faithful alignment with JAX's functional paradigm. `Sbijax` implements a fully functional API in the idiom of Haiku: every method is a factory function that returns a tuple of pure functions. All non-permanent state variables, i.e., neural network parameters and optimizer states, are passed explicitly instead of being held by an object. A neural estimator, e.g., `nle(make_maf(.))`, returns training primitives and a sampling function which are consumed by the generic drivers `train` and `sample`, while an ABC method, e.g., `sabc(prior, simulator)`, returns a sampler that draws from the approximate posterior directly. No function mutates internal state facilitating composability, and ensures compatibility with JAX transformations such as `jit`, `vmap`, and `pmap`.
- 2) Separation of inference methods and neural network implementations. `sbijax` focuses on implementing SBI algorithms, while neural network components are defined using Haiku. This allows users to construct models directly with Haiku, and to incorporate probabilistic building blocks from Distrax (DeepMind et al., 2020) and surjectors (Dirmeier, 2024). As a result, model definitions remain flexible and low-level, while seamlessly interoperating with the broader JAX ecosystem.
- 3) A deliberately low-level interface. An inference workflow typically written as a short sequence of ordinary function calls, where the inputs and outputs of every step, e.g., the simulated data set and the network parameters, are managed by the developer explicitly. This asks developers to be deliberate about the workflow they are executing rather than calling the member functions of an object whose internal state is (often) opaque to them. We consider this trade-off worthwhile for a scientific software tool, since it makes a method transparent to the practitioner using it (although it induces some additional complexity) and straightforward to modify for a researcher extending it.
- 4) Support for extensibility and research. `Sbijax` is structured to facilitate experimentation with new SBI methods. Its modular design allows components such as neural architectures, training objectives, and sampling strategies to be easily replaced or extended, making it suitable as both a research framework and a practical toolbox. Since every method reduces to the same small set of primitives, namely an initialization, a gradient step, an evaluation step and a sampling function, a newly implemented objective can be trained and sampled from with the existing drivers without any further changes.
- 5) Accessibility for domain scientists. In addition to its flexibility, `sbijax` includes pre-implemented models with sensible defaults, enabling use without in-depth expertise in deep learning. In the simplest case, users only need to define a prior and a simulator to run inference workflows, often in as few as five lines of code.

Research impact statement

`Sbijax` has already been used extensively in the Machine Learning literature. Dirmeier et al. (2025) and Dirmeier & Mira (2025) proposed novel SBI methods for surjective neural likelihood estimation and posterior estimation using causal constraints, respectively, where they used `sbijax` heavily for their experimental evaluations. Ulzega et al. (2025) used `sbijax` to infer the posterior distribution of a complicated Bayesian model from the astrophysics literature. Albert et al. (2025) developed a novel ABC method that uses `sbijax` for model evaluation.

AI usage disclosure

Claude Code was used to assist with parts of the software, in particular bug fixes, tests, and documentation. All AI-assisted changes were reviewed by the authors, who take full responsibility for the software and the manuscript.

Acknowledgements

This research was supported by the Swiss National Science Foundation (Grant No. 200021_208249).

References

- Albert, C., Künsch, H. R., & Scheidegger, A. (2015). A simulated annealing approach to approximate Bayes computations. *Statistics and Computing*, 25, 1217–1232. <https://doi.org/10.1007/s11222-014-9507-8>
- Albert, C., Ulzega, S., Dirmeier, S., Scheidegger, A., Bassi, A., & Mira, A. (2025). Simulated annealing ABC with multiple summary statistics. *arXiv Preprint arXiv:2505.23261*.
- Beaumont, M. A., Cornuet, J.-M., Marin, J.-M., & Robert, C. P. (2009). Adaptive approximate Bayesian computation. *Biometrika*, 96(4), 983–990.
- Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., & Zhang, Q. (2018). JAX: Composable transformations of Python+NumPy programs (Version 0.3.13). <http://github.com/google/jax>
- Cabezas, A., Corenflos, A., Lao, J., & Louf, R. (2024). BlackJAX: Composable Bayesian inference in JAX. *arXiv Preprint arXiv:2402.10797*.
- Chen, Y., Gutmann, M. U., & Weller, A. (2023). Is learning summary statistics necessary for likelihood-free inference? *Proceedings of the 40th International Conference on Machine Learning*.
- Chen, Y., Zhang, D., Gutmann, M. U., Courville, A., & Zhu, Z. (2021). Neural approximate sufficient statistics for implicit models. *International Conference on Learning Representations*.
- Cranmer, K., Brehmer, J., & Louppe, G. (2020). The frontier of simulation-based inference. *Proceedings of the National Academy of Sciences*, 117(48), 30055–30062.
- DeepMind, Babuschkin, I., Baumli, K., Bell, A., Bhupatiraju, S., Bruce, J., Buchlovsky, P., Budden, D., Cai, T., Clark, A., Danihelka, I., Dedieu, A., Fantacci, C., Godwin, J., Jones, C., Hemsley, R., Hennigan, T., Hessel, M., Hou, S., ... Viola, F. (2020). *The DeepMind JAX Ecosystem*. <http://github.com/deepmind>
- Dillon, J. V., Langmore, I., Tran, D., Brevdo, E., Vasudevan, S., Moore, D., Patton, B., Alemi, A., Hoffman, M., & Saurous, R. A. (2017). TensorFlow Distributions. *arXiv Preprint arXiv:1711.10604*.
- Dirmeier, S. (2024). Surjectors: Surjection layers for density estimation with normalizing flows. *Journal of Open Source Software*, 9(94), 6188. <https://doi.org/10.21105/joss.06188>
- Dirmeier, S., Albert, C., & Perez-Cruz, F. (2025). Simulation-based inference for high-dimensional data using surjective sequential neural likelihood estimation. *The 41st Conference on Uncertainty in Artificial Intelligence*.
- Dirmeier, S., & Mira, A. (2025). Causal posterior estimation. *arXiv Preprint arXiv:2505.21468*.

- Dutta, R., Schoengens, M., Pacchiardi, L., Ummadisingu, A., Widmer, N., Künzli, P., Onnela, J.-P., & Mira, A. (2021). ABCpy: A high-performance computing perspective to approximate Bayesian computation. *Journal of Statistical Software*, 100(7), 1–38.
- Gloeckler, M., Deistler, M., Weilbach, C. D., Wood, F., & Macke, J. H. (2024). All-in-one simulation-based inference. *Proceedings of the 41st International Conference on Machine Learning*.
- Greenberg, D., Nonnenmacher, M., & Macke, J. (2019). Automatic posterior transformation for likelihood-free inference. *Proceedings of the 36th International Conference on Machine Learning*.
- Harris, C. R., Millman, K. J., Walt, S. J. van der, Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., & others. (2020). Array programming with NumPy. *Nature*, 585(7825), 357–362.
- Hennigan, T., Cai, T., Norman, T., Martens, L., & Babuschkin, I. (2020). *Haiku: Sonnet for JAX* (Version 0.0.10). <http://github.com/deepmind/dm-haiku>
- Miller, B. K., Weniger, C., & Forré, P. (2022). Contrastive neural ratio estimation. *Advances in Neural Information Processing Systems*. <https://doi.org/10.52202/068431-0236>
- Papamakarios, G., Sterratt, D., & Murray, I. (2019). Sequential neural likelihood: Fast likelihood-free inference with autoregressive flows. *Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics*.
- Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., ... Chintala, S. (2019). PyTorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*.
- Phan, D., Pradhan, N., & Jankowiak, M. (2019). Composable effects for flexible and accelerated probabilistic programming in NumPyro. *arXiv Preprint arXiv:1912.11554*.
- Radev, S. T., Schmitt, M., Schumacher, L., Elsemüller, L., Pratz, V., Schälte, Y., Köthe, U., & Bürkner, P.-C. (2023). BayesFlow: Amortized Bayesian workflows with neural networks. *Journal of Open Source Software*, 8(89), 5702. <https://doi.org/10.21105/joss.05702>
- Schmitt, M., Pratz, V., Köthe, U., Bürkner, P.-C., & Radev, S. T. (2024). Consistency models for scalable and fast simulation-based inference. *Advances in Neural Information Processing Systems*. <https://doi.org/10.52202/079017-4030>
- Sharrock, L., Simons, J., Liu, S., & Beaumont, M. (2024). Sequential neural score estimation: Likelihood-free inference with conditional score based diffusion models. *Forty-First International Conference on Machine Learning*.
- Sisson, S. A., Fan, Y., & Beaumont, M. (2018). *Handbook of approximate Bayesian computation*. CRC Press.
- Sisson, S. A., Fan, Y., & Tanaka, M. M. (2007). Sequential Monte Carlo without likelihoods. *Proceedings of the National Academy of Sciences*, 104(6), 1760–1765.
- Tejero-Cantero, A., Boelts, J., Deistler, M., Lueckmann, J.-M., Durkan, C., Gonçalves, P. J., Greenberg, D. S., & Macke, J. H. (2020). sbi: A toolkit for simulation-based inference. *Journal of Open Source Software*, 5(52), 2505. <https://doi.org/10.21105/joss.02505>
- Ulzega, S., Beer, J., Ferriz-Mas, A., Dirmeier, S., & Albert, C. (2025). Shedding light on the solar dynamo using data-driven Bayesian parameter inference. *The Astrophysical Journal*, 992(1), 61. <https://doi.org/10.3847/1538-4357/adfec3>
- Wildberger, J. B., Dax, M., Buchholz, S., Green, S. R., Macke, J. H., & Schölkopf, B. (2023). Flow matching for scalable simulation-based inference. *Advances in Neural Information*

Processing Systems. <https://doi.org/10.52202/075280-0737>