The Journal of Open Source Software

AutoEmulate: A PyTorch tool for end-to-end emulation workflows

Radka Jersakova [1*¶], **Sam F. Greenbury** [1*], **Ed Chalstrey** [1], **Edwin Brown** [2], **Marjan Famili** [1], **Chris Sprague** [1], **Paolo Conti** [1], **Camila Rangel Smith** [1], **Martin A. Stoffel** [1], **Bryan M. Li** [1,3], **Kalle Westerling** [1], **Sophie Arana** [1], **Max Balmus** [1,4], **Eric Daub** [1], **Steve Niederer** [1,4], **Andrew B. Duncan** [4], **and Jason D. McEwen** [1,5]

1 The Alan Turing Institute, London, United Kingdom **2** University of Sheffield, Sheffield, United Kingdom **3** University of Edinburgh, Edinburgh, United Kingdom **4** Imperial College London, London, United Kingdom **5** University College London, London, United Kingdom ¶ Corresponding author * These authors contributed equally.

DOI: 10.21105/joss.10087

Software

- Review
- Repository
- Archive

Editor: Evan Spotte-Smith

Reviewers:

- @stefanradev93
- @pmeier

Submitted: 25 September 2025
Published: 26 August 2026

Summary

Computational simulations lie at the heart of modern science and engineering, but they are often slow and computationally costly. A common solution is to use emulators: fast, cheap models trained to approximate the simulator. However, constructing these requires substantial expertise. AutoEmulate (Stoffel et al., 2025) is a Python package for end-to-end emulation workflows that automates much of the machine learning pipeline, making it easy to replace simulations with fast, accurate emulators. AutoEmulate has now been fully refactored to use PyTorch as the primary and default backend, enabling GPU acceleration, automatic differentiation, and seamless integration with the broader PyTorch ecosystem. The toolkit has also been extended with easy-to-use interfaces for common emulation tasks, including model calibration (determining which input values are most likely to have generated real-world observations) and active learning (where simulations are chosen to improve emulator performance at minimal computational cost). Together these updates make AutoEmulate uniquely suited to running performant end-to-end emulation workflows.

Statement of need

Physical systems are often modelled using computer simulations, which can be computationally expensive and time-consuming to run. This computational bottleneck can be resolved by approximating simulations with emulators, often leading to significant speed-ups (Kennedy & O'Hagan, 2000).

Emulation requires significant expertise in machine learning as well as familiarity with a broad and evolving ecosystem of tools. This creates a barrier to entry for domain researchers whose focus is on the underlying scientific problem. AutoEmulate (Stoffel et al., 2025) lowers the barrier to entry by automating the entire emulator construction process (training, hyperparameter tuning and model selection). This makes emulation accessible to non-specialists while also offering a reference set of emulators for benchmarking to experienced users.

AutoEmulate was originally built on scikit-learn, which is well suited for traditional machine learning but less flexible for complex workflows. AutoEmulate v2.0 introduced a PyTorch (Paszke et al., 2019) backend that provides GPU acceleration for faster training and inference and automatic differentiation via PyTorch's autograd system. The PyTorch refactor also

enabled fast Bayesian model calibration (identifying input values most likely to have generated real-world observations) using gradient-based inference methods such as Hamiltonian Monte Carlo implemented through Pyro (Bingham et al., 2018).

The latest version of AutoEmulate now also supports direct integration of custom simulators and active learning, in which the tool adaptively selects informative simulations to run to improve emulator performance at minimal computational cost. Additionally, the AutoEmulate refactor improved support for high-dimensional data through dimensionality reduction techniques such as principal component analysis (PCA) and variational autoencoders (VAEs).

State of the field

The primary motivation for developing AutoEmulate (Stoffel et al., 2025) was to make emulation accessible to domain researchers who are not machine learning experts. It achieves this through a simple, high-level interface and automated model selection, distinguishing it from other surrogate modelling tools such as SMT (Saves et al., 2024) or Emukit (Paleyes et al., 2023).

We further extended the capabilities of AutoEmulate to include a range of downstream tasks in a single package. The comprehensive nature of the tool distinguishes it from more specialised packages. For example, SBI (Boelts et al., 2025) or BayesFlow (Kühmichel et al., 2026) can train emulators on simulated data and perform Bayesian calibration, while Ax (Olson et al., 2025) or scikit-activeml (Herde et al., 2025) support active learning. While these packages can offer more advanced tooling for a particular task, none of them cover as wide a range of emulation capabilities.

More comprehensive uncertainty quantification toolkits such as OpenTURNS (Baudin et al., 2015) and PyApprox (Jakeman, 2023) do offer broad emulation functionality, though automated comparison across emulator classes is left to the user. To our knowledge, no existing tool combines automated model selection and hyperparameter tuning with sensitivity analysis, calibration, and active learning in a single package.

Software Design

AutoEmulate's design is centered around (i) a high-level API that automates common workflows, (ii) modularity and (iii) integrating with the wider ecosystem wherever possible. The design has now been updated from being entirely scikit-learn oriented to PyTorch-first.

AutoEmulate primarily targets users who are simulation but not ML experts, aiming to make emulator training as easy as possible. We also offer flexibility to advanced users by exposing customizable parameters through our APIs (set to sensible defaults to abstract complexity away from novice users).

The software's modular design is built on base classes for each component, enabling users to easily add new emulators and functionality. We chose PyTorch as the backend because of its autodiff and GPU capabilities as well as the mature ecosystem that we could integrate with. For example, both GPyTorch (Gardner et al., 2021) and Pyro (Bingham et al., 2018) are extensively utilised within the package.

Example usage

The AutoEmulate documentation provides a comprehensive set of tutorials showcasing all functionality. We are also collecting case studies demonstrating how to use AutoEmulate for real-world problems and complex workflows. Below we provide a brief overview of the main features.

The core use case for AutoEmulate is emulator construction. AutoEmulate takes as input variables x, y. The variable x is a 2D array with columns corresponding to simulation parameters and rows corresponding to parameter sets. The variable y is an array of one or more simulation outputs corresponding to each set of parameters. From this data, AutoEmulate constructs an emulator in just a few lines of code:

```
from autoemulate import AutoEmulate

ae = AutoEmulate(x, y)

result = ae.best_result()

emulator = result.model
```

This simple script runs a search over a library of emulator models, performs hyperparameter tuning and compares models using cross validation. Each model is stored along with metadata in a `Results` object. The user can then easily extract the best performing emulator.

AutoEmulate can additionally search over different data preprocessing methods, such as normalization or dimensionality reduction techniques (PCA, VAEs). Any `Transform` from PyTorch distributions can also be used. The transforms are passed as a list to permit the user to define a sequence of transforms to apply to the data. For example, the following code standardizes the input data and compares three different output transformations: no transformation, PCA with 16 components, and PCA with 32 components in combination with the default set of emulators:

```
from autoemulate.transforms import PCATransform, StandardizeTransform

ae = AutoEmulate(
    x,
    y,
    x_transforms_list=[[StandardizeTransform]],
    y_transforms_list=[
      [],
      [PCATransform(n_components=16)],
      [PCATransform(n_components=32)]
    ],
)
```

The result in this case will return the best combination of model and output transform. The returned emulator and transforms are wrapped together in a `TransformedEmulator` class, which outputs predictions in the original data space. The figure below shows an example result of fitting a Gaussian Process emulator in combination with PCA to a reaction-diffusion simulation (see the full tutorial for a detailed overview).

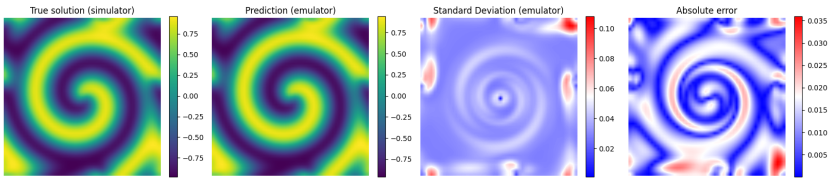

Figure 1: GP with PCA emulator prediction for a reaction diffusion simulation compared to the ground truth.

Once an emulator has been trained it can generate fast predictions for new input values,

enabling downstream tasks such as sensitivity analysis or model calibration. For example, to run Sobol sensitivity analysis one only needs to pass the trained emulator and some information about the data. Below is a minimal example assuming a simulation with two input parameters `param1` and `param2`, each with a plausible range of values, and two outputs `output1` and `output2`:

```
from autoemulate.core.sensitivity_analysis import SensitivityAnalysis

input_parameters_ranges = {
    'param1': (0, 1),
    'param2': (0, 10),
}

problem = {
    'num_vars': 2,
    'names': ["param1", "param2"],
    'bounds': input_parameters_ranges.values(),
    'output_names': ["output1", "output2"],
}

sa = SensitivityAnalysis(emulator, problem=problem)
sobol_df = sa.run()
```

AutoEmulate also provides a simple interface for calibration given a trained emulator, input parameter ranges (same as in the sensitivity analysis example), and real-world observations:

```
from autoemulate.calibration.bayes import BayesianCalibration

observations = {'output1': 0.5, 'output2': 7.2}

bc = BayesianCalibration(
    emulator,
    input_parameters_ranges,
    observations,
)
mcmc = bc.run()
```

Lastly, AutoEmulate makes it easy to integrate custom simulators through subclassing. This enables simulator-in-the-loop workflows like active learning, which selects the most informative simulations to improve emulator performance at minimal computational cost.

Research Impact Statement

In the last year, we have collaborated with experts across diverse domains. These collaborations have driven software development through feature requests and bug reports. For example, we implemented a full end-to-end calibration workflow used by collaborators in cardiac modelling, which utilises sensitivity analysis, history matching and Bayesian calibration. We demonstrate how to use AutoEmulate in patient calibration pipelines in one of our case studies. Similarly, we have worked with domain experts in signal processing, which led to the implementation of interval excursion set calibration. This work was presented in a poster at the Operating in the Future Electromagnetic Environment (OFEME) Symposium 2025. We have also had contributors outside the core development team responding to existing issues and adapting the tool for their own use cases (e.g., contributing new types of emulators).

AI usage disclosure

Human authors have made all the core design decisions and authored much of the code and documentation. Generative AI tools have been used to assist with code and documentation writing. Specifically, the team uses GitHub Copilot in auto mode or selects one of the available versions of Claude, GPT and Gemini. We confirm that human authors have reviewed, edited and validated all AI-assisted outputs. We have also added a section on the use of generative AI tools in our contributing guidelines.

References

Baudin, M., Dutfoy, A., Iooss, B., & Popelin, A.-L. (2015). OpenTURNS: An industrial software for uncertainty quantification in simulation. In *Handbook of uncertainty quantification* (pp. 1–38). Springer International Publishing. https://doi.org/10.1007/978-3-319-11259-6_64-1

Bingham, E., Chen, J. P., Jankowiak, M., Obermeyer, F., Pradhan, N., Karaletsos, T., Singh, R., Szerlip, P. A., Horsfall, P., & Goodman, N. D. (2018). Pyro: Deep universal probabilistic programming. *CoRR*, *abs/1810.09538*. https://doi.org/10.48550/arXiv.1810.09538

Boelts, J., Deistler, M., Gloeckler, M., Tejero-Cantero, Á., Lueckmann, J.-M., Moss, G., Steinbach, P., Moreau, T., Muratore, F., Linhart, J., Durkan, C., Vetter, J., Miller, B. K., Herold, M., Ziaeemehr, A., Pals, M., Gruner, T., Bischoff, S., Krouglova, N., … Macke, J. H. (2025). Sbi reloaded: A toolkit for simulation-based inference workflows. *Journal of Open Source Software*, *10*(108), 7754. https://doi.org/10.21105/joss.07754

Gardner, J. R., Pleiss, G., Bindel, D., Weinberger, K. Q., & Wilson, A. G. (2021). *GPyTorch: Blackbox matrix-matrix Gaussian process inference with GPU acceleration*. https://doi.org/10.48550/arXiv.1809.11165

Herde, M., Pham, M. T., Kottke, D., Benz, A., Lührs, L., Mergard, P., Sandrock, C., Cheng, J., Roghman, A., Müjde, M., Rauch, L., & Sick, B. (2025). scikit-activeml: A Comprehensive and User-friendly Active Learning Library. *Preprints*. https://doi.org/10.20944/preprints202507.0252.v1

Jakeman, J. D. (2023). PyApprox: A software package for sensitivity analysis, Bayesian inference, optimal experimental design, and multi-fidelity uncertainty quantification and surrogate modeling. *Environmental Modelling & Software*, *170*, 105825. https://doi.org/10.1016/j.envsoft.2023.105825

Kennedy, M. C., & O'Hagan, A. (2000). Predicting the output from a complex computer code when fast approximations are available. *Biometrika*, *87*(1), 1–13. https://doi.org/10.1093/biomet/87.1.1

Kühmichel, L., Huang, J. M., Pratz, V., Arruda, J., Olischläger, H., Habermann, D., Kucharsky, S., Elsemüller, L., Mishra, A., Bracher, N., Jedhoff, S., Schmitt, M., Bürkner, P.-C., & Radev, S. T. (2026). BayesFlow 2: Multi-backend amortized Bayesian inference in Python. *arXiv Preprint arXiv:2602.07098*. https://doi.org/10.48550/arXiv.2602.07098

Olson, M., Santorella, E., Tiao, L. C., Cakmak, S., Garrard, M., Daulton, S., Lin, Z. J., Ament, S., Beckerman, B., Onofrey, E., Igusti, P., Lara, C., Letham, B., Cardoso, C., Shen, S. S., Lin, A. C., Grange, M., Kashtelyan, E., Eriksson, D., … Bakshy, E. (2025). Ax: A platform for adaptive experimentation. In L. Akoglu, C. Doerr, J. N. van Rijn, R. Garnett, & J. R. Gardner (Eds.), *Proceedings of the fourth international conference on automated machine learning* (Vol. 293, pp. 21/1–25). PMLR. https://proceedings.mlr.press/v293/olson25a.html

Paleyes, A., Mahsereci, M., & Lawrence, N. D. (2023). Emukit: A Python toolkit for decision making under uncertainty. *Proceedings of the Python in Science Conference*.

https://doi.org/10.25080/gerudo-f2bc6f59-009

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E. Z., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., … Chintala, S. (2019). PyTorch: An imperative style, high-performance deep learning library. *CoRR*, *abs/1912.01703*. https://doi.org/10.48550/arXiv.1912.01703

Saves, P., Lafage, R., Bartoli, N., Diouane, Y., Bussemaker, J., Lefebvre, T., Hwang, J. T., Morlier, J., & Martins, J. R. R. A. (2024). SMT 2.0: A surrogate modeling toolbox with a focus on hierarchical and mixed variables Gaussian processes. *Advances in Engineering Software*, *188*, 103571. https://doi.org/10.1016/j.advengsoft.2023.103571

Stoffel, M. A., Li, B. M., Westerling, K., Arana, S., Balmus, M., Daub, E., & Niederer, S. (2025). AutoEmulate: A Python package for semi-automated emulation. *Journal of Open Source Software*, *10*(107), 7626. https://doi.org/10.21105/joss.07626