The Journal of Open Source Software

OpenReservoirComputing: GPU-Accelerated Reservoir Computing in JAX

Jan P. Williams [1¶], **Dima Tretiak** [1], **Steven L. Brunton** [1], **J. Nathan Kutz** [2,3,4], **and Krithika Manohar** [1]

1 Department of Mechanical Engineering, University of Washington, USA **2** Department of Applied Mathematics, University of Washington, USA **3** Department of Electrical and Computer Engineering, University of Washington, USA **4** Autodesk Research, London, UK ¶ Corresponding author

DOI: 10.21105/joss.10486

Software

- Review
- Repository
- Archive

Editor: Neea Rusch

Reviewers:

- @mbsuraj
- @CDJellen
- @ymahlau

Submitted: 03 March 2026
Published: 06 July 2026

Summary

OpenReservoirComputing (ORC) is a Python library for reservoir computing (RC) built on JAX and Equinox. RC is a form of machine learning that functions by lifting a low-dimensional sequence or signal into a high-dimensional dynamical system and training a simple, linear readout layer from the high-dimensional dynamics back to a lower-dimensional quantity of interest. The most common application of RC is time-series forecasting, where the goal is to predict a signal's future evolution. RC has achieved state-of-the-art performance on this task, particularly when applied to chaotic dynamical systems. RC can also perform classification and control tasks. ORC provides modular components for custom RC models and built-in models for forecasting, classification, and control. By building on JAX and Equinox, ORC offers GPU acceleration, JIT compilation, and automatic vectorization. These capabilities make prototyping new models faster, enable larger reservoir architectures, and allow seamless integration with other deep learning models.

Statement of Need

Time-series prediction, classification, and control are fundamental tasks across science and engineering, arising in applications from climate modeling and fluid dynamics to robotics and neuroscience. Deep learning approaches to these tasks typically require large datasets, long training times, and expensive tuning of optimization hyperparameters. RC offers a compelling alternative. Since only the readout layer is trained via a single ridge regression, RC models can be trained in a fraction of the time required by comparable recurrent neural networks, often with less data and fewer hyperparameters to tune (Lukoševičius & Jaeger, 2009; Pathak et al., 2018; Wyder et al., 2025). This makes RC particularly attractive for rapid prototyping, real-time applications, and data-limited settings. However, realizing these benefits in practice requires software that is both efficient and adaptable.

ORC's built-in models provide an easy entry point for users new to the field. In particular, a new user can supply their own time-series data to instantiate, train, and forecast in three simple lines of code.

```
import orc
U_train = ...
esn = orc.forecaster.ESNForecaster(data_dim=3, res_dim=400)
esn, R = orc.forecaster.train_RCForecaster(esn, U_train)
U_pred = esn.forecast(fcast_len=1000, res_state=R[-1])
```

Built-in visualization tools make it easy to evaluate model performance. Varying the

hyperparameters of built-in models lets users explore how RC performance depends on configuration choices. ORC's JAX foundation makes scaling to higher-dimensional parallel reservoir architectures equally simple.

Much RC research is aimed at designing performant reservoir architectures. ORC makes this easy through its use of abstract base classes for Embedding, Driver, and Readout layers. Users need only define forward pass logic to integrate a new reservoir topology or readout strategy, while reusing the rest of the framework. This modular approach also enables ablation studies on how different components affect RC performance. Because of ORC's functional approach in JAX, built-in and user-created models provide end-to-end differentiability by default. This enables gradient-based optimization of input sequences for control problems. This also makes ORC well suited to integrate with deep learning models such as those presented by Özalp et al. (2023, 2025).

State of the Field

	ORC	**ReservoirPy**	**RC.jl**
Language	Python	Python	Julia
GPU	✓	✓*	✓
Auto. Differentiable	✓	×	✓
Parallelizable	✓	×	✓
Vectorizable	✓	×	✓
Forecasting	✓	✓	✓
Classification	✓	✓	✓
Control	✓	×	×
Continuous Time	✓	×	×

Table 1: Comparison of reservoir computing libraries across key features. ✓ indicates full support; × indicates no support. *ReservoirPy's GPU support is available via its JAX backend (v0.4.0+) but does not fully exploit JAX's functional programming model. *Parallelizable* denotes native support for parallel RC architectures as in (Pathak et al., 2018) and *vectorizable* denotes native support for vectorization (e.g. `vmap`).

The most commonly used open-source library for reservoir computing is ReservoirPy (Trouvain et al., 2020). Like ORC, ReservoirPy provides built-in architectures and an API for custom layers. ReservoirPy was initially built on NumPy and SciPy with the maintainers adding a JAX backend in v0.4.0. However, ORC differs from ReservoirPy in several important ways.

First, ORC was *designed* on top of JAX (Bradbury et al., 2018) and Equinox (Kidger & Garcia, 2021), which provide a different programming model based on functional transformations. This enables native GPU/TPU acceleration, JIT compilation, and composable transformations (`jit`, `vmap`, `grad`) that cannot be retrofitted into a NumPy-based architecture. While the JAX backend of ReservoirPy does improve performance, the API cannot fully exploit JAX's capabilities. For example, ORC's autoregressive forecast loop uses `jax.lax.scan`, avoiding Python overhead; ReservoirPy's object-oriented design makes this impossible. Second, ORC has a different built-in feature set. ORC supports continuous-time reservoir dynamics via Diffrax (Kidger, 2021), allowing users to define reservoir equations as ordinary differential equations solved with adaptive-step integrators. ORC also supports novel architectures such as Taylor-expanded and GRU-based drivers alongside standard echo state networks. ReservoirPy does not provide these functionalities. Third, ORC models compose seamlessly with other deep learning models in Equinox, whereas ReservoirPy is suited to standalone RC tasks.

Other open source libraries for RC include PyTorch-ESN (Nardo, 2018) and ReservoirComputing.jl (Martinuzzi et al., 2022). PyTorch-ESN allows for integration with other PyTorch models,

but is not as widely adopted and offers a much more limited feature set than ORC or ReservoirPy. For instance, it does not support sparse matrix operations or parallel reservoirs. ReservoirComputing.jl is a widely used Julia library for RC that influenced many of ORC's design choices, but lacks Python ML ecosystem integration. Table 1 summarizes the functionality of ORC, ReservoirPy, and ReservoirComputing.jl.

Software Design

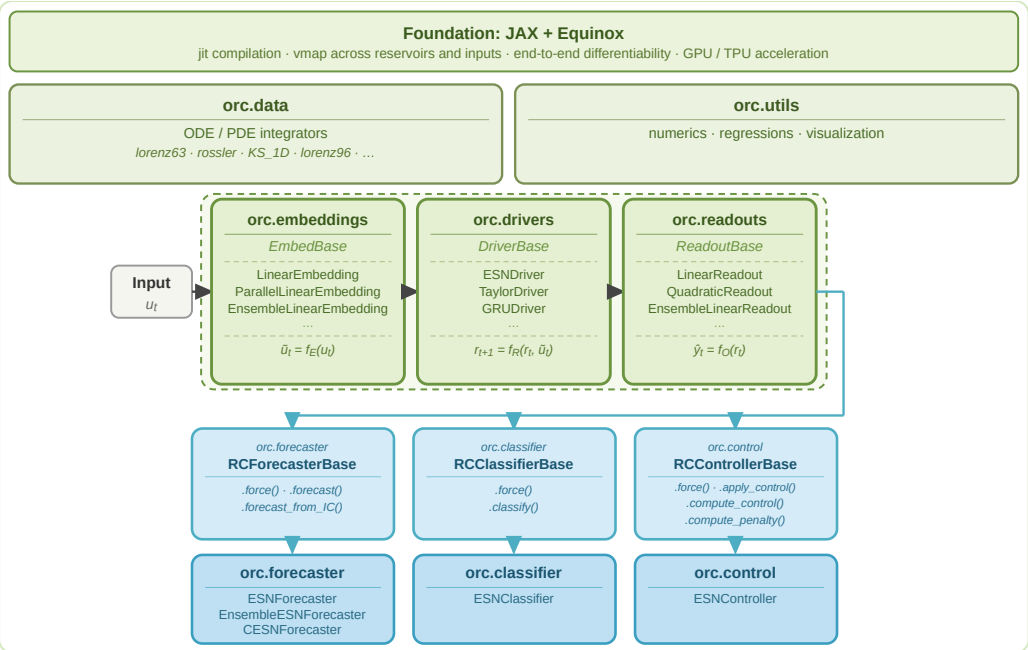

Figure 1: ORC three-layer pipeline architecture. Each reservoir computer (RC) is decomposed into (i) an embedding function that lifts a low-dimensional signal u_t to the reservoir dimension, (ii) a driver function that propagates the reservoir state, and (iii) a readout that maps back to a target y. For control and forecasting RCs, the target y is typically u_{t+1} (either in the presence of a forcing term or not) and for classification the target y is a label.

ORC models are decomposed into three components, illustrated in Figure 1: (i) an embedding f_E that lifts a low-dimensional input signal u_t to a high-dimensional space, (ii) a driver f_R that propagates the high-dimensional state r_t, and (iii) a readout f_O that maps the latent state back to an approximation of some low-dimensional signal y_t. Depending on the task at hand, y_t may be a future time-step of u_t, a label associated with input data, or some other target signal. ORC differs from many existing approaches that unify (i) and (ii). Separating the embedding from the reservoir state propagation allows for cleaner application of RC to non-standard tasks, such as acting as a surrogate model for model predictive control.

All components are implemented as Equinox modules (Kidger & Garcia, 2021), which are immutable pytree-registered objects. Model parameters (reservoir weights, readout matrices) are stored as JAX arrays within the module, and parameter updates produce new module instances via `eqx.tree_at` rather than mutating state in place. This functional design enables JAX's composable transformations to operate directly on model objects and allows ORC models to be composed with other Equinox modules.

ORC supports parallel reservoirs (Pathak et al., 2018) by default via a `chunks` parameter in each module, enabling spatiotemporal RC methods unavailable in other libraries. This extra

tensor dimension also allows for simple batching during training, avoiding excessive GPU VRAM requirements.

ORC provides unified training functions (`train_RCForecaster`, `train_RCClassifier`, `train_RCController`) that work with any model inheriting from the corresponding base class, including user-defined models with custom components. These functions delegate shape handling to the readout layer, allowing the same training function to handle both discrete and continuous-time models.

The library provides three built-in model classes: `ESNForecaster` for time-series prediction, `ESNClassifier` for sequence classification, and `ESNController` for learning control policies with exogenous control inputs. Each composes embedding, driver, and readout components and provides task-specific methods (`forecast`, `classify`, `apply_control`). Users who need custom architectures can subclass the abstract base classes, define only the components that differ, and immediately use the unified training functions without reimplementing teacher forcing, autoregressive prediction, or ridge regression.

ORC also includes a data generation module with ODE and PDE integrators for standard benchmark systems, including the Lorenz-63 attractor, Rössler system, double pendulum, Lorenz-96 model, and the Kuramoto-Sivashinsky equation, all implemented using Diffrax.

Research Impact Statement

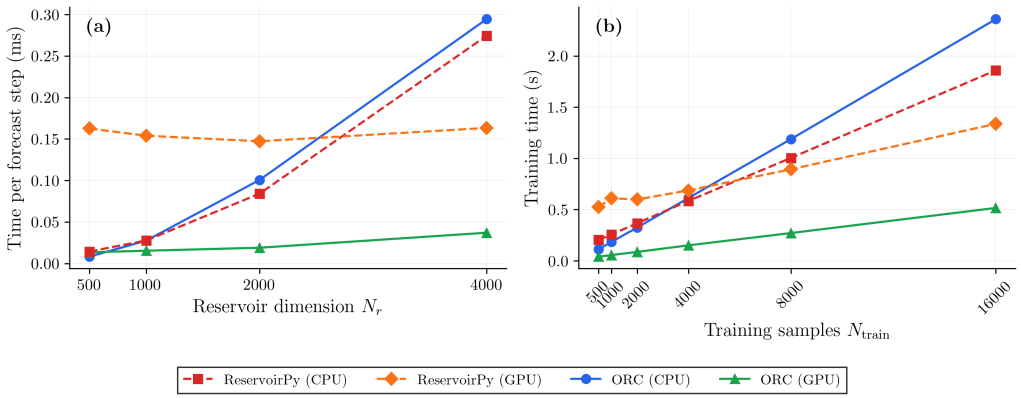

Figure 2: ORC with GPU acceleration enables significantly faster performance than ReservoirPy, even when using ReservoirPy's JAX backend. Panel (a) shows the time per forecast step of RC models with varying reservoir dimension trained to forecast the Lorenz system, while panel (b) shows training time for RC models with fixed reservoir dimension of 2000 but a varying number of training samples. Performance of the two libraries with and without GPU acceleration are shown. GPU results were obtained on an NVIDIA A40 GPU and CPU results were obtained with an Apple M2 chip.

ORC enables easy reimplementation of architectures that integrate RC with larger neural networks (Özalp et al., 2023, 2025), and makes it easier to iterate on these ideas than any existing library. We also benchmark ORC against ReservoirPy across reservoir sizes and numbers of training samples, as shown in Figure 2. We find that with GPU acceleration, ORC scales far more favorably than ReservoirPy. Moreover, ORC is the only current package that supports training RC for control tasks, enabled by its efficiency and the end-to-end differentiability that JAX and Equinox provide. RC-based control has recently been shown to be advantageous over other forms of RNN-based MPC (Williams et al., 2024). ORC's performance also makes it well suited for the parallel architectures needed to extend RC to higher-dimensional settings.

AI Usage Disclosure

Claude (Anthropic, Opus 4-4.8) was used for code assistance during debugging, for proofreading and editing the text of this paper, and for generating the architecture visualization SVG (Figure 1). ChatGPT (OpenAI, GPT-4o) was used to generate the ORC logo. All AI-assisted outputs, including code, paper text, and figures, were reviewed, edited, and validated by the authors before inclusion.

Acknowledgements

The authors acknowledge support from the NSF AI Institute in Dynamic Systems (grant number 2112085). The authors also thank Anastasia Bizyaeva, Noa Kaplan, and Ling-Wei Kong for insightful conversations.

References

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/jax-ml/jax

Kidger, P. (2021). *On neural differential equations* [PhD thesis]. University of Oxford.

Kidger, P., & Garcia, C. (2021). Equinox: Neural networks in JAX via callable PyTrees and filtered transformations. *Differentiable Programming Workshop at Neural Information Processing Systems*.

Lukoševičius, M., & Jaeger, H. (2009). Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, *3*(3), 127–149. https://doi.org/10.1016/j.cosrev.2009.03.005

Martinuzzi, F., Rackauckas, C., Abdelrehim, A., Mahecha, M. D., & Mora, K. (2022). ReservoirComputing.jl: An efficient and modular library for reservoir computing models. *Journal of Machine Learning Research*, *23*(288), 1–8. http://jmlr.org/papers/v23/22-0611.html

Nardo, S. (2018). *PyTorch-ESN: An echo state network module for PyTorch*. https://github.com/stefanonardo/pytorch-esn; GitHub.

Özalp, E., Margazoglou, G., & Magri, L. (2023). Reconstruction, forecasting, and stability of chaotic dynamics from partial data. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, *33*(9), 093107. https://doi.org/10.1063/5.0159479

Özalp, E., Nóvoa, A., & Magri, L. (2025). Real-time forecasting of chaotic dynamics from sparse data and autoencoders. *Computer Methods in Applied Mechanics and Engineering*, *450*, 118600. https://doi.org/10.1016/j.cma.2025.118600

Pathak, J., Hunt, B., Girvan, M., Lu, Z., & Ott, E. (2018). Model-free prediction of large spatiotemporally chaotic systems from data: A reservoir computing approach. *Physical Review Letters*, *120*(2), 024102. https://doi.org/10.1103/PhysRevLett.120.024102

Trouvain, N., Pedrelli, L., Dinh, T. T., & Hinaut, X. (2020). ReservoirPy: An efficient and user-friendly library to design echo state networks. *Artificial Neural Networks and Machine Learning – ICANN 2020*, 494–505. https://doi.org/10.1007/978-3-030-61616-8_40

Williams, J. P., Kutz, J. N., & Manohar, K. (2024). *Reservoir computing for system identification and predictive control with limited data*. https://doi.org/10.48550/arXiv.2411.05016

Wyder, P. M., Goldfeder, J. A., Yermakov, A., Zhao, Y., Riva, S., Williams, J. P., Zoro, D., Rude, A. S., Tomasetto, M., Germany, J., Bakarji, J., Maierhofer, G., Cranmer, M., & Kutz, J. N. (2025). Common task framework for a critical evaluation of scientific machine learning algorithms. *Neural Information Processing Systems 2025 Datasets and Benchmarks Track*. https://openreview.net/forum?id=pkFKjmUE3L