ReservoirPy: Fast and Flexible Echo State Networks in Python

Vaibhav Vasudevan
05/18/25

1. ReservoirPy is a Python library for building, training, and analyzing echo state networks (ESNs), a class of RNNs used for time series prediction and nonlinear dynamical systems. It provides an accessible API to rapidly prototype reservoir computing architectures and apply them to tasks like chaotic signal modeling.

2. We chose ReservoirPy because we wanted to explore a neural network method relevant to physics-based time series data. Echo state networks are structurally simpler than typical deep learning models but can capture complex system dynamics efficiently. I heard about ReservoirPy through my friend through his research and thought it was very interesting. I also made sure it runnable on my computer before using this.

3. The first public release of ReservoirPy was in 2020, developed by researchers at Université Côte d'Azur and INRIA. Similar libraries include pyESN, Oger, and EasyESN, but ReservoirPy is more actively maintained and has better documentation and modularity. The version I have right now (and the most recently updated) is 0.3.13.post1.

4. ReservoirPy is actively maintained on GitHub by its original authors. The repository shows regular updates and merges, and issues are typically addressed within a few weeks. The GitHub README includes a contributing guide and encourages community collaboration through pull requests. In fact, if you publish a paper using their package, they will feature you on the github if you email them.

5 & 6. Installation was simple using pip:

```
pip install reservoirpy
```

The install worked without issues in JupyterLab and pulled in only common Python dependencies like numpy, scikit-learn, joblib, and dill. No external compilers or conda-specific setup was required. I was able to import the library and run working examples within minutes. Its modular API supports chaining layers (e.g., Reservoir >> Ridge), which made building and experimenting with models fast and intuitive.

7. The full source code is available from GitHub: https://github.com/reservoirpy/reservoirpy.

8. This package is not widely embedded into other packages.

9. I used it inside a JupyterLab notebook, which works well because I can visualize outputs, debug cells interactively, and easily embed plots.

10. I used ReservoirPy in a JupyterLab notebook to run a 300-node Echo State Network on the chaotic Mackey-Glass time series. The code used the mackey_glass() dataset from the built-in reservoirpy.datasets, and trained a model using:

```
esn = Reservoir(...) >> Ridge(...)
esn.fit(u_train, y_train)
y_pred = esn.run(u_test)
```

11. Yes, the package integrates with matplotlib. While ReservoirPy does not produce plots directly, it returns outputs in NumPy array format, which can be immediately visualized using standard Python tools.

12.

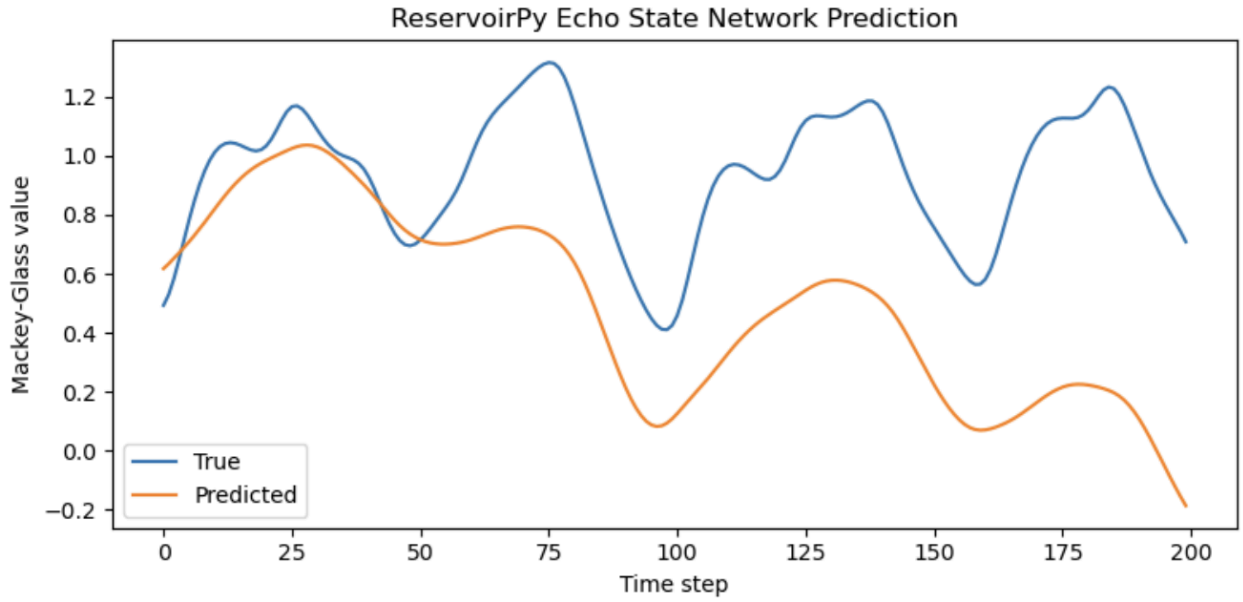

Figure 1: *Prediction results of a 300-node Echo State Network trained on 1500 steps of the Mackey-Glass time series and tested on the following 500. The blue line shows the ground truth; the orange line is the network's prediction. The test MSE was approximately 4.04, indicating some mismatch but successful short-term trend capturing.*

The figure was created using matplotlib.pyplot with:

```
plt.plot(y_test[:200], label=\"True\")
```

```
plt.plot(y_pred[:200], label=\"Predicted\")
```

13. ReservoirPy is entirely in Python. It does not require or compile any C, C++, or Fortran extensions. All modules are Python files, inspectable directly on GitHub.

14. The input is a generated time series, not a loaded dataset. I used:

```
series = mackey_glass(2000)
```

which comes from reservoirpy.datasets. This returns a NumPy array of shape (2000,), reshaped for the network. No files or external data were needed—everything was synthetic and self-contained.

15. The output is a NumPy array of predictions, generated entirely in memory. After training the Echo State Network on synthetic Mackey-Glass data, the model produces predicted values via:

```
y_pred = esn.run(u_test)
```

This prediction is compared directly against the true values, which are also synthetic and generated within the notebook. No files are written by default since everything is handled in variables or plotted using matplotlib.

16. ReservoirPy provides unit tests located in its /tests directory on GitHub. These are written with pytest and are automatically run in their continuous integration workflow. There are no formal benchmarking scripts included.

17. The results matched expected behavior for Echo State Networks: good short-term prediction, divergence over longer horizons. Combined with the library's test coverage and academic backing, this gave me confidence in its reliability.

18. ReservoirPy depends on standard scientific libraries like numpy, scikit-learn, scipy, joblib, and dill. I ran pip show reservoirpy to find out these dependencies.

19. The documentation is available at reservoirpy.readthedocs.io. It includes installation instructions, tutorials, API references, and example notebooks. It was complete and clear enough for me to use the package without needing outside help.

20. Yes.
@incollection{Trouvain2020,
  doi = {10.1007/978-3-030-61616-8_40},
  url = {https://doi.org/10.1007/978-3-030-61616-8_40},
  year = {2020},

```
  publisher = {Springer International Publishing},
  pages = {494--505},
  author = {Nathan Trouvain and Luca Pedrelli and Thanh Trung Dinh and Xavier Hinaut},
  title = {{ReservoirPy}: An Efficient and User-Friendly Library to Design Echo State Networks},
  booktitle = {Artificial Neural Networks and Machine Learning {\textendash} {ICANN} 2020}
}
```

21. I primarily used the official ReservoirPy documentation and built-in dataset functions. I didn't rely on any outside references beyond what was provided in the library itself and its GitHub repo.

22. Yes, I was able to find two papers.
    1. https://arxiv.org/abs/2401.10683
    2. https://pmc.ncbi.nlm.nih.gov/articles/PMC10982987/

23. I didn't need to learn new Python methods, but I did review how model training and prediction work in a pipeline like Reservoir >> Ridge.

24. This whole package is new to me. However, I did talk a bit with my friend on how to implement this right and what example to do.