The Journal of Open Source Software

# jaxhps: An elliptic PDE solver built with machine learning in mind

**Owen Melia** [1¶], **Daniel Fortunato** [2,3], **Jeremy Hoskins** [4], **and Rebecca Willett** [1,4,5]

**1** Department of Computer Science, University of Chicago, USA **2** Center for Computational Mathematics, Flatiron Institute, USA **3** Center for Computational Biology, Flatiron Institute, USA **4** Computational and Applied Mathematics, Department of Statistics, University of Chicago, USA **5** Data Science Institute, University of Chicago, USA ¶ Corresponding author

**DOI:** 10.xxxxxx/draft

**Software**

- Review
- Repository
- Archive

**Editor:** Anjali Sandip

**Reviewers:**

- @dc-luo
- @abhijit-c

**Submitted:** 23 June 2025
**Published:** unpublished



## Summary

Elliptic partial differential equations (PDEs) can model many physical phenomena, such as electrostatics, acoustics, wave propagation, and diffusion. In scientific machine learning settings, a high-throughput PDE solver may be required to generate a training dataset, run in the inner loop of an iterative algorithm, or interface directly with a deep neural network. To provide value to machine learning users, such a PDE solver must be compatible with standard automatic differentiation frameworks, scale efficiently when run on graphics processing units (GPUs), and maintain high accuracy for a large range of input parameters. We design the jaxhps package with these use-cases in mind by implementing a highly efficient and accurate solver for elliptic problems with native hardware acceleration and automatic differentiation support. This is achieved by expressing a highly-efficient solution method for elliptic PDEs in JAX (Bradbury et al., 2018). This software implements algorithms specifically designed for fast GPU execution of a family of elliptic PDE solvers, which are described in full in Melia et al. (2025).

Our Python package can numerically compute solutions $u(x)$ to problems of the form:

$$\mathcal{L}u(x) = f(x), \qquad x \in \Omega, \tag{1}$$
$$u(x) = g(x), \qquad x \in \partial\Omega. \tag{2}$$

In our setting, $\mathcal{L}$ is a linear, elliptic, second-order partial differential operator with spatially varying coefficient functions. The spatial domain, $\Omega$, can be a 2D square or 3D cube.

## Statement of need

While there is a vast array of PDE solvers implemented in JAX, we make a distinct contribution by implementing methods from the hierarchical Poincaré–Steklov (HPS) family of algorithms (Adrianna Gillman et al., 2015; A. Gillman & Martinsson, 2014; Martinsson, 2013). These methods use modern numerical analysis tools to resolve physical phenomena that are challenging for simpler tools, such as finite difference or finite element methods. One example of such a physical phenomenon is the oscillatory behavior of time-harmonic wave propagation simulations, which HPS methods resolve accurately and finite element methods do not (Babuška & Sauter, 1997; Yesypenko & Martinsson, 2024).

While open-source implementations of HPS methods exist for users of MATLAB (Fortunato, 2024) and C++ (Chipman, 2024), these packages do not offer native hardware acceleration or automatic differentiation capabilities. In addition, these packages do not offer support for three-dimensional problems. Yesypenko (2024) is a Python implementation of the hardware-accelerated HPS-like method described in Yesypenko & Martinsson (2024), but it is designed

for performance on extremely large 2D systems, which requires different design choices than the machine learning-focused optimizations we include in `jaxhps`.

## Software overview

The software is designed with two goals:

- Allow users to interact with a simple interface that abstracts the complex HPS algorithms.
- Organize the flow of data to allow the user to reuse computations where possible.

A typical user of `jaxhps` will wish compute a solution $u(x)$ to equations (1) and (2). The user will first specify $\Omega$ by creating `DiscretizationNode` and `Domain` objects. These objects automatically compute a high-order composite spectral discretization of $\Omega$. The `Domain` class exposes utilities for interpolating between the composite spectral discretization and a regular discretization specified by the user. If $f$ or $\mathcal{L}$ have local regions of high curvature, the `Domain` object's discretization can also be computed in an adaptive way that assigns more discretization points to those parts of $\Omega$. Additional utilities for interacting with the composite spectral discretization can be found in the `quadrature` module.

After the `Domain` is initialized, a `PDEProblem` object is created by the user from data specifying $\mathcal{L}$ and (optionally) $f$. The `PDEProblem` object also stores pre-computed interpolation and differentiation operators that can be reused during repeated calls to the solver.

The user can then execute the HPS algorithm by calling the `build_solver()` method, specifying $f$ and $g$, and finally calling the `solve()` method. During the `build_solver()` and `solve()` methods, pointers to various solution operators are stored in the `PDEProblem` object. If the problem size is large, and these solution operators can not all be stored simultaneously on a GPU, care must be taken to organize the computation and data transfer between the GPU and CPU memory. To facilitate this, we provide `solve_subtree()`, `upward_pass_subtree()`, and `downward_pass_subtree()`, newly developed algorithms designed to minimize data transfer costs. A full description of these algorithms can be found in Melia et al. (2025). After computing the solution, the user can use automatic differentiation to compute the gradient of the solution with respect to input parameters by calling `jax.jvp()` or `jax.vjp()`. Multiple examples showing these capabilities are included in the source repository and the documentation.

Finally, some researchers may want to design new algorithms by operating on the outputs of various subroutines underlying these HPS methods. To facilitate this, we expose a large collection of these subroutines in the `local_solve`, `merge`, `up_pass`, and `down_pass` modules.

## Author Contributions and Disclosure

- **Owen Melia**: Conceptualization; Methodology; Software; Writing - original draft.
- **Daniel Fortunato**: Conceptualization; Methodology; Supervision; Writing - review & editing.
- **Jeremy Hoskins**: Conceptualization; Methodology; Supervision; Writing - review & editing.
- **Rebecca Willett**: Conceptualization; Methodology; Supervision; Funding acquisition; Project administration; Writing - review & editing.

## Acknowledgements

The authors would like to thank Manas Rachh, Leslie Greengard, Vasilis Charisopoulos, and Olivia Tsang for many useful discussions. OM, JH, and RW gratefully acknowledge the support of the NSF-Simons National Institute for Theory and Mathematics in Biology (NITMB) via grants NSF DMS-2235451 and Simons Foundation MP-TMPS-00005320. OM

and RW gratefully acknowledge the support of NSF DMS-2023109, DOE DE-SC0022232, AFOSR FA9550-18-1-0166, the Physics Frontier Center for Living Systems funded by the National Science Foundation (PHY-2317138), and the support of the Margot and Tom Pritzker Foundation. The Flatiron Institute is a division of the Simons Foundation.

## References

Babuška, I. M., & Sauter, S. A. (1997). Is the pollution effect of the FEM avoidable for the helmholtz equation considering high wave numbers? *SIAM Journal on Numerical Analysis*, *34*(6), 2392–2423. https://doi.org/10.1137/S0036142994269186

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

Chipman, D. (2024). EllipticForest: A direct solver library for EllipticPartial differential equations on adaptive meshes. *Journal of Open Source Software*, *9*(96), 6339. https://doi.org/10.21105/joss.06339

Fortunato, D. (2024). A high-order fast direct solver for surface PDEs. *SIAM Journal on Scientific Computing*, *46*(4), A2582–A2606. https://doi.org/10.1137/22M1525259

Gillman, Adrianna, Barnett, A. H., & Martinsson, P.-G. (2015). A spectrally accurate direct solution technique for frequency-domain scattering problems with variable media. *BIT Numerical Mathematics*, *55*(1), 141–170. https://doi.org/10.1007/s10543-014-0499-8

Gillman, A., & Martinsson, P. G. (2014). A direct solver with $O(N)$ complexity for variable coefficient elliptic PDEs discretized via a high-order composite spectral collocation method. *SIAM Journal on Scientific Computing*, *36*(4), A2023–A2046. https://doi.org/10.1137/130918988

Martinsson, P. G. (2013). A direct solver for variable coefficient elliptic PDEs discretized via a composite spectral collocation method. *Journal of Computational Physics*, *242*, 460–479. https://doi.org/10.1016/j.jcp.2013.02.019

Melia, O., Fortunato, D., Hoskins, J., & Willett, R. (2025). *Hardware acceleration for HPS algorithms in two and three dimensions*. *arXiv:2503.17535*. https://doi.org/10.48550/arXiv.2503.17535

Yesypenko, A. (2024). *SlabLU: A Two-Level Sparse Direct Solver for Elliptic PDEs in Python* (Version 1.0.1). https://doi.org/10.5281/zenodo.11238664

Yesypenko, A., & Martinsson, P.-G. (2024). SlabLU: A two-level sparse direct solver for elliptic PDEs. *Advances in Computational Mathematics*, *50*(4), 90. https://doi.org/10.1007/s10444-024-10176-x