

mosaix-pde: Differentiable pattern-forming PDEs for machine learning, optimization, and control

Alexander E. Cohen¹✉, Samuel Degnan-Morgenstern¹, Simon Daubner², Jorn Dunkel¹, and Martin Z. Bazant¹

¹ Massachusetts Institute of Technology, United States ² Imperial College London, United Kingdom ✉ Corresponding author

DOI: [10.xxxxxx/draft](https://doi.org/10.xxxxxx/draft)

Software

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

Editor: [Fruzsina Agocs](#) ↗

Reviewers:

- [@streeve](#)
- [@p-zubieta](#)

Submitted: 23 October 2025

Published: unpublished

License

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

Summary

Pattern formation occurs in diverse physical systems across many length scales, from quantum systems to planetary atmospheres, and has applications in nanotechnology and biology (Barad et al., 2021; Kim et al., 2022). The physical laws and dynamics that dictate pattern formation are often expressed in the form of partial differential equations (PDEs). To learn models for these systems, optimize, control, or apply modern machine learning techniques to these systems, we need fast, differentiable, and GPU-powered numerical solvers.

The *mosaix-pde* package provides implementations of PDEs that describe pattern formation in a range of physical systems. The package also provides a framework for extending the implementation to additional PDEs. The code is written in JAX (Bradbury et al., 2018) and is fully differentiable and GPU-accelerated. To solve the time-dependent PDEs, *mosaix-pde* converts the PDEs using the method of lines into a system of ordinary differential equations (ODEs), and evolves the resulting ODEs using *diffax* (Kidger, 2021). Notably, *diffax* provides binomial checkpointed adjoint methods, which is often essential for differentiating through PDE solves with a large number of time steps, as the memory requirements of backpropagation scales linearly with the number of steps. *mosaix-pde* also provides implementations of specialized time stepping methods for many pattern forming systems, including semi-implicit Fourier methods (Zhu et al., 1999), Strang splitting (Bao & Cai, 2012), and the stabilized explicit Runge-Kutta method ROCK2 (Abdulle & Medovikov, 2001). For simple support for non-regular geometries, some PDEs are implemented using the smoothed boundary method for complex geometries (Yu et al., 2012). The goal of this package is to provide specialized code for the integration of physical simulations of pattern formation with inverse design, optimization, machine learning, and control.

The *mosaix-pde* package is organized around Domains, Equations, and Solvers [Figure 1](#).

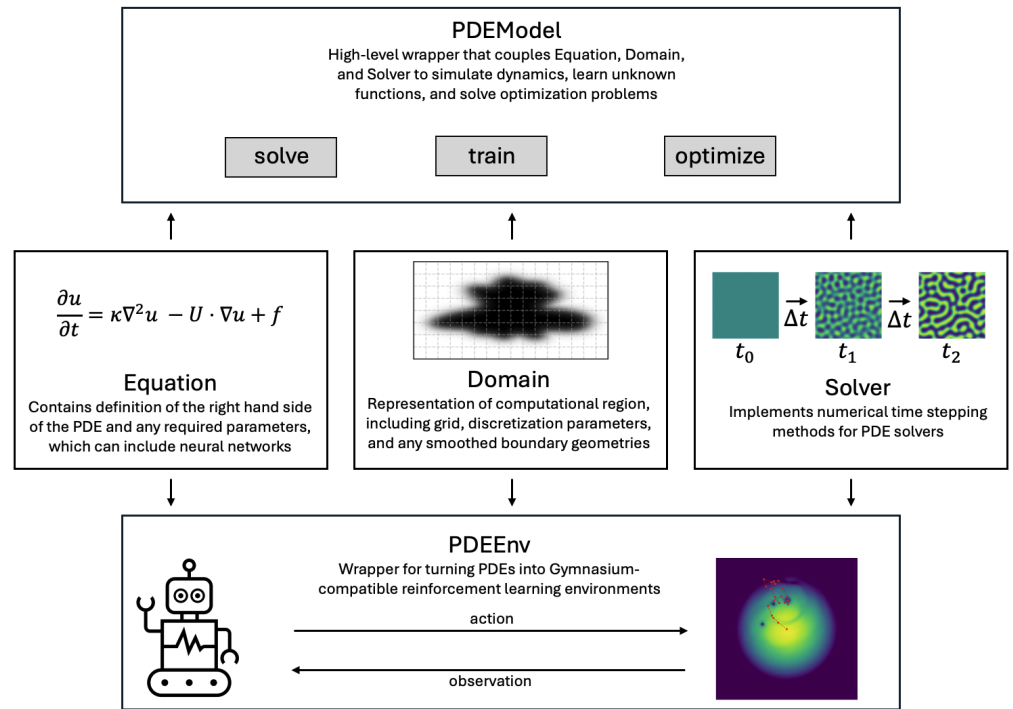


Figure 1: Code structure enables Equation, Domain, and Solver modules to be combined to build PDE models for machine learning applications.

31 The Domain class sets up the computational region for the simulation, including the mesh and
 32 axes in both real and Fourier space. The Domain also stores the Shape, which is used in the
 33 context of the smoothed boundary method. The Shape is initialized with a binary mask where
 34 1s indicate the geometry and 0s indicate the empty space. Upon initialization, the binary mask
 35 is converted to a mask with a smoothly varying shape parameter at the boundary, as required for
 36 the smoothed boundary method. This smoothing is accomplished by evolving an Allen-Cahn
 37 equation with a Laplacian term with reduced curvature minimization to maintain sharply
 38 curved features of the original shape. The degree of smoothing and curvature minimization
 39 can be controlled with hyperparameters. The Equation class consists of implementations
 40 of the right hand side of the various PDEs with both finite difference and Fourier spectral
 41 methods. Each class of PDE is contained in a module, and within each module there exists
 42 different implementations of the PDE for different sets of parameters, dimensionality, and
 43 other variations. Parameters for the PDEs can in general be Equinox modules (Kidger &
 44 Garcia, 2021), making it easy to take gradients with respect to the parameters regardless of
 45 the type of function using filter or partition transformations. Several useful functions that
 46 can be used as PDE parameters are provided as Equinox modules in the functions module,
 47 including periodic CNNs and Legendre polynomial expansions. The solvers are subclasses of
 48 `diffraX.AbstractSolver` and provide implementations of specialized time stepping methods
 49 for specific PDEs. The specification of a Domain, Equation, and a Solver is required for all
 50 downstream use cases of `mosaix-pde`.

51 Two interfaces are provided through the `PDEModel` and `PDEEnv` class for integrating with
 52 machine learning methods. The `PDEModel` is initialized with a combination of a Domain,
 53 Equation, and Solver, and provides three main methods. The first is a `solve` method for
 54 solving the specified Equation on the given Domain with the specified Solver. In addition, the
 55 `train` method provides the utilities for fitting parameters or functions within the Equation to a
 56 dataset. The `train` method uses a multiple shooting approach which is both computationally
 57 faster and more robust to noise in the dataset than other approaches. The multiple shooting

approach works by vmap-ing over multiple starting points in the dataset and evaluating the loss at future time points relative to each starting point. The specific starting points and residual evaluation points are specified through the `inds` argument of the `train` function. Currently, the optimization can be performed using the Levenberg-Marquardt or Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithms, provided through the `optimistix` package (Rader et al., 2024). Gradients can be computed using forward- or reverse-mode automatic differentiation methods, which scale differently with the number of parameters Figure 2 (Ma et al., 2021). The Levenberg-Marquardt method requires Jacobians of the residuals for the Hessian approximation, which can be computed using forward mode automatic differentiation. Since the Levenberg-Marquardt method uses Gauss-Newton approximations of the Hessian, this method generally converges faster than BFGS. However, BFGS does not require the Jacobian of the residuals for the Hessian approximation, and thus the gradients can be computed easily using backpropagation, which scales much better with the number of parameters Figure 2. This tradeoff between scaling with parameter numbers and convergence of optimization must be considered for these differentiable physics optimization problems. Finally, the `optimize` method provides an interface for minimizing a scalar function of the PDE solution. The function to minimize is specified through the `objective_function` argument, and BFGS is used to perform the optimization.

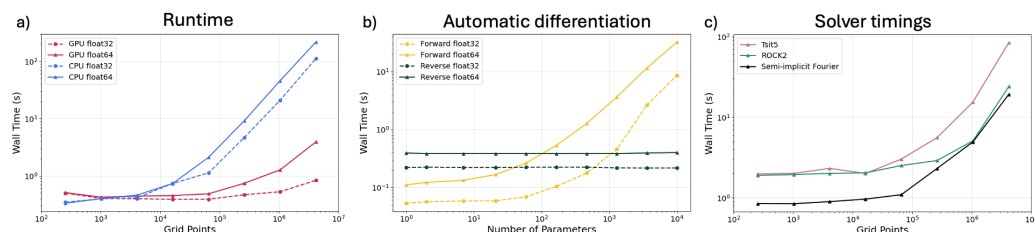


Figure 2: Benchmarking performance of solvers and gradients with different number of grid points and parameters.

For reference, we benchmark the performance of numerically solving and computing gradients of PDE solutions Figure 2. We report the wall time scaling as a function of grid points of 10,000 time steps of the Cahn-Hilliard equation using a semi-implicit Fourier time stepping method, run on both GPU and CPU with Float32 and Float64 precision. We further show the wall time scaling with respect to the number of parameters when computing gradients through 1,000 time steps of the Cahn-Hilliard equation using forward- and reverse-mode automatic differentiation. Finally, we compare a Tsit5 time stepper with our ROCK2 implementation and semi-implicit Fourier method, all coupled with a PID step size controller for the Cahn-Hilliard equation. PDEModel streamlines model learning and optimization by unifying a Domain, Equation, and Solver.

The `PDEEnv` class is useful for turning a PDE into a Gymnasium-registered reinforcement learning (RL) environment that can be used to train RL agents with libraries like Stable Baselines (Raffin et al., 2021; Towers et al., 2024). In addition to the Domain, Equation, and Solver, the `PDEEnv` class requires a `step_dt`, which is the time span of one step of the environment, and a `numeric_dt` which is the time step to use for numerical integration. These are separate parameters because the reaction time of the agent is often larger than the time step needed for numerical stability. Beyond these fields, many other pieces of information must be provided to form the RL environment, including reward functions, observation functions, and reset functions.



Figure 3: Single episode of an RL environment created from the Gross-Pitaevskii equation.

We demonstrate an example of creating an RL environment designed to form vortices in a Bose-Einstein condensate by controlling the position of an external laser source [Figure 3](#). The episode is simulated by sampling random actions that move the position of the laser (red line), where the reward is calculated by counting the number of vortices in the condensate (black and white circles).

Statement of need

Pattern formation and phase separation are fundamental processes across physics, chemistry, biology, and materials science, with technological applications ranging from developmental biology to nanostructured materials. At the same time, the rapid growth of scientific machine learning has shown how partial differential equation (PDE) models can be combined with modern optimization and learning techniques to accelerate discovery, most prominently in applications such as material modeling ([Zhao et al., 2020, 2023](#)), weather and climate modeling ([Kochkov et al., 2024](#)), and biophysics ([Supekar et al., 2023](#)). Building on these advances, there is growing interest in extending such capabilities to ever more complicated pattern-forming systems, where fast, differentiable, and GPU-accelerated PDE solvers can enable parameter learning, design optimization, and reinforcement learning-based control. To support this, the community needs open-source tools that are performant, easy to use, well documented, and straightforward to extend. Existing simulation libraries for pattern formation provide valuable tools, but are often not directly integrated with these machine learning workflows ([Burns et al., 2020](#); [Daubner et al., 2025](#); [Walker et al., 2023](#); [Zwicker, 2020](#)). In addition, packages that treat PDEs as reinforcement learning environments are generally restricted to a small set of select equations ([Bhan et al., 2024](#); [Werner & Peitz, 2024](#)). Our framework extends this ecosystem by coupling performant PDE solvers with differentiability, RL interfaces, and optimization capabilities, making it easier to study and control complex spatiotemporal dynamics across disciplines. The code is currently being used by researchers to learn models for battery nanoparticles, optimize phase separation in materials, and control pattern formation in Bose-Einstein condensates. In the future, we plan to expand the range of physical systems and PDEs supported by the package and continue advancing numerical methods for differentiable simulation, with the goal of providing an accessible and practical framework for machine learning with PDEs.

Acknowledgments

The authors acknowledge the MIT Office of Research Computing and Data for providing computational resources and advice on open-source scientific computing software.

References

- Abdulle, A., & Medovikov, A. A. (2001). Second order chebyshev methods based on orthogonal polynomials. *Numerische Mathematik*, 90(1), 1–18.
- Bao, W., & Cai, Y. (2012). Mathematical theory and numerical methods for bose-einstein condensation. *arXiv Preprint arXiv:1212.5341*.
- Barad, H.-N., Kwon, H., Alarcón-Correa, M., & Fischer, P. (2021). Large area patterning of nanoparticles and nanostructures: Current status and future prospects. *ACS Nano*, 15(4), 5861–5875.
- Bhan, L., Bian, Y., Krstic, M., & Shi, Y. (2024). *PDE control gym: A benchmark for data-driven boundary control of partial differential equations*. <https://arxiv.org/abs/2405.11401>
- 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>
- Burns, K. J., Vasil, G. M., Oishi, J. S., Lecoanet, D., & Brown, B. P. (2020). Dedalus: A flexible framework for numerical simulations with spectral methods. *Physical Review Research*, 2(2), 023068. <https://doi.org/10.1103/PhysRevResearch.2.023068>
- Daubner, S., Cohen, A. E., D'Ágostini, B., & Cooper, S. J. (2025). Evoxels: A differentiable physics framework for voxel-based microstructure simulations. *arXiv Preprint arXiv:2507.21748*.
- 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 2021*.
- Kim, H., Skinner, D. J., Glass, D. S., Hamby, A. E., Stuart, B. A., Dunkel, J., & Riedel-Kruse, I. H. (2022). 4-bit adhesion logic enables universal multicellular interface patterning. *Nature*, 608(7922), 324–329.
- Kochkov, D., Yuval, J., Langmore, I., Norgaard, P., Smith, J., Mooers, G., Klöwer, M., Lottes, J., Rasp, S., Düben, P., & others. (2024). Neural general circulation models for weather and climate. *Nature*, 632(8027), 1060–1066.
- Ma, Y., Dixit, V., Innes, M. J., Guo, X., & Rackauckas, C. (2021). A comparison of automatic differentiation and continuous sensitivity analysis for derivatives of differential equation solutions. *2021 IEEE High Performance Extreme Computing Conference (HPEC)*, 1–9.
- Rader, J., Lyons, T., & Kidger, P. (2024). Optimistix: Modular optimisation in JAX and equinox. *arXiv:2402.09983*.
- Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., & Dormann, N. (2021). Stable-Baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268), 1–8. <http://jmlr.org/papers/v22/20-1364.html>
- Supekar, R., Song, B., Hastewell, A., Choi, G. P., Mietke, A., & Dunkel, J. (2023). Learning hydrodynamic equations for active matter from particle simulations and experiments. *Proceedings of the National Academy of Sciences*, 120(7), e2206994120.
- Towers, M., Kwiatkowski, A., Terry, J., Balis, J. U., De Cola, G., Deleu, T., Goulão, M., Kallinteris, A., Krimmel, M., KG, A., & others. (2024). Gymnasium: A standard interface for reinforcement learning environments. *arXiv Preprint arXiv:2407.17032*.
- Walker, B. J., Townsend, A. K., Chudasama, A. K., & Krause, A. L. (2023). VisualPDE:

- 173 Rapid interactive simulations of partial differential equations. *Bulletin of Mathematical*
174 *Biology*, 85(11), 113.
- 175 Werner, S., & Peitz, S. (2024). Numerical evidence for sample efficiency of model-based over
176 model-free reinforcement learning control of partial differential equations. *2024 European*
177 *Control Conference (ECC)*, 2965–2971.
- 178 Yu, H.-C., Chen, H.-Y., & Thornton, K. (2012). Extended smoothed boundary method for
179 solving partial differential equations with general boundary conditions on complex boundaries.
180 *Modelling and Simulation in Materials Science and Engineering*, 20(7), 075008.
- 181 Zhao, H., Deng, H. D., Cohen, A. E., Lim, J., Li, Y., Fraggadakis, D., Jiang, B., Storey, B. D.,
182 Chueh, W. C., Braatz, R. D., & others. (2023). Learning heterogeneous reaction kinetics
183 from x-ray videos pixel by pixel. *Nature*, 621(7978), 289–294.
- 184 Zhao, H., Storey, B. D., Braatz, R. D., & Bazant, M. Z. (2020). Learning the physics of
185 pattern formation from images. *Physical Review Letters*, 124(6), 060201.
- 186 Zhu, J., Chen, L.-Q., Shen, J., & Tikare, V. (1999). Coarsening kinetics from a variable-
187 mobility cahn-hilliard equation: Application of a semi-implicit fourier spectral method.
188 *Physical Review E*, 60(4), 3564.
- 189 Zwicker, D. (2020). Py-pde: A python package for solving partial differential equations.
190 *Journal of Open Source Software*, 5(48), 2158.

DRAFT