The Journal of Open Source Software

SiNDAE: The Simultaneous Neural Differential-Algebraic systems of equations training Python package

Nicolas Smits [1], **Laurens Lueg** [1], **Victor Alves** [1], **Carl D. Laird** [1], **John R. Kitchin** [1], **and Lorenz T. Biegler** [1]

1 Department of Chemical Engineering, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Open Journals

Reviewers:

- @openjournals

Submitted: 01 January 1970
Published: unpublished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

Problem Statement

When modeling dynamical systems, first-principles mechanistic, or constitutive equations are often incomplete or insufficient to capture the true underlying complexity of the phenomena at play. Depending on the system, the modeling shortfall could manifest itself as a reaction rate, growth term, or transfer coefficients for example, which could depend on measured state variables or inputs via a relationship which is not explicitly modeled. While domain experts may recognize portions of the missing dynamics, formulating and validating new governing equations may be slow, or even intractable. Hybrid modeling resolves these dilemmas by leveraging existing mechanistic knowledge while selectively embedding data-driven surrogates to improve fitting and generalization performance. In this way, hybrid models have a two-fold advantage over data-driven or purely-mechanistic approaches. Unlike data-driven models, hybrid models can enforce physical invariants such as mass, energy, or momentum balances into solution routines, allowing mappings learned from data to remain physically consistent. Simultaneously, unlike purely-mechanistic models, hybrid models are able to use valuable experimental data to inform and improve the predictive capability of dynamical systems. The mechanistic vs. data-driven trade off can be tuned to achieve the best performance based on modeling goals, whether that be simulation, model-based-design of experiments (MBDoE), embedded surrogate optimization or other decision-making tasks in the context of engineering (Franceschini & Macchietto, 2008; Mahanty, 2023).

While the predictive and extrapolative benefits of hybrid modeling are well-documented across decades of literature (Herrera-Ruiz et al., 2025; Narayanan et al., 2022; Psichogios & Ungar, 1992; Thompson & Kramer, 1994), existing open-source software usually trains the so-called hybrid models via the *sequential* approach, in which the model is simulated (i.e., integrated forward in time and/or space coordinates) at every step before the neural network parameters are updated by first-order optimization methods such as gradient descent. Furthermore, applications target Ordinary Differential Equation (ODE) systems almost exclusively, with a few exceptions (Drgona et al., 2023) which also solve the training problem using sequential integration. Comparatively little tooling addresses hybrid Differential-Algebraic Equation (DAE) systems, where algebraic constraints must hold alongside differential dynamics. SiNDAE fills this software gap by training hybrid models, including DAE systems, with the simultaneous approach for dynamic optimization problems (Biegler, 2007): the model equations are discretized in time, and the state trajectories and network parameters are computed together as the solution of a single nonlinear program (NLP). This approach is faster than sequential training for small- to moderate-size problems and provides a natural framework for hybrid DAEs, and a decomposition variant extends it to large data sets (Lueg, Alves, Schicksnus, et al., 2026).

DRAFT

Background

In the field of process systems modeling in science and engineering, paradigms combining data-driven components with purely mechanistic models have gained traction as promising hybrid improvements on traditional methods. The milestone approaches published in this area of research include: Physics Informed Neural Networks (PINNs) (Raissi et al., 2019), Neural Ordinary Differential Equations (NeurODEs) (Chen et al., 2018), and Universal Differential Equations (UDEs) (Rackauckas et al., 2021). Said hybrid methods have been applied to ODE systems in domains ranging from bioprocesses (Bangi et al., 2022; Narayanan et al., 2022) to power systems (Xiao et al., 2023) to wastewater treatment (Huang et al., 2025). However, mature scientific machine learning frameworks such as torchdiffeq (Chen et al., 2018), DiffEqFlux.jl (Rackauckas et al., 2019), and diffrax (Kidger, 2022) rely on sequential integrate-and-backpropagate mechanisms designed for ODEs.

Recently, work extending hybridization to Differential Algebraic Systems of Equations (DAEs), so that explicit algebraic constraints are obeyed, has relied on sequential training or penalty loss functions. These methods use techniques such as integration and adjoint sensitivities (Kim et al., 2021), stabilization or projection onto the constraint manifold (White et al., 2023), operator splitting between the differential and algebraic subsystems (Koch et al., 2025), or penalty terms on the constraint residual (Moya & Lin, 2022). Consequently, algebraic constraints are enforced only softly or approximately. Stabilization methods keep physical invariants only approximately satisfied during simulation (White et al., 2023), and penalty formulations enforce the DAE only as soft constraints, whose accuracy depends on hyperparameters that, chosen poorly, leave the training of hybrid models ill-conditioned or slow to converge (Moya & Lin, 2022). Furthermore, most existing DAE-focused PINN research aims to learn trajectory solutions of fully specified systems (Moya & Lin, 2022), rather than discovering unknown functional components within constrained systems.

To overcome these issues, Lueg et al. propose two novel training strategies, the simultaneous and the decomposition approaches, to tackle rigorous DAE constraint enforcement, addressing higher-index hybrid DAE systems, and scale to many trajectories (Lueg, Alves, Schicksnus, et al., 2026). The simultaneous approach employs a strategy similar to simultaneous approaches for the dynamic optimization of purely-mechanistic DAEs by discretizing the domain using direct transcription methods such as orthogonal collocation, so that the network parameters and the discretized states and algebraic variables of all trajectories become the decision variables of a single NLP, which is then solved using a capable nonlinear solver such as IPOPT (Wächter & Biegler, 2006). Said method effectively achieves constraint enforcement and higher-index system solutions because no simulation is performed during training: algebraic equations of any index are ordinary NLP constraints, satisfied at every collocation point at the solution. However, since the network parameters couple all trajectories, the NLP grows with the number of parameters and the amount of training data, which limits the method to moderately sized systems. To address this limitation, the decomposition approach adapts training to a bi-level optimization problem, where neural network weights are updated via a sensitivity-based stochastic gradient descent outer loop (Adam), with the sensitivities of the subproblem solutions obtained by implicit differentiation of their optimality (KKT) conditions, while interior iterations solve subproblems parallelized across trajectories with the neural network parameters frozen. Although the decomposition approach alternates between solving the subproblems and updating the weights, it retains rigorous constraint enforcement of the mechanistic equations, since each subproblem remains a fully discretized NLP (only the network equation is relaxed with penalized slack variables during training), and higher-index performance while scaling to large numbers of trajectories.

SiNDAE

SiNDAE implements the simultaneous and decomposition approaches from *A Simultaneous Approach for Training Neural DAEs* by Lueg et al. (Lueg, Alves, Schicksnus, et al., 2026),

DRAFT

bridging the tooling and functionality gap by allowing components of DAEs to be learned subject to algebraic constraints efficiently. The package leverages Pyomo and Pyomo DAE to build the underlying nonlinear programs (NLPs) solved in both approaches, ensuring both rigorous constraint enforcement and sufficient modeling flexibility for diverse applications. SiNDAE's API design is intended to be user-friendly, mirroring widespread machine learning packages' architecture, namely scikit-learn's (Pedregosa et al., 2011) and SparkML (Meng et al., 2016) estimator oriented design. Similarly, its installation is intended to be hassle-free by alleviating the need for external binaries for the linear and nonlinear solvers required, thus streamlining adoption. Interoperability into complementary machine learning, modeling, and optimization ecosystems is facilitated through SiNDAE's export formats, enabling the dissemination of accurate hybrid models for use in downstream use-cases. Together, SiNDAE's functionality and design is intended to facilitate the training of hybrid DAE systems in a user-friendly format, with auxiliary integrations kept in mind.

Statement of need

Audience & Scientific Gap:

SiNDAE addresses a critical gap in scientific machine learning by providing a dedicated framework for modeling and training hybrid Differential-Algebraic Equation (DAE) systems effectively using the simultaneous approach (Biegler, 2007). The package targets researchers and practitioners in science and engineering in general, including applied physicists who hold mechanistic DAE models with uncharacterized or uncertain terms (e.g., complex reaction kinetics or mass transfer coefficients) and possess noisy, incomplete state measurements. While existing hybrid differential equation packages train models sequentially and focus almost exclusively in Ordinary Differential Equations (ODEs), SiNDAE trains them simultaneously, allowing users to learn unknown functional components directly within index-1 and higher-index DAE systems. By enforcing governing algebraic equations as hard constraints at every discretization point during both training and inference, SiNDAE ensures that unobserved state variables remain physically consistent, enabling robust extrapolation to unmeasured operating regimes, a defining advantage of hybrid modeling (Herrera-Ruiz et al., 2025; Stosch et al., 2014).

Usability & Open Access

A major barrier to the adoption and open reproduction of hybrid modeling workflows is the lack of open-source software and the reliance on proprietary or complex solver installation stacks (Mahanty, 2023). Traditional dynamic optimization workflows often depend on solvers like IPOPT (Wächter & Biegler, 2006), paired with efficient sparse linear solvers (e.g., HSL's MA27) that require specialized licenses and compiled binaries, hindering code sharing across communal repositories. SiNDAE overcomes this hurdle by featuring a completely license-binary-free default installation powered by the POUNCE (Kitchin, 2026b) interior-point solver and the FERAL (Kitchin, 2026a) linear solver, both distributed directly via standard Python wheels. Practitioners can execute the entire pipeline out of the box, spanning parameter pre-training and simultaneous/decomposition training to model inference, while retaining optional support for traditional cyipopt backends. Furthermore, SiNDAE adopts an estimator-oriented API architecture modeled after scikit-learn (HybridDAE and ProblemDefinition), reducing the learning curve for ML practitioners. Trained models can be exported to Equinox, ONNX, JSON, or OMLT NetworkDefinition objects (Ceccon et al., 2022), allowing identified surrogates to be embedded seamlessly into downstream optimization and control workflows. Additionally, to enable expedited hybrid modeling workflows using modern AI coding agents (primarily Claude Code), a directory called `sindae-skills` containing an instruction file and several skill files has been included. When configured for a coding agent working on a SiNDAE project, the instruction file guides the user interaction and modeling process through latex equation

DRAFT

rendering and transparency rules, while skill files provide design decision and implementation best practices.

Ecosystem & Dependencies

SiNDAE is built upon established scientific Python infrastructure. Symbolic model construction and Lagrange–Radau collocation on finite elements are managed via Pyomo and Pyomo.DAE (Bynum et al., 2021; Nicholson et al., 2018), while Pyomo's PyNumero interface exposes the underlying NLP structures and KKT systems for implicit differentiation. Neural network architectures, exact higher-order automatic differentiation, and outer-loop optimization updates are powered by JAX (Bradbury et al., 2018), Equinox (Kidger & Garcia, 2021), and Optax (DeepMind et al., 2020). Fundamental data manipulation, scientific routines, and visualization are supported by NumPy, SciPy, and Matplotlib. Parallelization of the decomposition strategy is enabled through the optional use of MPI for Python (Dalcin et al., 2005). Exporting and printing functionality are handled by the jax2onnx (Enpasos, 2025) and tabulate (Astanin, 2026) packages respectively. Figure 1 illustrates the resulting dependency architecture.

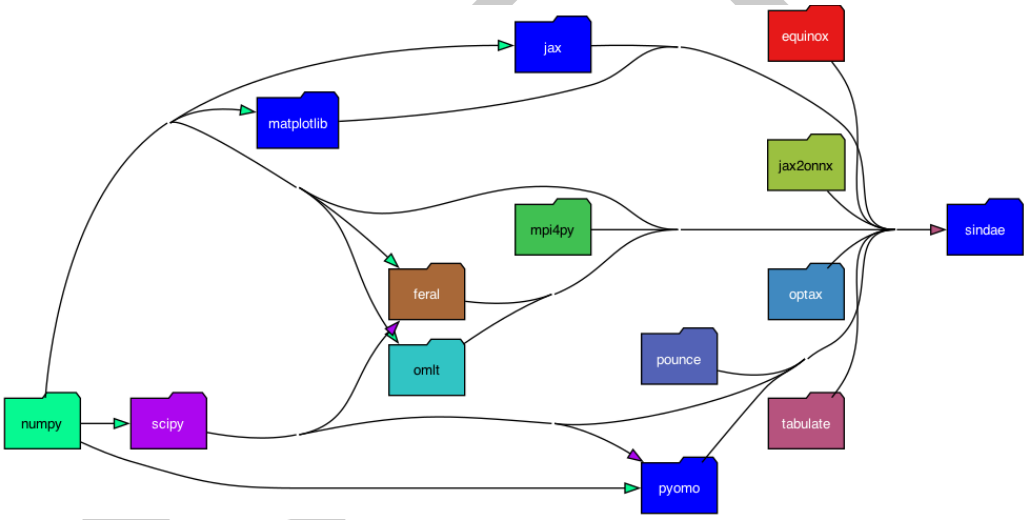


Figure 1: Dependency graph generated with pydeps showing the packages used in SiNDAE

State of the field

Existing open-source packages and proposed approaches for hybrid modeling with neural differential equations, such as torchdiffeq (Chen et al., 2018), DiffEqFlux.jl (Rackauckas et al., 2019), and diffrax (Kidger, 2022), have their models trained using ODE integrators, while the DAE-focused approaches in the literature either still use integrator-based training and/or rely on penalty terms added to the loss functions during training (Drgona et al., 2023; Kim et al., 2021; Koch et al., 2025; Moya & Lin, 2022; White et al., 2023). On the other hand, tools that implement the simultaneous approach, such as Pyomo.DAE (Nicholson et al., 2018), were originally intended for purely mechanistic/physics-based models. Furthermore, well-known packages such as the Optimization and Machine Learning Toolkit (OMLT) (Ceccon et al., 2022) are focused in embedding already trained neural networks into Pyomo models for optimization and decision-making purposes, but it does not focus on the training step. Since the simultaneous approach replaces an integrator with a discretized NLP that is solved employing an interior-point method, it is a different approach compared to the integrate-and-backpropagate architecture of the previous existing packages. Therefore, SiNDAE was conceptualized and built on top of the Pyomo (Bynum et al., 2021) ecosystem, facilitating both the training of embedded neural networks to the discretization capabilities of Pyomo.DAE

and the modeling of physics-based equations.

Software design

The main design decision in SiNDAE is to function on top of an algebraic modeling language that is free and open-source, i.e. the Pyomo ecosystem (Bynum et al., 2021), augmenting a stable and robust programming package with hybrid modeling capabilities in the context of scientific machine learning. Furthermore, this design decision allows SiNDAE to be a package that facilitates not only the training step of the neural DAEs, but also the joint representation of both physics-based and machine-learning based expressions, due to the Pyomo modeling paradigm. Regarding the training step, SiNDAE offers two approaches that share the same problem definition and can be switched with a single keyword argument. First, the simultaneous approach solves the training problem as a single, monolithic constrained NLP, while providing the interior-point solver with exact second derivatives. For small to medium-sized hybrid models with few neural components and not many trajectories, this approach converges adequately in feasible computational time. However, as more training trajectories are available and deeper neural networks may be needed, this approach rapidly becomes challenging from a computational time perspective, since the size of the underlying NLP grows with the number of network parameters and trajectories quickly.

Therefore, based on our published recent work (Lueg, Alves, Schicksnus, et al., 2026) we also provide a second training method, called the decomposition-based approach. It replaces a single large NLP with smaller per-trajectory ones that can be solved in parallel with parallel computing, i.e., Message Passing Interface (MPI) (Dalcin et al., 2005) in Python, at the cost of an outer gradient-based loop. Users of SiNDAE can easily switch between both training methods with a single keyword argument definition without modifying their model instance.

Regarding the neural network embedment in the large-scale optimization training problem, software design decisions were also made. We observed trade-off relationships related to how the neural network is embedded in the training problem itself. By default, the neural networks can be written as explicit Pyomo expressions, exposing exact second derivatives to the solver, while the Pyomo grey-box interface option evaluates the network in modern ML packages (i.e., JAX) providing only first derivatives, while employing a quasi-Newton approximation of the Hessian. This can be slower per iteration but scales adequately to deeper neural networks. Moreover, since NLP solvers require robust initial estimates, the `fit()` pipeline first smooths the measured data and pre-trains the neural components on the smoothed trajectories, and uses these results to initialize training. All of these software design decisions are implemented in a way that a practitioner does not have to intervene, but advanced users have easy access to these steps.

Lastly, SiNDAE's software stack separates the mechanistic model representations from the neural network ones. Models are written in Pyomo, so that the equations are expressed as in conventional dynamic optimization approaches, while JAX handles the neural networks and their derivatives. PyNumero (Rodriguez et al., 2023) bridges both by exposing the NLP and Karush-Kuhn-Tucker (KKT) matrices needed by the decomposition-based approach. POUNCE and FERAL are used as the default solvers so that the full pipeline installs from Python wheels without requiring any extra steps. IPOPT and HSL (formerly the Harwell Subroutine Library) solvers are available as optional dependencies if the user has an HSL license.

Vignette

To illustrate `SiNDAE`'s core capabilities and user workflow, consider a classic process engineering system: a two-tank liquid cascade operating as an index-1 Differential-Algebraic Equation (DAE), shown in Figure 2a. In this system, liquid levels $h_1(t)$ and $h_2(t)$ evolve under a constant feed rate F_{in} and inter-tank flows $q_1(t)$ and $q_2(t)$:

DRAFT

$$A_1 \frac{dh_1}{dt} = F_{in} - q_1, \qquad A_2 \frac{dh_2}{dt} = q_1 - q_2, \qquad q_2 = c_2\sqrt{h_2}$$

In practical scenarios, physical relationships are often only partially known. Here, the outlet relation of the second tank (q_2) is explicitly governed by a known valve equation, while the inter-tank discharge law (q_1) is unknown and must be learned from noisy, sparse trajectory data of the liquid levels.

The core philosophy of `SiNDAE` is to eliminate the mathematical and software engineering overhead traditional in dynamic optimization. Rather than requiring users to manually formulate orthogonal collocation matrices, derive sensitivity relations, or manage complex C++/Fortran solver dependencies, `SiNDAE` encapsulates the entire hybrid training and inference pipeline into a high-level, estimator-oriented architecture.

To train this hybrid model with `SiNDAE`, a user only needs to perform three concise steps:

1. **Define the Physics:** Subclass `ProblemDefinition` to specify the mechanistic DAE in `Pyomo` format and designate which variables enter and exit the neural surrogate (Figure 2b).
2. **Configure and Fit:** Assemble the multi-stage training pipeline by pairing the problem with stage configuration objects. Calling `fit()` automatically handles data smoothing, supervised network pre-training, and training with the simultaneous approach (Figure 2c). Users can seamlessly swap between the simultaneous and decomposition algorithms or customize neural architectures via `SiNDAE`'s grey-box interface.
3. **Predict and Export:** Call `predict()` on the fitted estimator to evaluate new initial conditions or operating regimes (Figure 2c). Because `predict()` solves the discretized hybrid DAE with the trained network fixed, all predicted trajectories satisfy hard algebraic constraints at every collocation point.

As shown in Figure 2d, although the neural network $f_{NN}(h_1, h_2; \theta)$ receives no prior structural information, its learned response collapses onto the true physics ($c_1\sqrt{h_1}$). Furthermore, predictions under unseen initial charges track reference trajectories (Figure 2e). Once identified, the trained hybrid model can be directly exported to ONNX, Equinox, or `OMLT` formats for use in downstream surrogate optimization and control tasks.

DRAFT

The Journal of Open Source Software

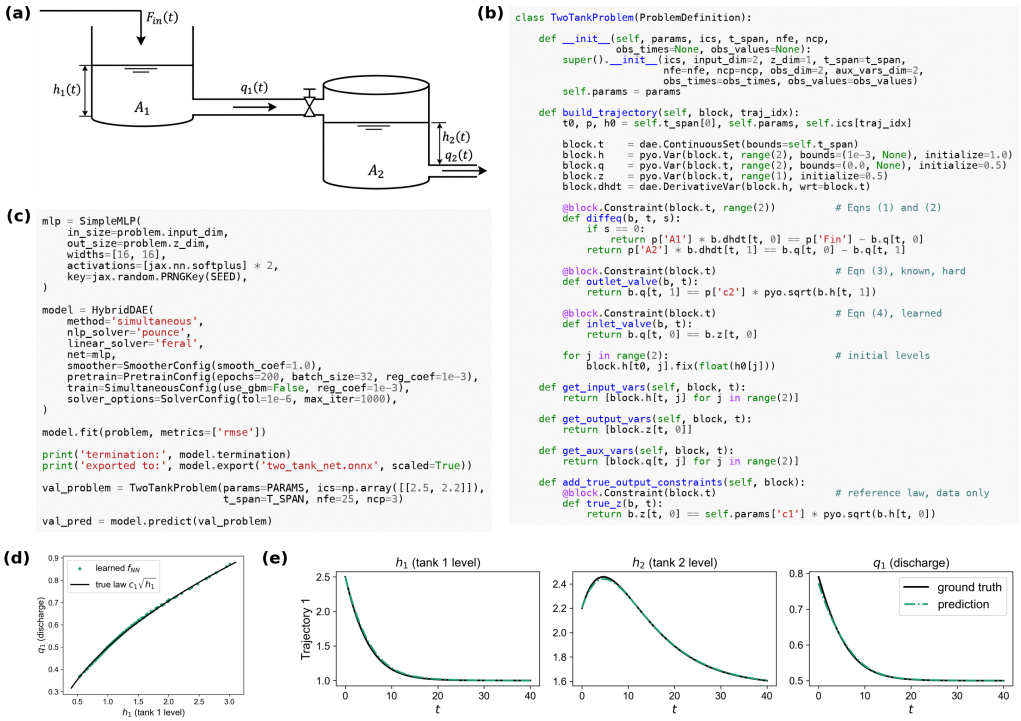


Figure 2: A complete SiNDAE workflow on a two-tank cascade DAE. (a) the physical system, with the inter-tank discharge q_1 treated as unknown; (b) system definition via a ProblemDefinition subclass; (c) estimator assembly, multi-stage fit, export, and constrained predict execution; (d) recovery of the un-modeled discharge law against the physical truth; (e) physically consistent prediction on holdout initial conditions.

Research impact statement

SiNDAE implements in a straightforward interface, the training methods introduced in our recent work by Lueg et al. (Lueg, Alves, Schicksnus, et al., 2026), published in *Computational Optimization and Applications*. SiNDAE started from the research code used in that work. The package is released on PyPI, tested through continuous integration, and documented with worked examples that include the case-studies from our published work previously mentioned.

SiNDAE has also been used in ongoing research presented at conferences and is being used in internally funded projects by our research teams. Presentations include the decomposition approach for scalable training of hybrid models (Lueg, Alves, Biegler, et al., 2026) and a multiscale hybrid model for predicting monoclonal antibody glycosylation in mammalian cell cultures (Alves et al., 2026), both presented at the 2026 AIChE Annual Meeting.

Availability

SiNDAE is available on PyPI and has its source code hosted on GitHub. The documentation contains thorough descriptions of the API and functionality, as well as theoretical grounding for both the simultaneous and the decomposition methods available, instructions on how to configure a hybrid DAE system, as well as a gallery of examples.

The authors aim to supply both proper documentation to the users in the open-source software community and provide sufficient grounding in theory to enable them to design, implement, and distribute hybrid DAE models for their use-case.

DRAFT

AI Usage Disclosure

AI Coding assistants, primarily Claude Code, were used during the development of `SiNDAE`. Their role was confined to restructuring the original research code of (Lueg, Alves, Schicksnus, et al., 2026) into an installable package, migrating the solver backends, writing tests, configuring documentation, and debugging. The numerical methods, the problem formulations, and the validation criteria are the authors' own. Claude was also used to help review the grammar, wording, and terminology of this manuscript and to suggest its restructuring to follow JOSS required guidelines and sections. All AI-assisted text was reviewed and revised by the authors. AI-assisted code was verified by the authors and through the package's automated test suite.

Acknowledgements

The authors gratefully acknowledge the support from the Center for Advanced Process Decision-making (CAPD) in the Department of Chemical Engineering at Carnegie Mellon University

References

Alves, V., Robinson, A. S., Kitchin, J. R., & Laird, C. D. (2026, November). A multiscale, hybrid scientific machine learning approach for predicting monoclonal antibody glycosylation in mammalian cell cultures. *2026 AIChE Annual Meeting*. https://studio.m-anage.com/aiche/2026/prelim.cgi/Paper/737900

Astanin, S. (2026). *tabulate: Pretty-print tabular data in Python*. GitHub. https://github.com/astanin/python-tabulate

Bangi, M. S. F., Kao, K., & Kwon, J. S.-I. (2022). Physics-informed neural networks for hybrid modeling of lab-scale batch fermentation for β-carotene production using Saccharomyces cerevisiae. *Chemical Engineering Research and Design*, *179*, 415–423. https://doi.org/10.1016/j.cherd.2022.01.041

Biegler, L. T. (2007). An overview of simultaneous strategies for dynamic optimization. *Chemical Engineering and Processing: Process Intensification*, *46*(11), 1043–1053. https://doi.org/10.1016/j.cep.2006.06.021

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*. https://github.com/jax-ml/jax

Bynum, M. L., Hackebeil, G. A., Hart, W. E., Laird, C. D., Nicholson, B. L., Siirola, J. D., Watson, J.-P., & Woodruff, D. L. (2021). *Pyomo – optimization modeling in Python* (Third, Vol. 67). Springer. https://doi.org/10.1007/978-3-030-68928-5

Ceccon, F., Jalving, J., Haddad, J., Thebelt, A., Tsay, C., Laird, C. D., & Misener, R. (2022). OMLT: Optimization & machine learning toolkit. *Journal of Machine Learning Research*, *23*(349), 1–8. https://doi.org/10.48550/arXiv.2202.02414

Chen, R. T. Q., Rubanova, Y., Bettencourt, J., & Duvenaud, D. (2018). Neural ordinary differential equations. *Advances in Neural Information Processing Systems*, *31*. https://doi.org/10.48550/arXiv.1806.07366

Dalcin, L., Paz, R., & Storti, M. (2005). MPI for Python. *Journal of Parallel and Distributed Computing*, *65*(9), 1108–1115. https://doi.org/10.1016/j.jpdc.2005.03.010

DeepMind, Babuschkin, I., Baumli, K., Bell, A., Bhupatiraju, S., Bruce, J., Buchlovsky, P., Budden, D., Cai, T., Clark, A., Danihelka, I., Dedieu, A., Fantacci, C., Godwin, J., Jones, C., Hemsley, R., Hennigan, T., Hessel, M., Hou, S., … Viola, F. (2020). *The DeepMind JAX Ecosystem*. https://github.com/google-deepmind

DRAFT

Drgona, J., Tuor, A., Koch, J., Shapiro, M., King, E., & Vrabie, D. (2023). *Domain aware deep-learning algorithms integrated with scientific-computing technologies (DADAIST)* (PNNL-34895). Pacific Northwest National Laboratory. https://doi.org/10.2172/2331469

Enpasos. (2025). *jax2onnx: Export JAX, Flax NNX, Flax Linen, and Equinox to ONNX*. GitHub. https://github.com/enpasos/jax2onnx

Franceschini, G., & Macchietto, S. (2008). Model-based design of experiments for parameter precision: State of the art. *Chemical Engineering Science*, *63*, 4846–4872. https://doi.org/10.1016/j.ces.2007.11.034

Herrera-Ruiz, J. F., Fontalvo, J., & Prado-Rubio, O. A. (2025). Hybrid modeling for bioprocesses: Architectures, applications, and perspectives. *Engineering Reports*, *7*(12). https://doi.org/10.1002/eng2.70502

Huang, X., Kandris, K., & Katsou, E. (2025). Training stiff neural ordinary differential equations in data-driven wastewater process modelling. *Journal of Environmental Management*, *373*, 123870. https://doi.org/10.1016/j.jenvman.2024.123870

Kidger, P. (2022). *On neural differential equations* [PhD thesis, University of Oxford]. https://doi.org/10.48550/arXiv.2202.02435

Kidger, P., & Garcia, C. (2021). Equinox: Neural networks in JAX via callable PyTrees and filtered transformations. *arXiv Preprint arXiv:2111.00254*. https://doi.org/10.48550/arXiv.2111.00254

Kim, S., Ji, W., Deng, S., Ma, Y., & Rackauckas, C. (2021). Stiff neural ordinary differential equations. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, *31*(9), 093122. https://doi.org/10.1063/5.0060697

Kitchin, J. R. (2026a). *FERAL: A pure-Rust sparse symmetric indefinite linear solver with certified inertia counts*. GitHub. https://doi.org/10.5281/zenodo.20162687

Kitchin, J. R. (2026b). *POUNCE: A pure-Rust port of the Ipopt interior-point NLP solver*. GitHub. https://doi.org/10.5281/zenodo.21537291

Koch, J., Shapiro, M., Sharma, H., Vrabie, D., & Drgoňa, J. (2025). Learning neural differential algebraic equations via operator splitting. *2025 IEEE 64th Conference on Decision and Control (CDC)*, 2077–2084. https://doi.org/10.1109/CDC57313.2025.11312239

Lueg, L. R., Alves, V., Biegler, L. T., Kitchin, J. R., & Laird, C. D. (2026, November). A decomposition approach for scalable training of hybrid models via neural differential-algebraic equations. *2026 AIChE Annual Meeting*. https://studio.m-anage.com/aiche/2026/prelim.cgi/Paper/737989

Lueg, L. R., Alves, V., Schicksnus, D., Kitchin, J. R., Laird, C. D., & Biegler, L. T. (2026). A simultaneous approach for training neural differential-algebraic systems of equations. *Computational Optimization and Applications*. https://doi.org/10.1007/s10589-026-00823-y

Mahanty, B. (2023). Hybrid modeling in bioprocess dynamics: Structural variabilities, implementation strategies, and practical challenges. *Biotechnology and Bioengineering*, *120*(8), 2072–2091. https://doi.org/10.1002/bit.28503

Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., Freeman, J., Tsai, D., Amde, M., Owen, S., Xin, D., Xin, R., Franklin, M. J., Zadeh, R., Zaharia, M., & Talwalkar, A. (2016). MLlib: Machine learning in Apache Spark. *Journal of Machine Learning Research*, *17*(1), 1235–1241. https://doi.org/10.48550/arXiv.1505.06807

Moya, C., & Lin, G. (2022). DAE-PINN: A physics-informed neural network model for simulating differential algebraic equations with application to power networks. *Neural Computing and Applications*, *35*(5), 3789–3804. https://doi.org/10.1007/s00521-022-07886-y

DRAFT

Narayanan, H., Luna, M., Sokolov, M., Butté, A., & Morbidelli, M. (2022). Hybrid models based on machine learning and an increasing degree of process knowledge: Application to cell culture processes. *Industrial & Engineering Chemistry Research*, *61*(25), 8658–8672. https://doi.org/10.1021/acs.iecr.1c04507

Nicholson, B., Siirola, J. D., Watson, J.-P., Zavala, V. M., & Biegler, L. T. (2018). pyomo.dae: A modeling and automatic discretization framework for optimization with differential and algebraic equations. *Mathematical Programming Computation*, *10*(2), 187–223. https://doi.org/10.1007/s12532-017-0127-0

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830. https://doi.org/10.48550/arXiv.1201.0490

Psichogios, D. C., & Ungar, L. H. (1992). A hybrid neural network-first principles approach to process modeling. *AIChE Journal*, *38*(10), 1499–1511. https://doi.org/10.1002/aic.690381003

Rackauckas, C., Innes, M., Ma, Y., Bettencourt, J., White, L., & Dixit, V. (2019). DiffEqFlux.jl – a Julia library for neural differential equations. *arXiv Preprint arXiv:1902.02376*. https://doi.org/10.48550/arXiv.1902.02376

Rackauckas, C., Ma, Y., Martensen, J., Warner, C., Zubov, K., Supekar, R., Skinner, D., Ramadhan, A., & Edelman, A. (2021). Universal differential equations for scientific machine learning. *arXiv Preprint arXiv:2001.04385*. https://doi.org/10.48550/arXiv.2001.04385

Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2019). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, *378*, 686–707. https://doi.org/10.1016/j.jcp.2018.10.045

Rodriguez, J. S., Parker, R. B., Laird, C. D., Nicholson, B. L., Siirola, J. D., & Bynum, M. L. (2023). Scalable parallel nonlinear optimization with PyNumero and Parapint. *INFORMS Journal on Computing*, *35*(2), 509–517. https://doi.org/10.1287/ijoc.2023.1272

Stosch, M. von, Oliveira, R., Peres, J., & Feyo de Azevedo, S. (2014). Hybrid semi-parametric modeling in process systems engineering: Past, present and future. *Computers & Chemical Engineering*, *60*, 86–101. https://doi.org/10.1016/j.compchemeng.2013.08.008

Thompson, M. L., & Kramer, M. A. (1994). Modeling chemical processes using prior knowledge and neural networks. *AIChE Journal*, *40*(8), 1328–1340. https://doi.org/10.1002/aic.690400806

Wächter, A., & Biegler, L. T. (2006). On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, *106*(1), 25–57. https://doi.org/10.1007/s10107-004-0559-y

White, A., Kilbertus, N., Gelbrecht, M., & Boers, N. (2023). Stabilized neural differential equations for learning dynamics with explicit constraints. *Advances in Neural Information Processing Systems*, *36*, 12929–12950. https://doi.org/10.48550/arXiv.2306.09739

Xiao, T., Chen, Y., Huang, S., He, T., & Guan, H. (2023). Feasibility study of neural ODE and DAE modules for power system dynamic component modeling. *IEEE Transactions on Power Systems*, *38*(3), 2666–2678. https://doi.org/10.1109/tpwrs.2022.3194570

DRAFT