Differentiable composition for model discovery

Omer Rochman Sharabi
Montefiore Institute
University of Liège
o.rochman@uliege.be

Gilles Louppe
Montefiore Institute
University of Liège
g.louppe@uliege.be

Abstract

We propose DiffComp, a symbolic regressor that can learn arbitrary function compositions, including derivatives of various orders. We use DiffComp in conjunction with a Physics Informed Neural Network (PINN) to discover differential equations from data. DiffComp has a layered structure where a set of user-defined basic functions are composed up to a specified depth. As it is differentiable, it can be trained using gradient descent. We test the architecture using simulated data from common PDEs and compare to existing model discovery frameworks, including PySINDy and DeePyMoD. We then test on oceanographic data.

1 Introduction

Plenty of processes, many of them physical systems, can be modeled using differential equations; wave propagation, heat dissipation, the motion of celestial bodies, or car traffic are some examples. Usually, a scientist, using hard-to-come-by domain knowledge, designs a model and then runs simulations (ranging from pen and paper calculations to programs on supercomputers) which are later compared to real data. While this process works and has given us most of the models we use today, it has limitations. As we move on to more complex systems, it may be beneficial to be able to learn the dynamics that govern a given system from data, i.e. observations of the system.

Previous works approached the problem of model discovery from multiple angles [3, 16]. One approach is black box regression, where a black box model (usually some neural network) can be used to learn the dynamics of a system [9, 15, 17]. This approach, owing to the power of its black box approximators, allows us to learn complex dynamics. On the other hand, it suffers from certain drawbacks; most notably failure to predict new out-of-training-distribution data, and lack of human interpretability. Several works propose to mitigate those problems by learning a symbolic representation of the model dynamics. A black box model is used in [11] to learn the dynamics of bodies in the Solar System, then a tree-based symbolic regressor, PySR [7], is used to find a symbolic expression for the black box model, and finally that symbolic expression is fine tuned. Biggio et al. [2] propose a transformer-based symbolic regressor. Both and Kusters [5] introduce a fully differentiable model discovery algorithm based on PINNs [13] and Sparse Bayesian Learning (SBL) [6]. In the latter framework, one defines a library of possible terms that they think the model might include and then performs sparse linear regression on those terms.

Model discovery is usually framed as follows: we observe data generated by an unknown process, generally an ODE or a PDE, and want to discover the underlying dynamics of the process. Formally, we have ODEs: $t \in \mathbb{R}$, $u(t) \equiv u$, $0 = F(u, t)$; and PDEs: $t \in \mathbb{R}, \boldsymbol{x} \in D \subset \mathbb{R}^d$, $u(t, \boldsymbol{x}) \equiv u \in \mathbb{R}$, $0 = F(u, t, \boldsymbol{x})$. The unknown operator F may include partial derivatives of u with respect to $\boldsymbol{x}$ and t. It is the goal of model discovery to learn this F. In the following we will denote both $u(t, x)$ and $u(t)$ simply as u, write $\frac{du}{dt} \equiv \dot{u}$ and $\frac{\partial u}{\partial t} \equiv u_t$, and focus on the subproblem $u_t = F(u, t)$. Bold symbols are vectors, and in particular $\boldsymbol{u} \in \mathbb{R}^K$ is the vector $u(\{t_i, \boldsymbol{x}_i\}_1^K)$, where K is the number of observations or, during training, the batch size. Some known ODE and PDE examples from physics

Machine Learning and the Physical Sciences workshop, NeurIPS 2022.

are Newton's law $\ddot{r} = \frac{F(r)}{m}$, the heat equation $u_t = \Delta u$ and the wave equation $u_{tt} = c^2 \Delta u$, where the laplacian $\Delta u = \nabla \cdot (\nabla u)$. In this work, we build upon [5] and extend their model by substituting the SBL component with DiffComp, a fully differentiable architecture that can learn composite functions and is therefore not limited in its search by the user-defined library.

2 DiffComp

We combine DiffComp, a differentiable symbolic regressor, with PINNs, following [5] but substituting the SBL component. PINNs is a method to solve differential equations in which a NN ansatz is placed on the solution, making solving the equation equivalent to minimizing the PINN loss [13], which consists of a residual term, a boundary term, and an initial term. Another option, more suited to our context, is to constrain the equation using measured data. Having access to a dataset $\{\boldsymbol{u}_i, \boldsymbol{x}_i, t_i\}_1^N$ of measurements, the new loss becomes

$$\mathcal{L} = \frac{1}{N} \sum_i \|\partial_t \hat{\boldsymbol{u}}_i - F(\hat{\boldsymbol{u}}_i, t_i)\|^2 + \frac{1}{N} \sum_i \|\hat{\boldsymbol{u}}_i - \boldsymbol{u}_i\|^2, \tag{1}$$

where $\hat{\boldsymbol{u}}$ is our approximation of $\boldsymbol{u}$, which will usually be a neural network. Both and Kusters [5] do model discovery by adding an extra term to the loss, given by $\|\partial_t \hat{\boldsymbol{u}} - \boldsymbol{\Theta}\boldsymbol{\xi}\|^2$, where $\boldsymbol{\Theta}$ is a library of manually defined candidate functions and $\boldsymbol{\xi}$ a vector of coefficients. Using SBL, $\boldsymbol{\xi}$ can be made sparse. We substitute SBL with DiffComp.

DiffComp starts with a user-defined library of basic functions, and attempts to learn a composition of those functions to approximate some objective. The library is arranged in a layered graph. A layer consists of nodes that are fully connected to the previous layer via weighted edges. Each node contains one function from the user-defined library. During the forward pass, the outputs of the previous layers are weighted by the edges of a node and fed into the node as inputs to its function. By stacking layers we can compose complex functions even if the library contains only simple ones.

Mathematically, we are trying to learn F in $F(u, t) = u_t$. Let $\boldsymbol{z}_0 := [\hat{\boldsymbol{u}}, t, \boldsymbol{x}, ...] \in \mathbb{R}^n$ be the inputs, where $[\hat{\boldsymbol{u}}, t, \boldsymbol{x}, ...] \in \mathbb{R}^{M+M+M\times d+...=n}$ is a concatenation of $\hat{\boldsymbol{u}}$, t, $\boldsymbol{x}$ and possibly other inputs, such as terms known through domain expertise. Further, let $\boldsymbol{z}_k := f_k(\boldsymbol{\xi}_k \boldsymbol{z}_{k-1}) \in \mathbb{R}^{m_k}$ be the output of the k-th layer and M the total number of layers. The layers are defined component-wise as $\boldsymbol{z}_k^i = f_k^i(\boldsymbol{\xi}_k^i \boldsymbol{z}_{k-1}) \in \mathbb{R}$, where the f^is are user-defined functions such as derivatives, arithmetic operations or trigonometric functions, $\boldsymbol{\xi}_k^i \in \mathbb{R}^{m_{k-1}}$ are sparse learnable weights, and $\boldsymbol{z}_k = (z^1, ..., z^{m_k}) \in \mathbb{R}^{m_k}$. Each node selects, using $\boldsymbol{\xi}_k^i$, which components of the previous layer to use. In summary, the differentiable composition is given by

$$\hat{\boldsymbol{u}}, t, \boldsymbol{x} \to f_1^i(\boldsymbol{\xi}_1^i[\hat{\boldsymbol{u}}, t, \boldsymbol{x}]) = z_1^i \to \boldsymbol{z}_1 \to f_2^i(\boldsymbol{\xi}_2^i \boldsymbol{z}_1) = z_2^i \to \boldsymbol{z}_2 \to ... \to \boldsymbol{z}_M. \tag{2}$$

We remark that, in contrast to other frameworks, by virtue of the layered composition and PyTorch's [12] automatic differentiation, we can take derivatives not just of $\boldsymbol{u}$ but of any composition of functions i.e. $\frac{\partial f_k^i(\boldsymbol{\xi}_k^i \boldsymbol{z}_{k-1})}{\partial \boldsymbol{x}}$ (the $\boldsymbol{x}$ dependence is implicit in $\boldsymbol{z}_{k-1}$).

This architecture works best when learning a model where the dimensionality of the input x is high and the terms that appear in the real model are shallow compositions of library functions. The reason is that building a library containing all possible (shallow) terms combining the inputs becomes prohibitive for other methods, while ours can find those compositions. However, we expect the architecture to perform poorly when the terms that appear in the equation are a very deep composition of library functions, making domain knowledge necessary both for the method presented here and for other library-based methods. In that case, prior knowledge can be incorporated into the model. The model can also be used with a human in the loop that runs the model on pure inputs, selects relevant terms from the outputs of the model, and then uses those terms as inputs again to discover more complex terms. We have used this last training methodology with oceanographic data.

During training, we assume we observe points $\boldsymbol{x} \in \mathbb{R}$

3 Experiments

To evaluate the model discovery capacity of the DiffComp, we pick two well-known PDEs, generate datasets according to those equations, and attempt to learn them. This task can also be solved using

Method	Discovered equation	Runtime	MSE
2D heat equation: $u_t = u_{xx} + u_{yy}$			
PySINDy	$0.014u_{xx} + 0.014u_{yy}$	$< 1s$	0.05
DeePyMod	$0.255 + 0.556u_{xx} + 0.606u_{yy} - 0.627u - 0.201uu_{xx} - 0.239uu_{yy}$	$\sim$ 10min	9.38×10^{-5}
DiffComp	$0.971u_{xx} + 0.981u_{yy}$	$\sim$ 5.5min	4.01×10^{-5}
1D Burger's equation: $u_t = 0.05u_{xx} - uu_x$			
PySINDy	$0.05u_{xx} - 0.992uu_x$	$< 1s$	0.04
DeePyMod	$-0.336u_x + 0.198u_{xx} - 0.687uu_x$	$\sim$ 5min	1.57×10^{-4}
DiffComp	$0.05u_{xx} - 0.994uu_x$	$\sim$ 4min	2.09×10^{-6}
Ross et al. $(w_1\nabla^2 + w_2\nabla^4 + w_3\nabla^6)(\boldsymbol{u}\cdot\nabla)q + (w_4\nabla^4 + w_5\nabla^6)q + (\boldsymbol{u}\cdot\nabla)\nabla^2(w_6v_x + w_7v_y)$ [14]			
1st iteration	$w_1\nabla^2(\boldsymbol{u}\cdot\nabla)q + w_2\nabla^4 q$	$\sim$ 3min	0.996
2nd iteration	$w_1(\boldsymbol{u}\cdot\nabla)\nabla^4(\boldsymbol{u}\cdot\nabla)q + (\boldsymbol{u}\cdot\nabla)\nabla^4 q + w_3\nabla^6(\boldsymbol{u}\cdot\nabla)q + w_4\nabla^8 q$	$\sim$ 2.5min	0.764

Table 1: Comparison between methods. PySINDy runs almost instantly while DeePyMod and DiffComp have similar training times, on the order of 10min, as both involve training a neural network and taking its derivatives. Note that we have not optimized the parameters of PySINDy and DeePyMod.

existing library-based methods, such as PySINDy [8, 10] and DeepMoD [4]. We also test DiffComp on a more complex dataset of oceanographic data . Our library contained the following functions: addition, multiplication, and differentiation in the PDE tests, and addition, laplacian and advection in the oceanographic benchmark. A summary is presented in Table 1. Some of the tests were repeated with added noise and the results are shown in Table 2 in the appendix.

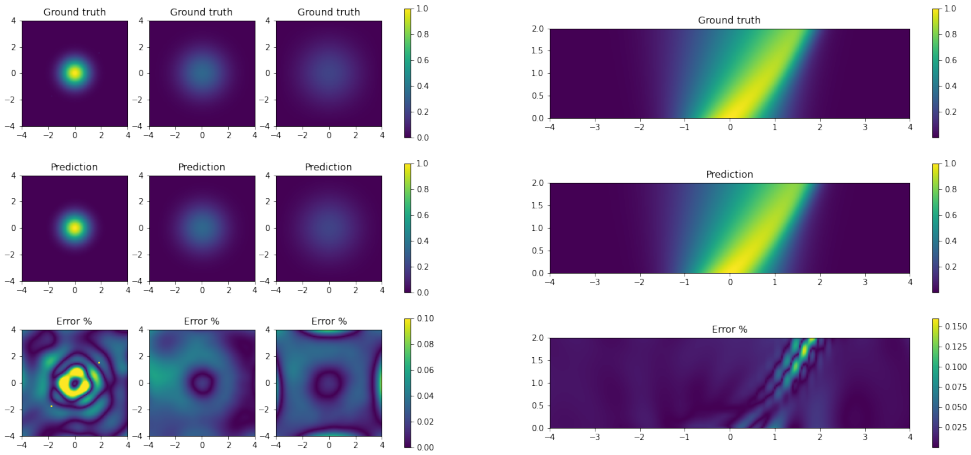
The heat equation, $u_t = u_{xx} + u_{yy}$ is a diffusion equation describing how a quantity diffuses in a medium over time. Our model finds the equation $u_t = 0.971u_{xx} + 0.981u_{yy}$. We notice in this case, in the case of Burger's equation, and in general, that the coefficients found are slightly below their true value. This is due to the effects of the regularization term and the fact that neither the data nor the network fulfill perfectly the equation. Figure 1a shows snapshots of the solution.

The Burger's equation $u_t = \nu u_{xx} - uu_x$ is a second order and non-linear PDE, when ν is zero, shockwaves can appear. We solve the equation for a value of ν = 0.05 and present the results in Fig. 1b. The equation our model finds is $u_t = 0.05u_{xx} - 0.994uu_x$.

High resolution oceanic simulations are very expensive, taking months to run sometimes, and low resolution simulations do not capture all the important interactions. A proposed solution to this problem is to use a forcing term, which when added to the low resolution simulation, improves its quality. Ross et al. [14] compare both symbolic and black box methods to learn the forcing term. We use a similar methodology, but instead of genetic programming we use DiffComp. Owing to the complexity of the data, our pipeline of training a neural network and taking its derivatives did not work because the neural network failed to converge. However, as [14] have implemented numerically the differential operations, we use their implementation and therefore do not need to use a network to compute derivatives.

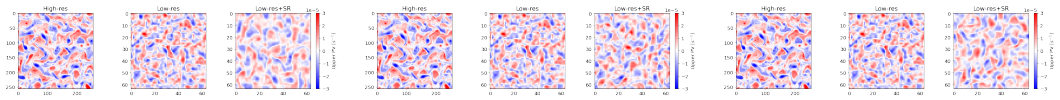
We adapt the training method used by [14], and incorporate a human in the loop. We use DiffComp to discover symbolic terms, but instead of relying solely on gradient descent like before, we learn the terms of $\boldsymbol{z}_{M-1}$ by gradient descent, and use the least squares method like [14] to get a linear combination of those terms, that is, the coefficents $\boldsymbol{\xi}_M$ in $\boldsymbol{z}_M = \boldsymbol{\xi}_M \boldsymbol{z}_{M-1}$. This is because the weights of the terms have very different orders of magnitude, which a gradient descent algorithm cannot capture. A human then selects the most meaningful terms of the expression, based on leading terms and ablation (removing terms and evaluating impact on performance). Those terms are then fed as added input to DiffComp to reduce the complexity of the composition that has to be discovered. Additionally, following the loss function of [14], instead of minimising the MSE between $\hat{\boldsymbol{u}}$ and $\boldsymbol{u}$, we maximise the correlation between them, again, because the wildly different orders of magnitude involved. We run this procedure three times and obtain progressively more complex expressions. Using as inputs q, u, v we obtain the first expression $w_1\nabla^2(\boldsymbol{u}\cdot\nabla)q + w_2\nabla^4 q$. Then, the three terms from the first input plus the two terms found as new inputs, we find the second expression $w_1(\boldsymbol{u}\cdot\nabla)\nabla^4(\boldsymbol{u}\cdot\nabla)q + (\boldsymbol{u}\cdot\nabla)\nabla^4 q + w_3\nabla^6(\boldsymbol{u}\cdot\nabla)q + w_4\nabla^8 q$. The combined symbolic expression (a sum of those two equation) is the final model. For comparison, Ross et al. [14] find

$$(w_1\nabla^2 + w_2\nabla^4 + w_3\nabla^6)(\boldsymbol{u}\cdot\nabla)q + (w_4\nabla^4 + w_5\nabla^6)q + (\boldsymbol{u}\cdot\nabla)\nabla^2(w_6v_x + w_7v_y). \quad (3)$$



(a) 2D heat equation at $t = 0$, $t = 0.5$, and $t = 1$ (b) 1D Burger's equation, y-axis is the time axis

Figure 1: Ground truth (top), prediction (middle), and error percentage (bottom)



(a) 1st expression discovered (b) 2nd expression discovered (c) Combined expression

Figure 2: Comparison between simulation at high resolution, low resolution and low resolution plus the added forcing term

The third iteration yields an unwieldy expression that is over-fitted. It is shown in the appendix (Eq. 4), along with other evaluation metrics from [14].

4 Conclusion

We introduced a differentiable symbolic regressor to be used with a differentiable surrogate, e.g. a neural network, to discover an unknown model directly from data. While DiffComp can also be used as a pure symbolic regressor, we note that its performance will be lower than state-of-the-art genetic and tree regressors, such as PySR [7] and gplearn [1]. We tested our method on two differential equations as well as a real-world benchmark from oceanography, and compared it to other methods. We found it competitive, with the disadvantage of being slower than PySINDy but the advantage of able to learn functions compositions of a certain depth, allowing for the discovery of terms that are not in the original library. Because it uses gradient descent to find the terms, DiffComp can seamlessly find parameters of the terms, like in Burger's equation, but struggles if the data spans too many orders of magnitude. Lastly, a few open questions remain; as the gradient dynamics are different from those in neural networks, are optimizers designed for said networks optimal? How to handle wildly different data scales, like in the forcing term example? How to make DiffComp better at finding deeper compositions?

Impact statement: This work aims to be a minor step towards more automation in science. As such, we believe that the societal impact of the method introduced will be minor, but positive. Additionally, at this stage of development we see no ethical concerns.

Acknowledgements

This work was partially supported by ServicePublic de Wallonie Recherche under grant n° 2010235 – ARIAC by DIGITALWALLONIA4.AI. Gilles Louppe is recipient of the ULiège - NRB Chair on Big Data and is thankful for the support of the NRB.

References

[1] gplearn: Genetic Programming in Python, with a scikit-learn inspired API. `https://github.com/trevorstephens/gplearn`. Accessed: 2022-09-26.

[2] L. Biggio, T. Bendinelli, A. Neitz, A. Lucchi, and G. Parascandolo. Neural symbolic regression that scales. *CoRR*, abs/2106.06427, 2021.

[3] J. Bongard and H. Lipson. Automated reverse engineering of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 104(24):9943–9948, 2007.

[4] G.-J. Both, S. Choudhury, P. Sens, and R. Kusters. Deepmod: Deep learning for model discovery in noisy data. *Journal of Computational Physics*, 428:109985, 2021.

[5] G.-J. Both and R. Kusters. Fully differentiable model discovery, 2021.

[6] S. L. Brunton, J. L. Proctor, and J. N. Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 113(15):3932–3937, 2016.

[7] M. Cranmer. Pysr: Fast & parallelized symbolic regression in python/julia, Sept. 2020.

[8] B. de Silva, K. Champion, M. Quade, J.-C. Loiseau, J. Kutz, and S. Brunton. Pysindy: A python package for the sparse identification of nonlinear dynamical systems from data. *Journal of Open Source Software*, 5(49):2104, 2020.

[9] N. Geneva and N. Zabaras. Transformers for modeling physical systems. *CoRR*, abs/2010.03957, 2020.

[10] A. A. Kaptanoglu, B. M. de Silva, U. Fasel, K. Kaheman, A. J. Goldschmidt, J. Callaham, C. B. Delahunt, Z. G. Nicolaou, K. Champion, J.-C. Loiseau, J. N. Kutz, and S. L. Brunton. Pysindy: A comprehensive python package for robust sparse system identification. *Journal of Open Source Software*, 7(69):3994, 2022.

[11] P. Lemos, N. Jeffrey, M. Cranmer, S. Ho, and P. Battaglia. Rediscovering orbital mechanics with machine learning, 2022.

[12] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[13] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics informed deep learning (part ii): Data-driven discovery of nonlinear partial differential equations, 2017.

[14] A. S. Ross, Z. Li, P. Perezhogin, C. Fernandez-Granda, and L. Zanna. Benchmarking of machine learning ocean subgrid parameterizations in an idealized model. *Earth and Space Science Open Archive*, page 43, 2022.

[15] A. Sanchez-Gonzalez, J. Godwin, T. Pfaff, R. Ying, J. Leskovec, and P. W. Battaglia. Learning to simulate complex physics with graph networks, 2020.

[16] M. Schmidt and H. Lipson. Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85, 2009.

[17] S. Seo and Y. Liu. Differentiable physics-informed graph networks, 2019.

Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default **[TODO]** to [Yes] , [No] , or [N/A] . You are strongly encouraged to include a **justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [Yes] See Section **??**.
- Did you include the license to the code and datasets? [No] The code and the data are proprietary.
- Did you include the license to the code and datasets? [N/A]

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

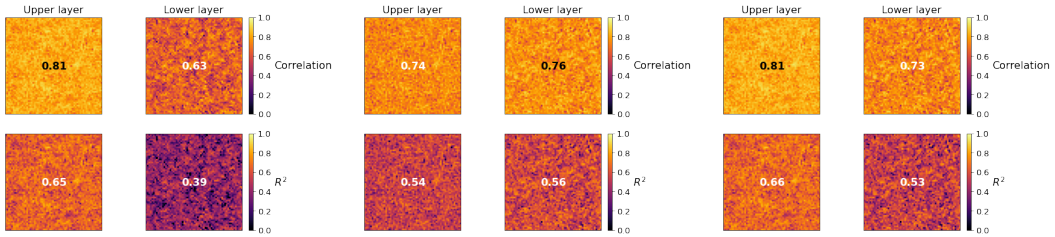
1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [Yes]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [N/A]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [N/A]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Appendix

All the presented in this work were ran on a Thinkpad P1, with an 11th Gen Intel(R) Core(TM) i7-11800H and 16GB of RAM. No GPUs were used. We rerun the experiments from Table 1 with added noise and show the results in Table 2. We also present further online 4 and offline metrics 3 from Ross et al. [14] to further test the symbolic expression found for the forcing term, and present the resulting term from the last iteration of training in 4.

Method	Discovered equation	runtime	MSE
2D heat equation: $u_t = u_{xx} + u_{yy}$			
PySINDy	$-2.587u^3$	$< 1s$	0.02
DeePyMod	$0.136u_{xx} - 0.879u$	$\sim$ 8min	9.76×10^{-2}
DiffComp	$0.931(-0.168x + 0.316y)(-1.893u_x + 2.441u_y - 0.005) + 0.53u_{xx} + 1.01u_{yy} - 1.471u_{xy} + 0.06$	$\sim$ 14min	3.43×10^{-5}
1D Burger's equation: $u_t = 0.05u_{xx} - uu_x$			
PySINDy	$-0.913uu_x$	$< 1s$	1.60×10^{-5}
DeePyMod	$-0.292u_x + 0.205u_{xx} + 0.741uu_x$	$\sim$ 13min	3.23×10^{-3}
DiffComp	$0.049u_{xx} - 0.998uu_x$	$\sim$ 7min	2.70×10^{-6}

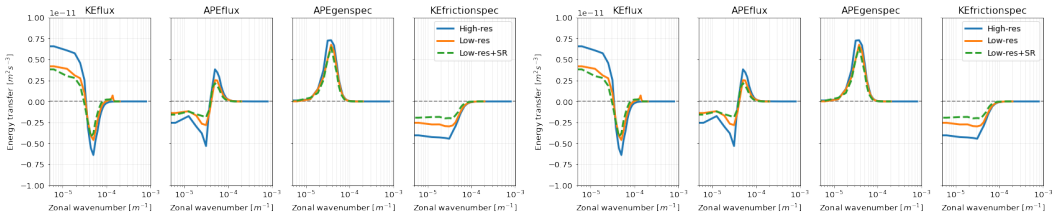
Table 2: The noise was created by sampling from a gaussian with mean 0 and std equal to 0.01 of the norm of the initial condition $u(0, \boldsymbol{x})$. PySINDy's WeakPDEFIND was used here



(a) 1st expression (b) 2nd expression (c) Combined expression

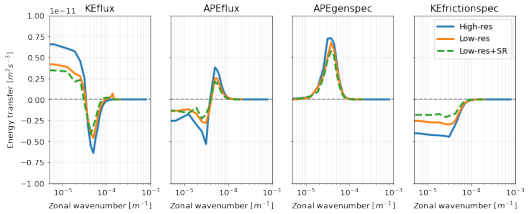
Figure 3: Offline metrics measured between the target forcing term and the discovered symbolic expression. Note that the correlation was the training target

$$
\begin{aligned}
&w_1(\boldsymbol{u}\cdot\nabla)\big(\nabla^2\big(\nabla^2((\boldsymbol{u}\cdot\nabla)(q))\big)\big) + w_{12}(\boldsymbol{u}\cdot\nabla)\big(w_8(\boldsymbol{u}\cdot\nabla)\big(\nabla^2\big(\nabla^2(q)\big)\big)\big)\\
&+ w_3\nabla^2\big(w_0(\boldsymbol{u}\cdot\nabla)\big(w_8(\boldsymbol{u}\cdot\nabla)\big(\nabla^2\big(\nabla^2(q)\big)\big)\big) + w_{11}\nabla^2\big(w_{13}(\boldsymbol{u}\cdot\nabla)\big(\nabla^2\big(\nabla^2(q)\big)\big)\big)\big)\\
&+ w_4(\boldsymbol{u}\cdot\nabla)\big(w_{10}\nabla^2\big(w_{13}(\boldsymbol{u}\cdot\nabla)\big(\nabla^2\big(\nabla^2(q)\big)\big)\big) + w_2(\boldsymbol{u}\cdot\nabla)\big(w_8(\boldsymbol{u}\cdot\nabla)\big(\nabla^2\big(\nabla^2(q)\big)\big)\big)\big)\\
&+ w_5\nabla^2\big(\nabla^2\big(\nabla^2\big(\nabla^2(q)\big)\big)\big) + w_6(\boldsymbol{u}\cdot\nabla)\big(\nabla^2\big(\nabla^2(q)\big)\big)\\
&+ w_7\nabla^2\big(w_{13}(\boldsymbol{u}\cdot\nabla)\big(\nabla^2\big(\nabla^2(q)\big)\big)\big) + w_9\nabla^2\big(\nabla^2\big(\nabla^2((\boldsymbol{u}\cdot\nabla)(q))\big)\big)
\end{aligned}
\tag{4}
$$



(a) 1st expression

(b) 2nd expression



(c) Combined expression

Figure 4: Comparison between the energy budgets of the high resolution simulation, the low resolution simulation, and the low resolution simulation including the discovered forcing terms. Just like [14] find, we see that despite the high correlation there is no improvement in new low resolution simulation. We believe that a human in the loop procedure where the human is a domain expert and applied domain knowledge when selecting the relevant terms will improve the results, as was done in [14]