Predicting Ordinary Differential Equations with Transformers

Sören Becker [1] **Michal Klein** [2 *] **Alexander Neitz** [3 †] **Giambattista Parascandolo** [4 †] **Niki Kilbertus** [1 5]

Abstract

We develop a transformer-based sequence-to-sequence model that recovers scalar ordinary differential equations (ODEs) in symbolic form from irregularly sampled and noisy observations of a single solution trajectory. We demonstrate in extensive empirical evaluations that our model performs better or on par with existing methods in terms of accurate recovery across various settings. Moreover, our method is efficiently scalable: after one-time pretraining on a large set of ODEs, we can infer the governing law of a new observed solution in a few forward passes of the model.

1. Introduction

Researchers in the natural sciences increasingly turn to machine learning (ML) to aid the discovery of natural laws from observational data alone, which is often abundantly available, hoping to bypass expensive and cumbersome targeted experimentation. While there may be fundamental limitations to what can be extracted from observations alone, recent successes of ML provide ample reason for excitement. Partially fueled by these promises, interest in symbolic regression (SR) has received renewed attention (La Cava et al., 2021; Makke & Chawla, 2022). A symbolic representation of a law has several advantages over black-box representations in that they are typically parsimonious and directly interpretable as well as amenable to analytic analysis.

Numerous symbolic regression methods have been proposed recently to infer functional relationships, i.e., to infer a function f symbolically given (noisy) examples $(x_i, f(x_i))_{i=1}^n$ (La Cava et al., 2021). Arguably, a more interesting—but also more challenging—task is to infer dynamical laws. We represent a dynamical law as an ODE $\dot{y} := {}^{dy}/_{dt} = f(y)$, which is fully determined by f. In this setting, the goal is to infer f from (noisy, irregular) samples $(t_i, y_i)_{i=1}^n$, where y is a solution of the ODE (and y_i denotes the observed value for $y(t_i)$). Compared to symbolic regression for functional relationships, relatively little work exists on directly inferring dynamical laws (see Section 2 for details). In principle, any functional symbolic regression method may be applied to $(y_i, \hat{\dot{y}}_i)_{i=1}^n$, where $\hat{\dot{y}}_i$ are estimated derivatives at the observed time points t_i, to obtain f. This naturally raises the question whether methods tailored to directly inferring dynamical laws yields better results.

In this work, we develop **N**eural **S**ymbolic **O**rdinary **D**ifferential **E**quation (**NSODE**), a transformer based sequence-to-sequence model specifically tailored to inferring dynamics directly in an end-to-end fashion from (t_i, y_i) samples of a single solution trajectory. NSODE leverages large scale pre-training for efficient inference at test time. We first (randomly) generate a total of >3M scalar, autonomous, non-linear, first-order ODEs, together with a total of >63M numerical solutions from various (random) initial conditions. All solutions are carefully checked for convergence of the numerical integration. Code and data are publicly available at `https://github.com/soerenab/nsode23`.

NSODE, an encoder-decoder transformer, is then trained in a supervised fashion to map observed trajectories, i.e., numeric sequences of the form $(t_i, y_i)_{i=1}^n$, directly to symbolic equations as strings, e.g., `"y**2+1.64*cos(y)"`, which is the prediction for f. This example directly highlights the benefit of symbolic representations in that the y^2 and $\cos(y)$ terms tell us something about the fundamental dynamics of the observed system; the constant `1.64` will have semantic meaning in a given context. NSODE combines and innovates on technical advances regarding input representations and an efficiently optimizable loss formulation. Our model outperforms most existing methods and is more efficient at inference time than models with competitive performance.

[*]Work done while at Helmholtz Center Munich. [†]Work done while at Max Planck Institute for Intelligent Systems. [1]Helmholtz AI, Helmholtz Center Munich, Munich, Germany. [2]Apple, Paris, France. [3]DeepMind, London, United Kingdom. [4]OpenAI, San Francisco, United States. [5]Technical University of Munich, Germany. Correspondence to: Sören Becker <soeren.becker@helmholtz-munich.de>, Niki Kilbertus <niki.kilbertus@helmholtz-munich.de>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023.

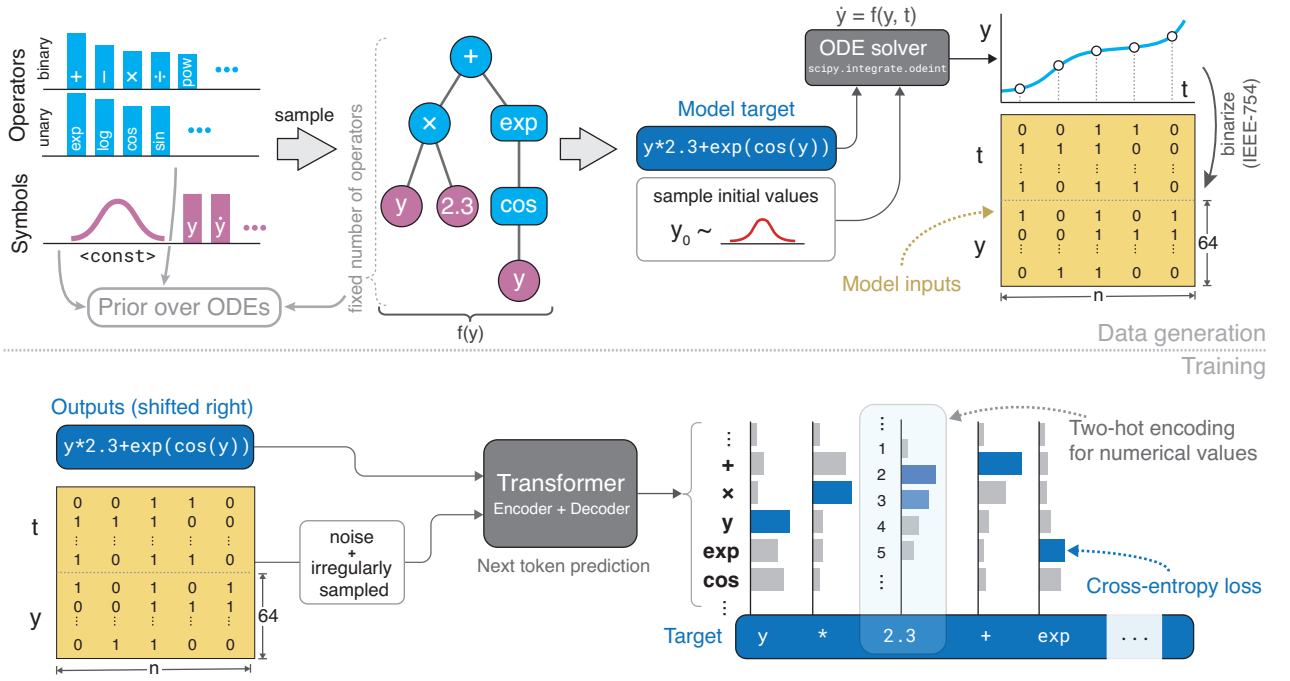


Figure 1: An overview illustration of the data generation (top) and training pipeline (bottom). In the lower right, blue bars correspond to one-hot and two-hot target encodings whereas gray bars correspond to model predictions. Our dataset stores solutions in numerical (non-binarized) form on the entire regular solution time grid.

2. Background and Related Work

Modeling dynamics and forecasting their behavior has a long history in machine learning. While Neural Ordinary Differential Equations (NODE) (Chen et al., 2018) (with a large body of follow up work) are perhaps the most prominent modern approach from the recent years, their inherent black-box character complicates interpretability or scientific understanding of the observed phenomena. Recent alternatives such as tractable dendritic RNNs (Brenner et al., 2022) and Neural Operators (Kovachki et al., 2021; Li et al., 2021) set out to facilitate scientific discovery by combining accurate predictions with improved interpretability. A considerable advantage of these and similar approaches is their scalability to high dimensional systems as well as their relative robustness to noise, missing or irregularly sampled data (Iakovlev et al., 2021) and challenging properties such as multiscale dynamics (Vlachas et al., 2022) or chaos (Park et al., 2022; Patel & Ott, 2022). The recent benchmark study by Gilpin (2021) provides an excellent overview of dynamics forecasting models including deep learning-based blackbox approaches as well as symbolic models.

However, in this work we focus on models that explicitly predict a mathematical expression in symbolic form because of the advantages of their interpretable and parsimonious representations. We now provide an overview of the most prominent types of symbolic regression techniques. Our overview also includes methods that have been primarily designed to infer functional relationships only, i.e., that are not specifically tailored to inferring dynamical laws.

Evolutionary algorithms. Traditionally, many approaches to symbolic regression broadly fall into the category of evolutionary algorithms such as genetic programming (GP) (Koza, 1993). Genetic programming randomly evolves a population of prospective mathematical expressions over multiple iterations and mimics natural selection by keeping only the best contenders across iterations, where superiority is measured by user-defined fitness functions (Schmidt & Lipson, 2009). Process-based modeling follows a similar approach but includes domain knowledge-informed constraints on particular components of the system in order to reduce the search space to reasonable candidates (Todorovski & Dzeroski, 1997; Bridewell et al., 2008; Simidjievski et al., 2020). A large body of work examines and improves upon all aspects of GP to overcome typical shortcomings such as premature convergence, overly complex output expressions, scalability (both in terms of time as well as memory), or the difficulty of incorporating prior knowledge (Schmidt & Lipson, 2010; Arnaldo et al., 2014; La Cava et al., 2016; Virgolin et al., 2017; La Cava et al., 2018; Virgolin et al., 2019; Burlacu et al., 2020; de Franca & Aldeia, 2021; Tohme et al., 2022; Mundhenk et al., 2021).

Regression (gradient-based methods). More recently,

symbolic regression has been approached via machine learning methods which exploit gradient information to optimize within the space of (finite) compositions of pre-defined basis functions. Brunton et al. (2016) builds on sparse linear regression to identify a linear combination of basis functions. This approach has inspired a large body of follow-up work generalizing the idea to partial observations (Bakarji et al., 2022), parameterized functions (Lejarza & Baldea, 2022), simultaneous discovery of coordinates (Champion et al., 2019), coordinate transformations that linearize the dynamics (Lusch et al., 2018), and partial differential equations (Rudy et al., 2017) among others. Similarly, McConaghy (2011) use path-wise regularized learning with ElasticNets on a large body of pre-generated non-linear basis functions for symbolic regression. These techniques often deploy sparsity-promoting regularizers and train one or multiple models for each set of observations. Once trained, the model itself represents the predicted symbolic expression, which can be read off the non-zero coefficients. This modeling principle is also employed by many other approaches that replace linear regression by neural networks with diverse sets of activation functions, both for differential equations (Long et al., 2019; Liu et al., 2020) and non-differential algebraic equations (Sahoo et al., 2018).

Hybrid models. Supervised learning with gradient-based optimization to directly output the symbolic expression (e.g., as a string) is challenged by the formulation of a differentiable loss between the predicted symbolic expression (string) and the observed data (numerical). Thus, prior work on functional symbolic regression resorted to reinforcement learning (Petersen et al., 2021; Landajuela et al., 2021) or combinations of neural networks and evolutionary algorithms (Atkinson et al., 2019; Costa et al., 2021). A hybrid approach combining gradient-free, human intuition-guided heuristic search via genetic programming with neural network-based optimization has been presented for non-differential equations by Udrescu et al. (2020). This method proceeds by a divide-and-conquer strategy in applying the hand-tuned heuristics. It has recently been extended to dynamical systems by Weilbach et al. (2021).

Monte Carlo methods. Finally, Jin et al. (2019); Brence et al. (2021) and extensions such as Gec et al. (2022) incorporate prior knowledge by flexibly specifying distributions over the allowed function space. This also allows them to perform Bayesian updates and ultimately equation discovery via Monte Carlo sampling from the (posterior) distribution.

Sequence-to-sequence models. The closest works to ours use pre-trained, attention-based sequence-to-sequence models for symbolic regression *of functional relationships* (Biggio et al., 2021; Valipour et al., 2021; Kamienny et al., 2022; Vastl et al., 2022) or (discrete) recurrence relations (D'Ascoli et al., 2022). They exploit the fact that symbolic expressions for (multi-variate) scalar functions can be both generated and evaluated on random inputs cheaply, resulting in essentially unlimited labeled training data that allows for gradient-based optimization using the cross-entropy loss on the symbol level (instead of numerical proximity between evaluations of the true and predicted functions). Our model differs in a number of key innovations described in Section 3.2 overcoming limitations of existing methods and rendering it suitable for inferring dynamical laws directly.

3. Method

Many previous symbolic regression methods have been described as "discovering natural laws". However, most of them learn *fixed functional relationships* from input-output pairs, whereas we seek to actually infer *the underlying dynamic law* that governs the behavior of the observed solution trajectory directly. One way to still apply functional SR to dynamics is to approximate derivatives $\hat{\dot{y}}_i$ from the observed data $(y_i)_{i=1}^n$ and use $(y_i, \hat{\dot{y}})_{i=1}^n$ to infer f (as in $\dot{y} = f(y)$). Since temporal derivatives are usually not measured directly, this approach crucially depends on the derivative estimation, typically via finite difference approximations. Even though higher-order finite difference methods can be extended to irregularly sampled and noisy observations, naturally the question arises whether methods specifically tailored to inferring dynamics directly may be superior. Qian et al. (2022) have also recently identified alternative loss formulations that bypass unobserved time derivatives as an open challenge in symbolic regression for dynamical systems. With NSODE, we propose a solution to this challenge.

Problem setting. Given noisy observations $\{(t_i, y_i)\}_{i=1}^n$ of a trajectory $y : [t_1, t_n] \to \mathbb{R}$ that is a solution of the scalar ODE $\dot{y} = f(y)$, we aim to recover the function f in symbolic form. In this formulation, we explicitly assume that the observed system actually evolves according to an ODE in canonical form $\dot{y} = f(y)$ such that f can be expressed in closed form using the mathematical operators seen during training (see Section 3.1 for details).

While solutions to the class of ODEs considered in this work are known to have relatively simple limiting behaviors (essentially either blowing up or reaching a constant equilibrium), within finite time they still exhibit rich and varied behavior. We show some examples in Appendix F.

3.1. Data Generation

Sampling symbolic expressions. To exploit large-scale supervised pretraining, we randomly generate a dataset of $\sim$63M ODEs in symbolic form along with their numerical solutions for multiple randomly sampled initial values. Since we assume ODEs to be in canonical form $\dot{y} = f(y)$, generating an ODE is equivalent to generating a symbolic

expression $f(y)$. We follow Lample & Charton (2019), who sample such an expression $f(y)$ as a unary-binary tree, where each internal node corresponds to an operator and each leaf node corresponds to a constant or variable. The algorithm consists of two phases: (1) A unary-binary tree is sampled uniformly from the distribution of unary-binary trees with up to $k \in \mathbb{N}$ internal nodes, which crucially does not overrepresent small trees corresponding to short expressions. Here the maximum number of internal nodes K is a hyperparameter of the algorithm. (2) The sampled tree is "decorated", that is, each binary internal node is assigned a binary operator, each unary internal node is assigned a unary operator, and each leaf is assigned a variable or constant. Hence, we have to specify a distribution over the N_{bin} binary operators, a distribution over the N_{una} unary operators, a probability $p_{\text{sym}} \in (0, 1)$ to decide between symbols and constants for leaf nodes, as well as a distribution p_{c} over constants. For constants, in NSODE we further distinguish explicitly between sampling an integer or non-integer value. Together with K, these choices uniquely determine a distribution over equations f and are described in detail in Appendix A. The top part of Figure 1 depicts an overview of the data generation procedure.

The pre-order traversal of a sampled tree results in the symbolic expression for f in prefix notation. After conversion to the more common mathematical infix notation, we simplify each expression using the computer algebra system SymPy (Meurer et al., 2017), and filter out constant equations $f(y) = c$ as well as expressions that contain operators or symbols that were not in the support of the original distribution.[1] We call the structure modulo the value of the constants of such an expression (i.e., replacing all actual constant values by a generic `<const>` token) a **skeleton**.

Many skeletons can be represented by different unary-binary trees and hence many of the generated trees will be simplified to the same skeleton. To ensure diversity and to mitigate potential dataset bias towards particular expressions, we discard duplicates on the skeleton level. To further cheaply increase the variability of ODEs we sample N_{const} unique sets of constants per skeleton. When sampling constants we take care not to modify the canonical expression by adhering to the rules listed in Appendix A.1. Our final dataset contains linear and non-linear as well as homogeneous and inhomogeneous ODEs and we provide summary statistics about the distribution over equations in Appendix C. Besides the number of internal nodes, a simple yet common measure of **complexity** for each symbolic equation is the overall count of symbols (e.g., y, or constants) as well as operators. We follow previous works on symbolic regression in using this complexity measure in our empirical evaluation and refer to La Cava et al. (2021) for an overview of proposed alternatives.

[1] With the exception of a unary $-$, which we do not discard.

Table 1: Overview of our model architecture.

	Encoder	Decoder
layers	6	6
heads	16	16
embed. dim.	512	512
forward dim.	2048	2048
activation	gelu	gelu
vocab. size	-	43
position enc.	learned	learned
parameters	23.3M	23.3M

Computing numerical solutions. We obtain numerical solutions for all generated initial value problems via SciPy's interface (Virtanen et al., 2020) to the LSODA software package (Hindmarsh & Laboratory, 1982) with both relative and absolute tolerances set to 10^{-9}. LSODA consists of a collection of ODE solvers and implements a strategy to automatically choose an appropriate solver for the problem at hand (e.g., recognizing stiff problems). We solve each equation on a fixed time interval $t \in [0, T]$ and store solutions on a regular grid of N_{grid} points. For each ODE, we sample up to N_{iv} initial values $y(0) = y_0$ uniformly from $(y_0^{\min}, y_0^{\max})$.[2] While LSODA attempts to select an appropriate solver, numerical solutions still cannot be trusted in all cases. Therefore, we check the validity of solutions via the following quality control check: we use 9th order central finite differences to approximate the temporal derivative of the solution trajectory (on the same equidistant temporal grid as the proposed solution and without adding noise), denoted by $\dot{y}_{\text{fd}}$, and filter out any solution for which $\|\dot{y}_{\text{fd}} - \dot{y}\|_\infty > \epsilon$, where we use $\epsilon = 1$.

3.2. Model Design Choices and Training

NSODE consists of an encoder-decoder transformer with architecture choices listed in Table 1. A visual overview of the training is depicted in Figure 1.

Representing input trajectories. A key difficulty in feeding numerical observations $(y_i)_{i=1}^n$ as input sequence to a transformer is that their range may differ greatly both within a single solution as well as across ODEs. For example, the linear ODE $\dot{y} = c \cdot y$ for a constant c is solved by an exponential $y(t) = y_0 \exp(ct)$ for initial value $y(0) = y_0$, which may span many orders of magnitude on a fixed time interval. To prevent numerical errors and vanishing or exploding gradients caused by the large range of values, we assume each representable 64-bit float value is a token and

[2] Due to a timeout per ODE, fewer solutions may remain if the solver fails for repeated initial value samples.

use its IEEE-754 encoding as the token representation (Biggio et al., 2021). We thus convert all pairs (t_i, y_i) to their IEEE-754 64 bit representations, channel them through a linear layer, and then feed them to the encoder. The linearly transformed bit pattern hence replaces the explicit embedding layer that commonly preceeds the encoder.

Representing symbolic expressions. The target sequence (i.e., the string for the symbolic expression of f) is tokenized on the (mathematical) symbol-level. For all operators and variables we include separate unique tokens in the vocabulary. These tokens are one-hot encoded and passed through a learnable embedding layer before their embedded representations are fed to the decoder.

Constants (as in fixed numerical values) play a special role in sequence-to-sequence approaches to symbolic regression. While the cross-entropy loss works well for discrete, one-hot encoded operators and symbols (e.g. `+,exp,sin,x,y`), one cannot directly add all possible constant values such as `1.452` to the vocabulary as separate tokens. Naively tokenizing on the digit level, i.e., representing real values literally as the sequence of characters (e.g., `"1,.,4,5,2"`), not only significantly expands the length of target sequences and thus the computational cost, but also requires a variable number of prediction steps for every single constant. As a workaround previous works on *functional SR* resort to one of two strategies: (1) represent all constants with a special `<const>` token and optimize their actual values in a separate fine-tuning step. (2) round constants to a finite number of possible values, which can then all be represented as individual tokens.

The second optimization of strategy (1) comes at a substantial computational cost as constants have to be fit per inferred expression. For efficiency and scalability, we would like the sequence-to-sequence model propose a complete equation, including values for the involved constants. Even more detrimental to our problem setting, this approach does not transfer to inferring ODEs: to optimize constants via a regression loss, one would first have to solve the predicted ODE $\dot{y} = \hat{f}(y)$ to obtain predicted $\{\hat{y}_i\}_{i=1}^n$ values that can be compared to the set of observations $\{y_i\}_{i=1}^n$. That is, the objective function to be optimized when fine-tuning constants involves solving an ODE. While differentiable ODE solvers exist, optimizing constants per inferred expression this way is prohibitively expensive and typically highly unstable. Even though strategy (2) avoids a separate optimization step and can leverage clever encoding schemes with improved token efficiency, it comes with an inherent loss of precision.

Therefore, we propose the following representation of constant values. Taking inspiration from Schrittwieser et al. (2020), we encode constants in a *two-hot* fashion. We fix a finite homogeneous grid on the real numbers $x_1 < x_2 < \ldots < x_m$ for some $m \in \mathbb{N}$ and add those values as tokens to the vocabulary. The range of integers, the grid range (x_1, x_m), and number of grid points m are hyperparameters that can be tuned for performance. Our choices are described in Appendix A.3. For any constant c in the target sequence we then find $i \in \{1, \ldots, m-1\}$ such that $x_i \leq c < x_{i+1}$ and encode c as a distribution supported on x_i, x_{i+1} with weights α, β such that $\alpha x_i + \beta x_{i+1} = c$. That is, the target in the cross-entropy loss for a constant token is not a strict one-hot encoding, but a distribution supported on two (neighboring) vocabulary tokens resulting in a lossless encoding of continuous values in $[x_1, x_m]$ which does not require rounding. While this two-hot representation can be used directly in the cross-entropy loss function and thus greatly facilitates training, it can not be passed directly through an embedding layer. For a generic constant in the target sequence represented as $\alpha x_i + \beta x_{i+1}$, we thus instead provide the linear combination of the two embeddings α `embed(x_i)` $+ \beta$ `embed(x_i+1)` as decoder input.

Decoding constants. When decoding a predicted sequence, we check at each prediction step whether the $\arg\max$ of the logits corresponds to one of the m constant tokens $\{x_1, \ldots, x_m\}$. If not, we proceed by conventional one-hot decoding to obtain predicted operators and variables. If instead the argmax corresponds to, for example, x_i, we also pick its largest-logit neighbor (x_{i-1} or x_{i+1}; suppose x_{i+1}), renormalize their probabilities by applying a softmax to all logits and use the resulting two probability estimates as weights α, β. Constants are then ultimately decoded as $\alpha x_i + \beta x_{i+1}$. We depict our decoding scheme in Figure 1.

Sampling solutions. To infer a symbolic expression for the governing ODE of a new observed solution trajectory $\{(t_i, y_i)\}_{i=1}^n$, all the typical policies such as greedy, sampling, or beam search are available. In our evaluation, we leverage computationally cheap forward passes to perform a beam search with 1536 beams. We provide details about how NSODE is evaluated in Section 4.3.

Training. We train two versions of our model. **NSODE** is trained on $n = 256$ time-points per trajectory, which are sampled on an equidistant grid over the training interval $[0, T]$. To increase the robustness of the model with respect to noisy, potentially incomplete observations, we train a second model, **NSODE-eps**, for which we add multiplicative Gaussian noise centered on 1 with a standard deviation of $\sigma = 0.01$ to the observed input trajectory. Furthermore we do not feed the full solution trajectory generated as described in Section 3.1 but keep only $n = 128$ time-points which are selected uniformly at random from the interval $[0, T]$. All details about model training such as hyperparameter choices and the used hardware are in Appendix A.3.

Table 2: Overview of baselines (f.d.: finite differences, ode: proposed for ODEs, MC: Monte Carlo, reg.: regression).

name	type	ode	f.d.	description	reference
AFP	GP	no	yes	age-fitness Pareto optimization	(Schmidt & Lipson, 2010)
FE-AFP	GP	no	yes	AFP with co-evolved fitness estimates	(Schmidt & Lipson, 2010)
EHC	GP	no	yes	AFP with epigenetic hillclimbing	(La Cava, 2016)
EPLEX	GP	no	yes	epsilon-lexicase selection	(La Cava et al., 2016)
GPGOMEA	GP	no	yes	gene-pool optimal mixing	(Virgolin et al., 2017)
FEAT	GP	no	yes	learned differentiable features	(La Cava et al., 2018)
PySR	GP	no	yes	AutoML-Zero + simulated annealing + const. optim.	(Cranmer, 2020)
SINDy	reg	yes	yes	sparse linear regression	(Brunton et al., 2016)
FFX	reg	no	yes	pathwise regularized ElasticNet regression	(McConaghy, 2011)
BSR	MC	no	yes	MCMC on linearly mixed tree-representations	(Jin et al., 2019)
ProGED	MC	yes	no	MC on probabilistic context free grammars+const. optim.	(Brence et al., 2021)

4. Experiments

4.1. Benchmark Datasets

We evaluate model performance and generalization on several test sets, which are summarized in Figure 5.

- **Classic**: To validate our approach on existing benchmarks, we turn to the functional symbolic regression literature and simply interpret functions as ODEs. In particular, we include all scalar functions listed in the overview in (McDermott et al., 2012), which includes equations from multiple established benchmarks (Keijzer, 2003; Koza, 1993; 1994; Uy et al., 2011; Vladislavleva et al., 2008). For example, we interpret the function $f(y) = y^3 + y^2 + y$ from Uy et al. (2011) as an autonomous ODE $\dot{y}(t) = f(y(t)) = y(t)^3 + y(t)^2 + y(t)$, which we solve numerically for randomly sampled initial values (as detailed in Section 3.1). This test set consists of 26 distinct equations.

- **Textbook**: To assess how NSODE performs on "real problems", we manually curated 12 non-linear ODEs from Wikipedia, physics textbooks, and lecture notes from university courses on ODEs. These equations are listed in Table 7 in Appendix B. We note that they are all relatively simple compared to the expressions in our generated training set, consisting mostly of low order polynomials, some of which with one fractional exponent.

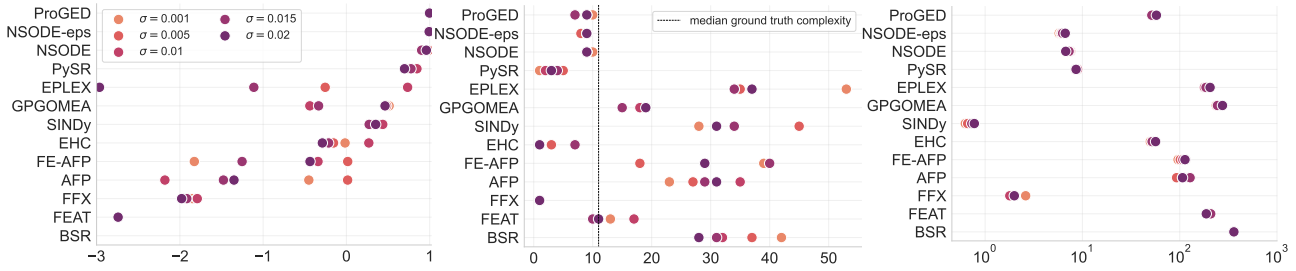
- **Large**: The **Classic** and **Textbook** datasets are relatively small and simple in terms of the complexity and operator diversity of the expressions (cf. Figure 5). Hence, we generate a larger and more diverse dataset by resampling equations from the training distribution described in Section 3.1 and solving them for new initial conditions to generate new, unseen trajectory. To further reduce bias towards our training distribution, we employ rejection sampling to ensure that no skeleton is included more than once and that we include at most 10 equations per complexity. The final dataset consists of 162 ODEs, which is comparable in size to the datasets used in the recent extensive functional SR benchmark study by La Cava et al. (2021). We refrained from including even more equations, because most SR methods require a separate optimization per expression, quickly rendering the evaluation of baselines computationally infeasible.

4.2. Baselines

We compare our method to 11 popular baselines choosing strong contenders from different categories described in Section 2. We provide a brief overview in Table 2 and defer details on hyperparameter choices to Appendix D. All baselines explicitly fit a separate regression function for each individual observed trajectory. Moreover, except for ProGED, which has a specific mode for ODE discovery, all models use functional SR and require finite difference approximations for the derivatives as inputs. We use smoothed finite differences as provided by the PySindy (de Silva et al., 2020) implementation `SmoothedFiniteDifference` with a smoothing window length of 15. Notebly, this implementation also provides methods to approximate temporal derivatives under unevenly spaced sampling intervals. Beyond the baselines in Table 2, we attempted to compare to AI Feynman (Udrescu et al., 2020; Weilbach et al., 2021), deep symbolic regression (Petersen et al., 2021), multiple regression genetic programming (Arnaldo et al., 2014), and semantic backpropagation-based genetic programming (Virgolin et al., 2019). However, due to their large inference time per equation we could not obtain sufficient results for a reasonable comparison to other models. The relatively long inference times of these methods are confirmed in La Cava et al. (2021, Fig. 1), where they all average at above 2.5 hours per expression.

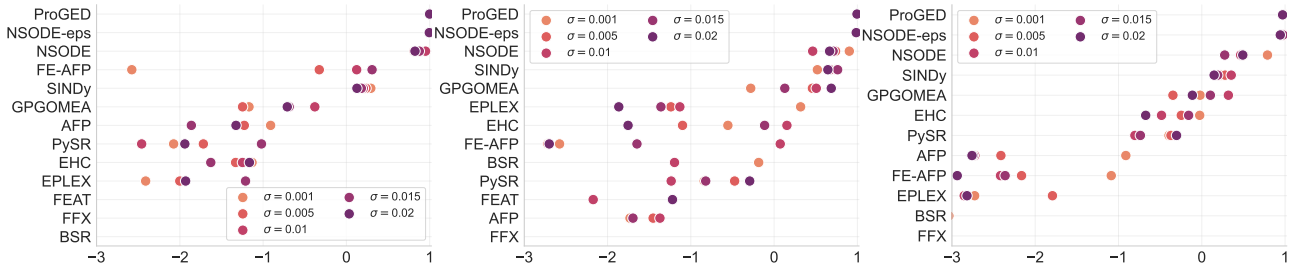
4.3. Metrics and Evaluation

Metrics. For performance evaluations we follow standard procedures within the symbolic regression literature (see, e.g., (La Cava et al., 2021)) and assess accuracy, expression parsimony and inference time per equation. To compute



(a) R^2 (b) complexity (c) inference time (sec.)

Figure 2: **Performance metrics on Classic.** Median R^2, complexity, and inference times on **Classic** with 192 irregularly spaced points and different noise levels σ in the interpolation regime $[0, T]$. Rows in all plots are ordered according to best R^2 scores. In (a) the x-axis is restricted to the relevant interval $[-3, 1]$; missing performances (e.g., for BSR) fall below this threshold.. The black dashed line in (b) denotes the median complexity across all samples in the testset.



(a) R^2 on **Classic** (b) R^2 on **Textbook** (c) R^2 on **Large**

Figure 3: **Performance across testsets.** Median R^2 scores across predictions per model using 128 irregularly spaced time points for different noise levels σ in the interpolation regime $[0, T]$. The x-axes are restricted to the relevant interval $[-3, 1]$; missing performances fall below this threshold.

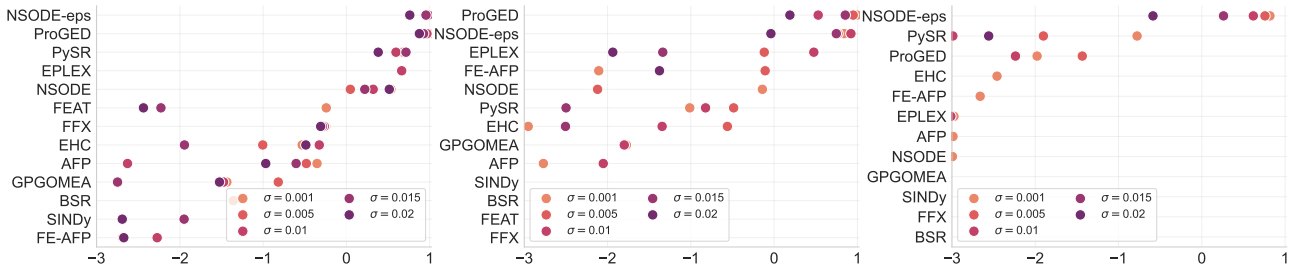
accuracy metrics we first integrate the predicted ODE expression over the interval $[0, T]$ using the same initial value as in the ground truth trajectory. The integrated predicted trajectory is then compared to the ground truth trajectory in terms of the well-established coefficient of determination (R^2), the L_1 and L_∞ norm of their difference as well as the average `numpy.isclose` as a possible alternative as suggested by (Biggio et al., 2021)[3]. Expression parsimony is measured in terms of equation complexity as introduced in Section 3.1. In short, the complexity is the total number of operators, variables, and constants in an expression.

Model selection. Many of the baseline methods as well as NSODE predict a list of candidate ODE equations. To select one final equation, we numerically integrate all predicted candidates over the training interval $[0, T]$ and select the candidate with the best R^2 score between the trajectory of the predicted ODE and the actually observed trajectory.

Noise and irregular sampling. To assess resilience to different signal-to-noise ratios we adopt the multiplicative noise model introduced in d'Ascoli et al. (2022). This noise model recognizes that zero-centered, fixed variance additive noise fails to take into account the magnitude of y_i and may affect the signal-to-noise ratio too much or too little depending on the scale of the observations. Instead we scale the standard deviation of additive, zero-centered Gaussian noise for y_i by $|y_i|$. This can equivalently be modeled as multiplicative noise from a Gaussian distribution centered on 1 where the choice of standard deviation σ determines the signal-to-noise ratio with $1/\sigma$. We evaluate performances for $\sigma \in \{0.001, 0.005, 0.01, 0.015, 0.02\}$. We remark that these noise levels go beyond the noise level used for training NSODE-eps. Additionally, we imitate irregularly spaced sampling intervals by uniformly randomly sub-sampling the original solution trajectory to $n \in \{128, 192, 256\}$ number of time points within the training interval $[0, T]$.

Extrapolation. One motivation for symbolic regression is to find an expression that not only describes the observed data well, but that can also be used for extrapolation. To assess the extrapolation capabilities of the predicted equations, we integrate the predicted ODEs on the adjacent interval $[T, T_{\text{extra}}]$ and evaluate the accuracy metrics between the

[3]For two arrays `a` and `b` we define the average isclose as `numpy.isclose(a, b).sum()/len(b)` with `atol=1e-10` and `rtol=0.05`; `a` corresponds to predictions, `b` corresponds to ground truth.



(a) R^2 on **Classic** (b) R^2 on **Textbook** (c) R^2 on **Large**

Figure 4: **Extrapolation performance.** Median R^2 scores across predictions on different datasets with 128 randomly spaced time points for different noise levels σ in the extrapolation regime $[T, T_{\text{extra}}]$. The x-axes are restricted to the relevant interval $[-3, 1]$; missing performances fall below this threshold.

dense, noise-free ground truth trajectory and the predicted trajectory over this extrapolation interval.

5. Results

For a meaningful comparison, R^2 scores are always reported on the noiseless and equidistantly sampled trajectories after solving the ground truth and predicted ODEs on $[0, T]$ for interpolation and on $[T, T_{\text{extra}}]$ for extrapolation, respectively.

Interpolation performance. Figure 2 summarizes the performance of all models for different noise levels in terms of median accuracy (R^2), complexity, and inference time on **Classic** with $n = 192$ time points. Methods are ordered by the best R^2 across all noise levels for all three plots. Our NSODE(-eps) and ProGED, the only other method not relying on finite difference approximations, are substantially more robust towards observation noise. NSODE-eps and ProGED achieve essentially equal R^2 values.

In terms of parsimony, several models predict expressions of lower complexity than NSODE(-eps) and ProGED. However, the median complexities of NSODE(-eps) and ProGED are in good agreement with the median complexity of the ground truth expressions (see Figure 5 for full distributions). Lower complexity is not necessarily better.

Finally, NSODE(-eps) is approximately one order of magnitude faster than ProGED, faring in the upper-mid range in terms of inference time overall. While inference time is implementation and hardware dependent, we emphasize the conceptual difference that with NSODE(-eps) a single model can be used for all predictions, whereas all other models need to be fit separately per equation.

These results qualitatively generalize across our datasets as well as to different numbers of training points: Figure 3 shows that NSODE(-eps) again take 2nd and 3rd on **Classic**, **Textbook**, and **Large** using $n = 128$ time points. Again, NSODE-eps and ProGED perform similarly. Additional results for all combinations of datasets and used time points n, as well as evaluations of the alternative accuracy metrics (L_1, L_∞ norm, average `numpy.isclose`) can be found in Appendix E and corroborate this overall trend. We thus answer our initial question in the affirmative: *Models tailored specifically to inferring dynamics are superior over (even highly optimized) re-purposed functional SR methods.* NSODE-eps additionally scales efficiently.

Extrapolation. Results for all testsets for $n = 128$ time points in the extrapolation regime are shown in Figure 4. Even in this challenging setting, NSODE-eps performs comparably to ProGED on the small **Classic** and **Textbook** testsets and is the only method producing reasonable results on **Large** for noise levels up to $\sigma = 0.01$, which has been used during training. Appendix E shows that these results also generalize to $n \in \{192, 256\}$. While NSODE-eps and ProGED again outperform other models, their accuracy degrades on extrapolation tasks as the noise level increases.

Robustness. Figures 3 and 4 show that NSODE-eps is considerably more robust to noisy observations than NSODE and works well even for noise levels two times higher than what has been used for training. It also generalizes well to different numbers of (irregularly spaced) observations, and even manages extrapolation beyond the observed time range in a range of settings.

6. Limitations

Although NSODE comes with conceptual advantages over classical symbolic regression approaches for the task of ODE prediction, notably in that it does not require estimates of temporal derivatives, the presented approach also comes with a number of limitations. Perhaps most severely, we restrict ourself to the arguably most simple class of differential equations: explicit autonomous scalar first-order ODEs. These equations serve as a good initial benchmark but are limiting for applications in scientific discovery in practice. This restriction represents a design choice for the

scope of this paper and does not necessarily imply a fundamental limitation of the approach: On the one hand, the model architecture can readily be extended to systems of equations, on the other hand we want to emphasize that systems of equations showcase a much richer set of qualitative behaviors, including oscillation and chaos, making model extensions towards them potentially non-trivial. As such it currently remains an open question and important challenge for future work to explore the scalability of the presented model paradigm for ODE prediction.

A second limitation of the presented model is that in its current form it can not profit from multiple observations of the same process. In other words, even if we have multiple observed trajectories of the same process available, the model can only be applied to each trajectory individually. Allowing the model to profit from multiple observations appears to be a promising step to further increase its robustness to noise and irregularly sampled data.

7. Conclusion

We have developed NSODE, an efficiently scalable method to infer ordinary differential equations $\dot{y} = f(y)$ from a single observed solution trajectory. NSODE follows the successful paradigm of large-scale pretraining of attention-based sequence-to-sequence models on essentially unlimited amounts of simulated data, where the inputs are the observed solution $\{(t_i, y_i)\}_{i=1}^n$ and the output is a symbolic expression for f. Once trained, our method performs on par or better than existing baselines and is an order of magnitude faster than similarly accurate symbolic regression techniques, which require a separate optimization for each expression. NSODE is robust to different noise levels, the number of irregularly spaced samples and recovers dynamics that extrapolate beyond the observed time range. While we have demonstrated the advantages of tailoring symbolic regression techniques specifically to recovering dynamics, interesting directions for future work include incorporating domain knowledge, and extending the framework to partial differential equations or high-dimensional systems of coupled differential equations. Despite the huge potential of automated dynamical law learning for scientific discovery and hypothesis generation in the sciences, we caution against blindly trusting model outputs to represent generalizable real-world natural laws without rigorous experimental validation.

8. Acknowledgements

SB is supported by the Helmholtz Association under the joint research school "Munich School for Data Science - MUDS". This work was supported by the Helmholtz Association's Initiative and Networking Fund on the HAICORE@FZJ partition.

References

Arnaldo, I., Krawiec, K., and O'Reilly, U.-M. Multiple regression genetic programming. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, pp. 879–886, 2014.

Atkinson, S., Subber, W., Wang, L., Khan, G., Hawi, P., and Ghanem, R. Data-driven discovery of free-form governing differential equations. *arXiv preprint arXiv:1910.05117*, 2019.

Bakarji, J., Champion, K., Kutz, J. N., and Brunton, S. L. Discovering governing equations from partial measurements with deep delay autoencoders. *arXiv preprint arXiv:2201.05136*, 2022.

Biggio, L., Bendinelli, T., Neitz, A., Lucchi, A., and Parascandolo, G. Neural symbolic regression that scales. In Meila, M. and Zhang, T. (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 936–945. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/biggio21a.html.

Brence, J., Todorovski, L., and Džeroski, S. Probabilistic grammars for equation discovery. *Knowledge-Based Systems*, 224:107077, 2021.

Brenner, M., Hess, F., Mikhaeil, J. M., Bereska, L. F., Monfared, Z., Kuo, P.-C., and Durstewitz, D. Tractable dendritic rnns for reconstructing nonlinear dynamical systems. In *International Conference on Machine Learning*, pp. 2292–2320. PMLR, 2022.

Bridewell, W., Langley, P., Todorovski, L., and Džeroski, S. Inductive process modeling. *Machine learning*, 71(1): 1–32, 2008.

Brunton, S. L., Proctor, J. L., and Kutz, J. N. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 113(15): 3932–3937, 2016. ISSN 0027-8424. doi: 10.1073/pnas.1517384113. URL https://www.pnas.org/content/113/15/3932.

Burlacu, B., Kronberger, G., and Kommenda, M. Operon c++ an efficient genetic programming framework for symbolic regression. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, pp. 1562–1570, 2020.

Champion, K., Lusch, B., Kutz, J. N., and Brunton, S. L. Data-driven discovery of coordinates and governing equations. *Proceedings of the National Academy of Sciences*, 116(45):22445–22451, 2019.

Chen, R. T., Rubanova, Y., Bettencourt, J., and Duvenaud, D. Neural ordinary differential equations. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pp. 6572–6583, 2018.

Costa, A., Dangovski, R., Dugan, O., Kim, S., Goyal, P., Soljačić, M., and Jacobson, J. Fast neural models for symbolic regression at scale, 2021.

Cranmer, M. Pysr: Fast & parallelized symbolic regression in python/julia, September 2020. URL http://doi.org/10.5281/zenodo.4041459.

d'Ascoli, S., Kamienny, P.-A., Lample, G., and Charton, F. Deep symbolic regression for recurrent sequences. *arXiv preprint arXiv:2201.04600*, 2022.

D'Ascoli, S., Kamienny, P.-A., Lample, G., and Charton, F. Deep symbolic regression for recurrence prediction. In Chaudhuri, K., Jegelka, S., Song, L., Szepesvari, C., Niu, G., and Sabato, S. (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 4520–4536. PMLR, 17–23 Jul 2022.

de Franca, F. O. and Aldeia, G. S. I. Interaction–transformation evolutionary algorithm for symbolic regression. *Evolutionary computation*, 29(3):367–390, 2021.

de Silva, B., Champion, K., Quade, M., Loiseau, J.-C., Kutz, J. N., and Brunton, S. Pysindy: A python package for the sparse identification of nonlinear dynamical systems from data. *Journal of Open Source Software*, 5(49):1–4, 2020.

Gec, B., Omejc, N., Brence, J., Džeroski, S., and Todorovski, L. Discovery of differential equations using probabilistic grammars. In *Discovery Science: 25th International Conference, DS 2022, Montpellier, France, October 10–12, 2022, Proceedings*, pp. 22–31. Springer, 2022.

Gilpin, W. Chaos as an interpretable benchmark for forecasting and data-driven modelling. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021. URL https://openreview.net/forum?id=enYjtbjYJrf.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. Array programming with NumPy. *Nature*, 2020.

Hindmarsh, A. and Laboratory, L. L. *ODEPACK, a Systematized Collection of ODE Solvers*. Lawrence Livermore National Laboratory, 1982. URL https://books.google.de/books?id=9XWPmwEACAAJ.

Hunter, J. D. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, 2007.

Iakovlev, V., Heinonen, M., and Lähdesmäki, H. Learning continuous-time {pde}s from sparse data with graph neural networks. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=aUX5Plaq7Oy.

Jin, Y., Fu, W., Kang, J., Guo, J., and Guo, J. Bayesian symbolic regression. *arXiv preprint arXiv:1910.08892*, 2019.

Kamienny, P.-A., d'Ascoli, S., Lample, G., and Charton, F. End-to-end symbolic regression with transformers. *arXiv preprint arXiv:2204.10532*, 2022.

Keijzer, M. Improving symbolic regression with interval arithmetic and linear scaling. In *European Conference on Genetic Programming*, pp. 70–82. Springer, 2003.

Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J., Grout, J., Corlay, S., Ivanov, P., Avila, D., Abdalla, S., Willing, C., and Jupyter development team. Jupyter notebooks - a publishing format for reproducible computational workflows. In *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, 2016.

Kovachki, N., Li, Z., Liu, B., Azizzadenesheli, K., Bhattacharya, K., Stuart, A., and Anandkumar, A. Neural operator: Learning maps between function spaces. *arXiv preprint arXiv:2108.08481*, 2021. doi: 10.48550/ARXIV.2106.06898.

Koza, J. R. *Genetic programming - on the programming of computers by means of natural selection*. Complex adaptive systems. MIT Press, 1993. ISBN 978-0-262-11170-6.

Koza, J. R. *Genetic programming II: automatic discovery of reusable programs*. MIT press, 1994.

La Cava, W., Spector, L., and Danai, K. Epsilon-lexicase selection for regression. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, pp. 741–748, 2016.

La Cava, W., Singh, T. R., Taggart, J., Suri, S., and Moore, J. H. Learning concise representations for regression by evolving networks of trees. In *International Conference on Learning Representations*, 2018.

La Cava, W., Orzechowski, P., Burlacu, B., de França, F. O., Virgolin, M., Jin, Y., Kommenda, M., and Moore, J. H. Contemporary symbolic regression methods and their relative performance. *arXiv preprint arXiv:2107.14351*, 2021.

La Cava, W. G. *Automatic Development and Adaptation of Concise Nonlinear Models for System Identification*. PhD thesis, University of Massachusetts Amherst, 2016.

Lample, G. and Charton, F. Deep learning for symbolic mathematics. In *International Conference on Learning Representations*, 2019.

Landajuela, M., Petersen, B. K., Kim, S., Santiago, C. P., Glatt, R., Mundhenk, N., Pettit, J. F., and Faissol, D. Discovering symbolic policies with deep reinforcement learning. In *International Conference on Machine Learning*, pp. 5979–5989. PMLR, 2021.

Lejarza, F. and Baldea, M. Data-driven discovery of the governing equations of dynamical systems via moving horizon optimization. *Scientific Reports*, 12(1):1–15, 2022.

Li, Z., Liu-Schiaffini, M., Kovachki, N., Liu, B., Azizzadenesheli, K., Bhattacharya, K., Stuart, A., and Anandkumar, A. Learning dissipative dynamics in chaotic systems, 2021.

Liu, J., Long, Z., Wang, R., Sun, J., and Dong, B. Rode-net: learning ordinary differential equations with randomness from data. *arXiv preprint arXiv:2006.02377*, 2020.

Long, Z., Lu, Y., and Dong, B. Pde-net 2.0: Learning pdes from data with a numeric-symbolic hybrid deep network. *Journal of Computational Physics*, 399:108925, 2019.

Lusch, B., Kutz, J. N., and Brunton, S. L. Deep learning for universal linear embeddings of nonlinear dynamics. *Nature communications*, 9(1):1–10, 2018.

Makke, N. and Chawla, S. Interpretable scientific discovery with symbolic regression: A review. *arXiv preprint arXiv:2211.10873*, 2022.

McConaghy, T. Ffx: Fast, scalable, deterministic symbolic regression technology. In *Genetic Programming Theory and Practice IX*, pp. 235–260. Springer, 2011.

McDermott, J., White, D. R., Luke, S., Manzoni, L., Castelli, M., Vanneschi, L., Jaskowski, W., Krawiec, K., Harper, R., De Jong, K., et al. Genetic programming needs better benchmarks. In *Proceedings of the 14th annual conference on Genetic and evolutionary computation*, pp. 791–798, 2012.

Meurer, A., Smith, C. P., Paprocki, M., Čertík, O., Kirpichev, S. B., Rocklin, M., Kumar, A., Ivanov, S., Moore, J. K., Singh, S., Rathnayake, T., Vig, S., Granger, B. E., Muller, R. P., Bonazzi, F., Gupta, H., Vats, S., Johansson, F., Pedregosa, F., Curry, M. J., Terrel, A. R., Roučka, v., Saboo, A., Fernando, I., Kulal, S., Cimrman, R., and Scopatz, A. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3, January 2017. ISSN 2376-5992. doi: 10.7717/peerj-cs.103. URL https://doi.org/10.7717/peerj-cs.103.

Mundhenk, T. N., Landajuela, M., Glatt, R., Santiago, C. P., Faissol, D. M., and Petersen, B. K. Symbolic regression via neural-guided genetic programming population seeding. *arXiv preprint arXiv:2111.00053*, 2021.

pandas development team, T. pandas-dev/pandas: Pandas, February 2020. URL https://doi.org/10.5281/zenodo.3509134.

Park, Y., Gajamannage, K., Jayathilake, D. I., and Bollt, E. M. Recurrent neural networks for dynamical systems: Applications to ordinary differential equations, collective motion, and hydrological modeling. *arXiv preprint arXiv:2202.07022*, 2022.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. PyTorch: An imperative style, high-performance deep learning library. In *NeurIPS*, 2019.

Patel, D. and Ott, E. Using machine learning to anticipate tipping points and extrapolate to post-tipping dynamics of non-stationary dynamical systems. *arXiv preprint arXiv:2207.00521*, 2022.

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., and Duchesnay, É. Scikit-learn: Machine learning in Python. *JMLR*, 2011.

Petersen, B. K., Landajuela, M., Mundhenk, T. N., Santiago, C. P., Kim, S. K., and Kim, J. T. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. In *Proc. of the International Conference on Learning Representations*, 2021.

Qian, Z., Kacprzyk, K., and van der Schaar, M. D-code: Discovering closed-form odes from observed trajectories. In *International Conference on Learning Representations*, 2022.

Rudy, S. H., Brunton, S. L., Proctor, J. L., and Kutz, J. N. Data-driven discovery of partial differential equations. *Science advances*, 3(4):e1602614, 2017.

Sahoo, S., Lampert, C., and Martius, G. Learning equations for extrapolation and control. In *International Conference on Machine Learning*, pp. 4442–4450. PMLR, 2018.

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

Schmidt, M. D. and Lipson, H. Age-fitness pareto optimization. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pp. 543–544, 2010.

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839): 604–609, 2020.

Simidjievski, N., Todorovski, L., Kocijan, J., and Džeroski, S. Equation discovery for nonlinear system identification. *IEEE Access*, 8:29930–29943, 2020.

Todorovski, L. and Dzeroski, S. Declarative bias in equation discovery. In *Proceedings of the Fourteenth International Conference on Machine Learning*, pp. 376–384, 1997.

Tohme, T., Liu, D., and Youcef-Toumi, K. Gsr: A generalized symbolic regression approach. *arXiv preprint arXiv:2205.15569*, 2022.

Udrescu, S.-M., Tan, A., Feng, J., Neto, O., Wu, T., and Tegmark, M. Ai feynman 2.0: Pareto-optimal symbolic regression exploiting graph modularity. *arXiv preprint arXiv:2006.10782*, 2020.

Uy, N. Q., Hoai, N. X., O'Neill, M., McKay, R. I., and Galván-López, E. Semantically-based crossover in genetic programming: application to real-valued symbolic regression. *Genetic Programming and Evolvable Machines*, 12(2):91–119, 2011.

Valipour, M., Panju, M., You, B., and Ghodsi, A. SymbolicGPT: A Generative Transformer Model for Symbolic Regression. In *Preprint Arxiv*, 2021. URL https://arxiv.org/abs/2106.14131.

van Rossum, G. and Drake, F. L. *Python 3 Reference Manual*. CreateSpace, 2009.

Vastl, M., Kulhánek, J., Kubalík, J., Derner, E., and Babuška, R. Symformer: End-to-end symbolic regression using transformer-based architecture. *arXiv preprint arXiv:2205.15764*, 2022.

Virgolin, M., Alderliesten, T., Witteveen, C., and Bosman, P. A. Scalable genetic programming by gene-pool optimal mixing and input-space entropy-based building-block learning. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 1041–1048, 2017.

Virgolin, M., Alderliesten, T., and Bosman, P. A. Linear scaling with and within semantic backpropagation-based genetic programming for symbolic regression. In *Proceedings of the genetic and evolutionary computation conference*, pp. 1084–1092, 2019.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, İ., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

Vlachas, P. R., Arampatzis, G., Uhler, C., and Koumoutsakos, P. Multiscale simulations of complex systems by learning their effective dynamics. *Nature Machine Intelligence*, 4(4):359–366, 2022.

Vladislavleva, E. J., Smits, G. F., and Den Hertog, D. Order of nonlinearity as a complexity measure for models generated by symbolic regression via pareto genetic programming. *IEEE Transactions on Evolutionary Computation*, 13(2):333–349, 2008.

Waskom, M. L. seaborn: statistical data visualization. *Journal of Open Source Software*, 2021.

Weilbach, J., Gerwinn, S., Weilbach, C., and Kandemir, M. Inferring the structure of ordinary differential equations. *arXiv preprint arXiv:2107.07345*, 2021.

Wes McKinney. Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman (eds.), *Proceedings of the 9th Python in Science Conference*, pp. 56 – 61, 2010. doi: 10.25080/Majora-92bf1922-00a.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T. L., Gugger, S., Drame, M., Lhoest, Q., and Rush, A. M. Transformers: State-of-the-art natural language processing. In *EMNLP*, 2020.

Zaheer, M., Guruganesh, G., Dubey, K. A., Ainslie, J., Alberti, C., Ontanon, S., Pham, P., Ravula, A., Wang, Q.,

Yang, L., et al. Big bird: Transformers for longer sequences. *Advances in Neural Information Processing Systems*, 33:17283–17297, 2020.

A. Implementation Details

A.1. Rules to Resample Constants

As described in Section 3.1, we generate ODEs as unary-binary trees, convert them to infix notation and parse them into a canonical form using `sympy`[4]. For each skeleton we then create up to 25 ODEs by sampling different values for the constants. When resampling constants we want to ensure that we do not accidentally modify the skeleton as this would additionally burden our model with resolving potential ambiguities in the grammar of ODE expressions. Furthermore, we do not want to reintroduce duplicate samples on the skeleton level after carefully filtering them out previously. We therefore introduce the following sampling rules for constants:

1. Do not sample constants of value 0.
2. When the original constant in the skeleton is negative, sample a negative constant, otherwise sample a positive constant.
3. Avoid base of 1 in power operations as $1^x = 1$.
4. Avoid exponent of 1 and -1 in power operations as $x^1 = x$ and $x^{-1} = 1/x$.
5. Avoid coefficients of value 1 and -1 as $1 \cdot x = x$ and $-1 \cdot x = -x$
6. Avoid divisions by 1 and -1 as $x/1 = x$ and $x/-1 = -x$

A.2. Data generation

As discussed in the main text, the choices of the maximum number of internal nodes per tree K, the choice and distribution over N_{bin} binary operators, the choice and distribution over N_{una} unary operators, the probability with which to decorate a leaf with a symbol p_{sym} (versus a constant with $1 - p_{\text{sym}}$), and the distribution p_{c} over constants uniquely determine the training distribution over ODEs f. These choices can be viewed as flexible and semantically interpretable tuning knobs to choose a prior over ODEs. For example, it may be known in a given context, that the system follows a "simple" law (small K) and does not contain exponential rates of change (do not include $\exp$ in the unary operators), and so on. The choice of the maximum number of operators per tree, how to sample the operators, and how to fill in the leaf nodes define the training distribution, providing us with flexible and semantically meaningful tuning knobs to choose a prior over ODE systems for our model. We summarize our choices in Tables 3 to 5, where $\mathcal{U}$ denotes the uniform distribution. Whenever a leaf node is decorated with a constant, the distribution over constants is determined by first choosing with equal probability whether to use an integer or a real value. In case of an integer, we sample it from p_{int}, and in case of a real-valued constant we sample it from p_{real} shown in Table 3. Finally, when it comes to the numerical solutions of the sampled ODEs, we fixed the parameters in Table 6 for our experiments.

We highlight that there is no such thing as "a natural distribution over equations" when it comes to ODEs. Hence, ad-hoc choices have to be made in one way or another. However, it is important to note that neither our chosen range of integers nor the range of real values for constants are in any way restrictive as they can be achieved by appropriate rescaling. In particular, the model itself represents these constant values merely be non-numeric tokens and interpolates between those anchor tokens (our two-hot encoding) to represent continuous values. Hence, the model is entirely agnostic to the actual numerical range spanned by these fixed grid tokens, but the relative accuracy in recovering interpolated values will be constant and thus scale with the absolute chosen range. Therefore, scaling p_{int} and p_{real} by essentially any power of 10 does not affect our findings. Similarly, the chosen range of initial values $(y_0^{\min}, y_0^{\max})$ is non-restrictive as one could simply scale each observed trajectory to have its starting value lie within this range.

[4]While `sympy` greatly helps with parsing functions into a canonical form, we remark that this is a pragmatic, best effort approach.

Table 3: Parameter settings for the data generation.

parameter	K	N_{bin}	N_{una}	p_{sym}	p_{int}	p_{real}
value	5	5	5	0.5	$\mathcal{U}(\{-10, \dots, 10\} \setminus \{0\})$	$\mathcal{U}((-10, 10))$

Table 4: Binary operators with their relative sampling frequencies

operator	$+$	$-$	$\cdot$	$\div$	pow
probability	0.2	0.2	0.2	0.2	0.2

Table 5: Unary operators with their relative sampling frequencies.

operator	sin	cos	exp	$\sqrt{}$	log
probability	0.2	0.2	0.2	0.2	0.2

Table 6: Parameters for numerical solutions of sampled ODEs.

parameter	N_{const}	N_{iv}	T	T_{extra}	N_{grid}	$(y_0^{\min}, y_0^{\max})$
value	25	25	2	4	1024	$(-5, 5)$

A.3. Model

For our Transformer model we choose the implementation of BigBird (Zaheer et al., 2020) available in HuggingFace. The model is trained on an internal academic compute cluster using 4 Nvidia A100 GPUs for 25 epochs after which we evaluate the best model based on the validation loss. We choose a batchsize of 600 and use a linear learning rate warm-up over 10,000 optimization step after which we keep the learning rate constant at 10^{-4}. For the fixed tokens that are used to decode constants, we choose an equidistant grid $-10 = x_1 < x_2 < \ldots < x_m = 10$ with $m = 21$. This worked well empirically and using fewer or more tokens did not seem to improve model performance substantially. We note that architecture and hyperparameter choices correspond to ad-hoc decision and were not systematically optimized. We use the same choices in all experiments.

While not relevant for our dataset as we check for convergence of the ODE solvers, we remark that the input-encoding via IEEE-754 binary representations also graciously represents special values such as `nan` or `inf` without causing errors. Those are thus valid inputs that may still provide useful training signal, e.g., "the solution of the ODE of interest goes to `inf` quickly".

B. Textbook equations dataset

Table 7 list the equations we collected from wikipedia, textbooks and lecture notes together with the initial values that we solved them for. We can also see that almost all of these equations simplify to low-order polynomials.

Table 7: Equations of the **Textbook** testset.

Name	Equation $f(x)$	simplified	y_0
autonomous Riccati	$0.6 \cdot y^2 + 2 \cdot y + 0.1$	$0.6 \cdot y^2 + 2 \cdot y + 0.1$	-0.2
autonomous Stuart-Landau	$-2.2/2 \cdot y^3 + 1.31 \cdot y$	$-1.1 \cdot y^3 + 1.31 \cdot y$	0.1
autonomous Bernoulli	$-1.3 \cdot y + 2.1 \cdot y^{2.2}$	$-1.3 \cdot y + 2.1 \cdot y^{2.2}$	0.6
compound interest	$0.1 \cdot y$	$0.1 \cdot y$	4.9
Newton's law of cooling	$-0.1 \cdot (y - 3)$	$0.3 - 0.1 \cdot y$	4.9
Logistic equation	$0.23 \cdot y \cdot (1 - y)$	$0.23 \cdot (y - y^2)$	4.9
Logistic equation with harvesting	$0.23 \cdot y \cdot (1 - 0.33 \cdot y) - 0.5$	$0.23 \cdot y - 0.76 \cdot y^2 - 0.5$	3.5
Logistic equation with harvesting 2	$2 \cdot y \cdot (1 - y/3) - 0.5$	$2 \cdot y - 0.66 \cdot y^2 - 0.5$	0.7
Solow-Swan	$y^0.5 \cdot (0.9 \cdot 8 - (3 + 2.5) \cdot y^{1-0.5})$	$7.2 \cdot y^{0.5} - 5.5 \cdot y$	0.1
Tank draining	$-\sqrt{2 \cdot 9.81} \cdot (2/9)^2 \cdot \sqrt{y}$	$-0.21 \cdot y^{0.5}$	1
Draining water through a funnel	$-(0.5^2/4) \cdot \sqrt{2 \cdot 9.81} \cdot (\sin 1/\cos 1)^2 \cdot y^{-1.5}$	$-0.67/y^{1.5}$	3
velocity of a body thrown vertically upwards	$-9.81 - 0.9 \cdot y/8.2$	$-0.10 \cdot y - 9.81$	0.1

C. Dataset statistics

We provide an overview over the complexity distribution and the absolute frequency of all operators (after simplification) for all datasets in Figure 5. We can see that our self-generated dataset covers by far the larges complexity whereas both complexities and operator diversity are much lower for equations in the **Classic** and **Textbook** ODEs.

D. Baselines

We here describe more detail on the optimization of the baseline comparison models. Most models have a number of hyperparameters which need to be optimized per equation. Unless specified below, we use the default hyperparameters values and hyperparameter grid search settings specified in the implementation alongside the benchmark study by (La Cava et al., 2021). Whenever supported by a model's implementation we use `GridSearchCV` from `scikit-learn` (Pedregosa et al., 2011) for hyperparameter optimization. For this optimization we split the observed trajectory into a training interval $[0, 1]$ and a validation interval $[1, 2]$. In order to obtain results in reasonable time, we set a runtime limit of 3 minutes per hyperparameter optimization run.

AFP, EHC, EPLEX & FE_AFP.
op_list=['n','v','+','-','*','/','exp','log','2','3', 'sqrt','sin','cos']

FEAT.
functions= '+, -, *, /, ^2, ^3, sqrt, sin, cos, exp, log, ^'

PySR.
niterations=40
binary_operators=['plus', 'sub', 'mult', 'pow', 'div']
unary_operators=['cos', 'exp', 'sin', 'neg', 'log', 'sqrt']

ProGED.
sample_size=64
task_type = 'differential'

SINDy. We use the implementation available in PySINDy (de Silva et al., 2020) and instantiate the basis functions with polynomials up to degree 10 as well as all unary operators listed in Table 5. When fitting SINDy to data we often encountered numerical issues especially when using high-degree polynomial or the exponential function. To attenuate such issues discard

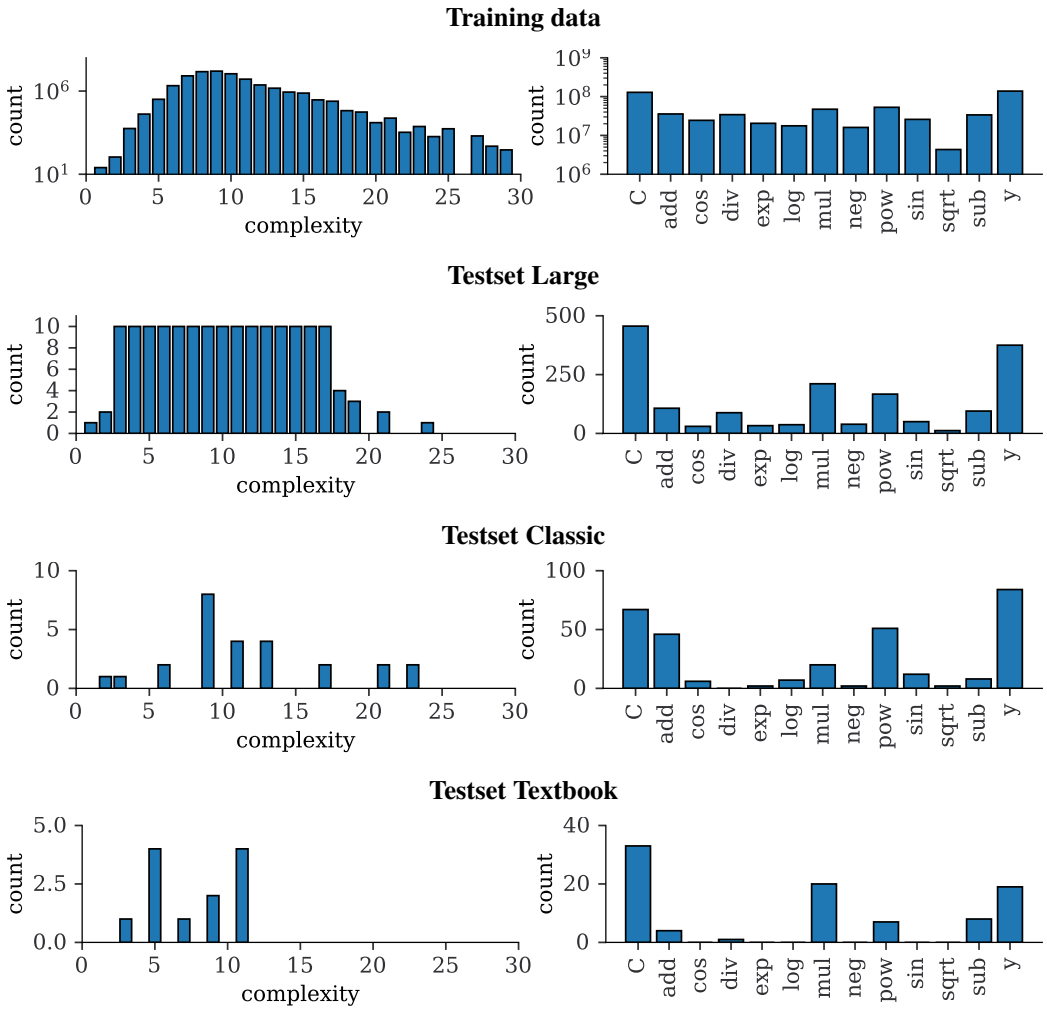


Figure 5: Distribution of complexity and operators for all datasets.

the particular basis function that raised a numerical error and restart the fitting process. We remark that removing basis functions and restarting the optimization is practically feasible for SINDy due to its extremely fast runtime. At the same time, being a regression based model, SINDy can (in contrast to genetic programming based models) not easily recover from a numerical issue caused by a particular basis function. We run a separate full grid search per (sample x noise level x number of time points) over the following hyperparameters and respective values (these all include the default values):

- optimizer-threshold (`np.logspace(-5, 0, 10)`): Minimum magnitude for a coefficient in the weight vector to not be zeroed out.
- optimizer-alpha ($[0.001, 0.0025, 0.005, 0.01, 0.025, 0.05, 0.1, 0.2]$): L2 regularizer on parameters.
- finite differences order ($[2, 3, 5, 7, 9]$): Order of finite difference approximation.
- maximum number of optimization iterations ([20, 100]): Maximum number of optimization steps.

Model selection. Most models provide a list of candidate solutions for each sample, e.g. the pareto-front (accuracy vs complexity tradeoff) in genetic processing based methods. To obtain a single predicted equation per model we use the model selection procedure outlined in 3.2.

E. Detailed results

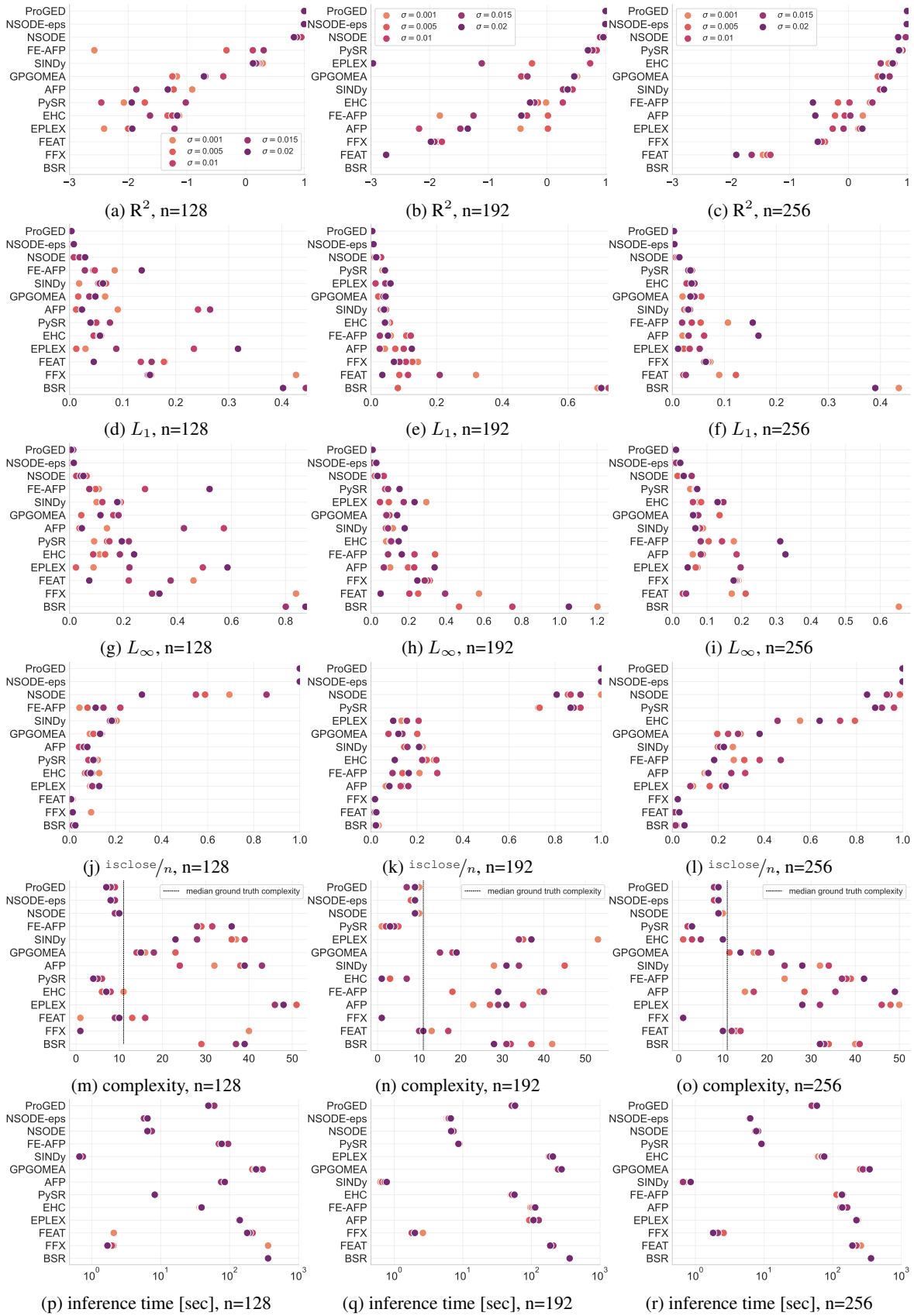
Here we provide detailed results on all experimental conditions across all datasets. We start with results on the interpolation interval $[0, T]$ in Appendix E.1 before showing results on the extrapolation interval $[T, T_{\text{extra}}]$ in Appendix E.1. To facilitate navigation we provide an overview in Table 8.

Table 8: Result overview.

Dataset	interval	fig. number
Classic	interpolation	Figure 6
Textbook	interpolation	Figure 7
Large	interpolation	Figure 8
Classic	extrapolation	Figure 9
Textbook	extrapolation	Figure 10
Large	extrapolation	Figure 11

E.1. Interpolation results

Across all datasets, the direct comparison of results with $n = 128$ time-points vs $n = 192$ time-points vs $n = 256$ time-points reveals a gradual performance degradation of models relying on finite difference approximation with increasing sparsity of the observed data both in terms of R^2 (top rows in Figures 6 to 8). NSODE-eps and ProGED on the other hand perform consistently well across these settings.



(a) R^2, n=128 (b) R^2, n=192 (c) R^2, n=256

(d) L_1, n=128 (e) L_1, n=192 (f) L_1, n=256

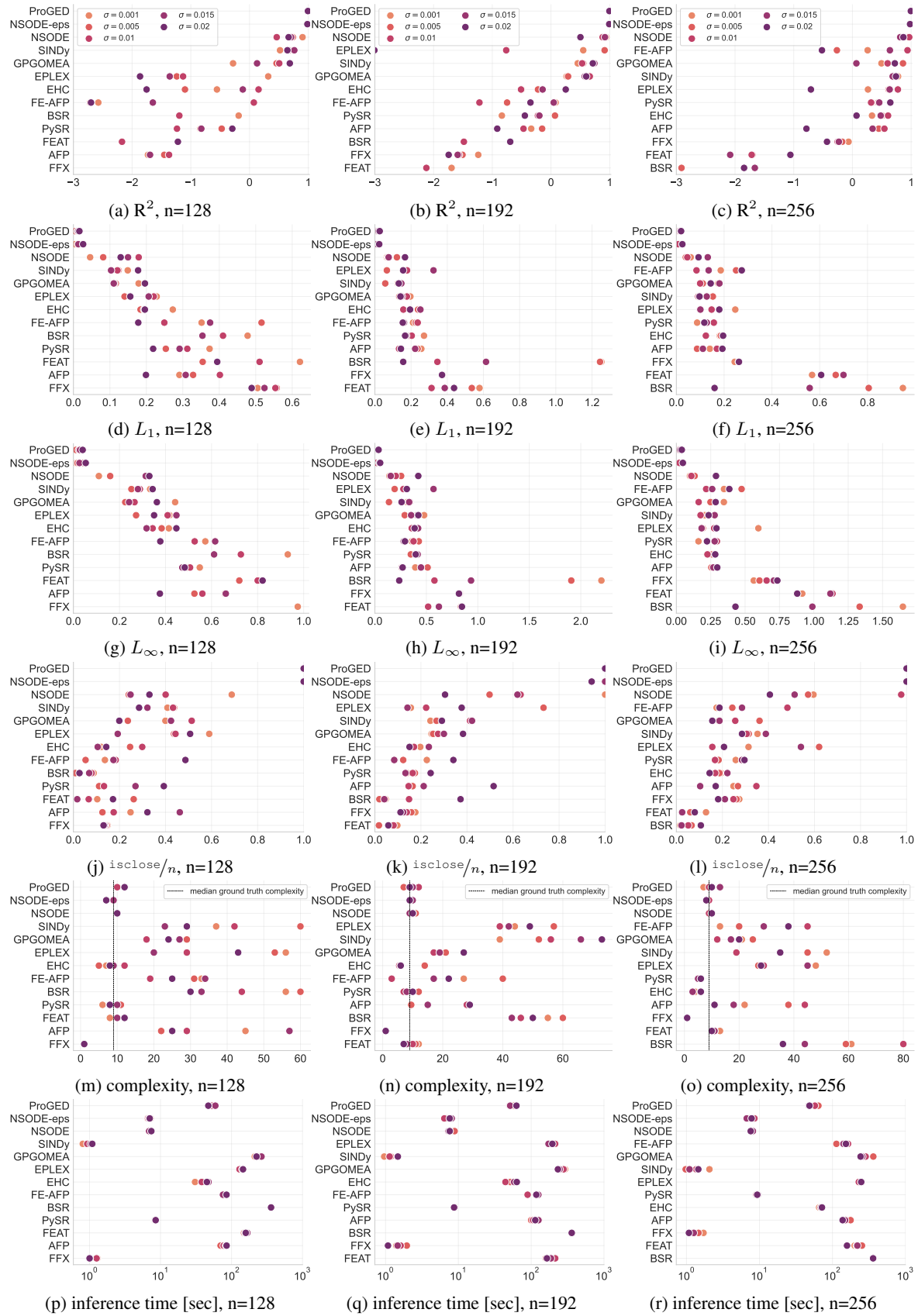
(g) L_∞, n=128 (h) L_∞, n=192 (i) L_∞, n=256

(j) $\texttt{isclose}/n$, n=128 (k) $\texttt{isclose}/n$, n=192 (l) $\texttt{isclose}/n$, n=256

(m) complexity, n=128 (n) complexity, n=192 (o) complexity, n=256

(p) inference time [sec], n=128 (q) inference time [sec], n=192 (r) inference time [sec], n=256

Figure 6: Interpolation. Median scores on **Classic** for n irregularly sampled time points across different noise levels σ.



(a) R^2, n=128 (b) R^2, n=192 (c) R^2, n=256

(d) L_1, n=128 (e) L_1, n=192 (f) L_1, n=256

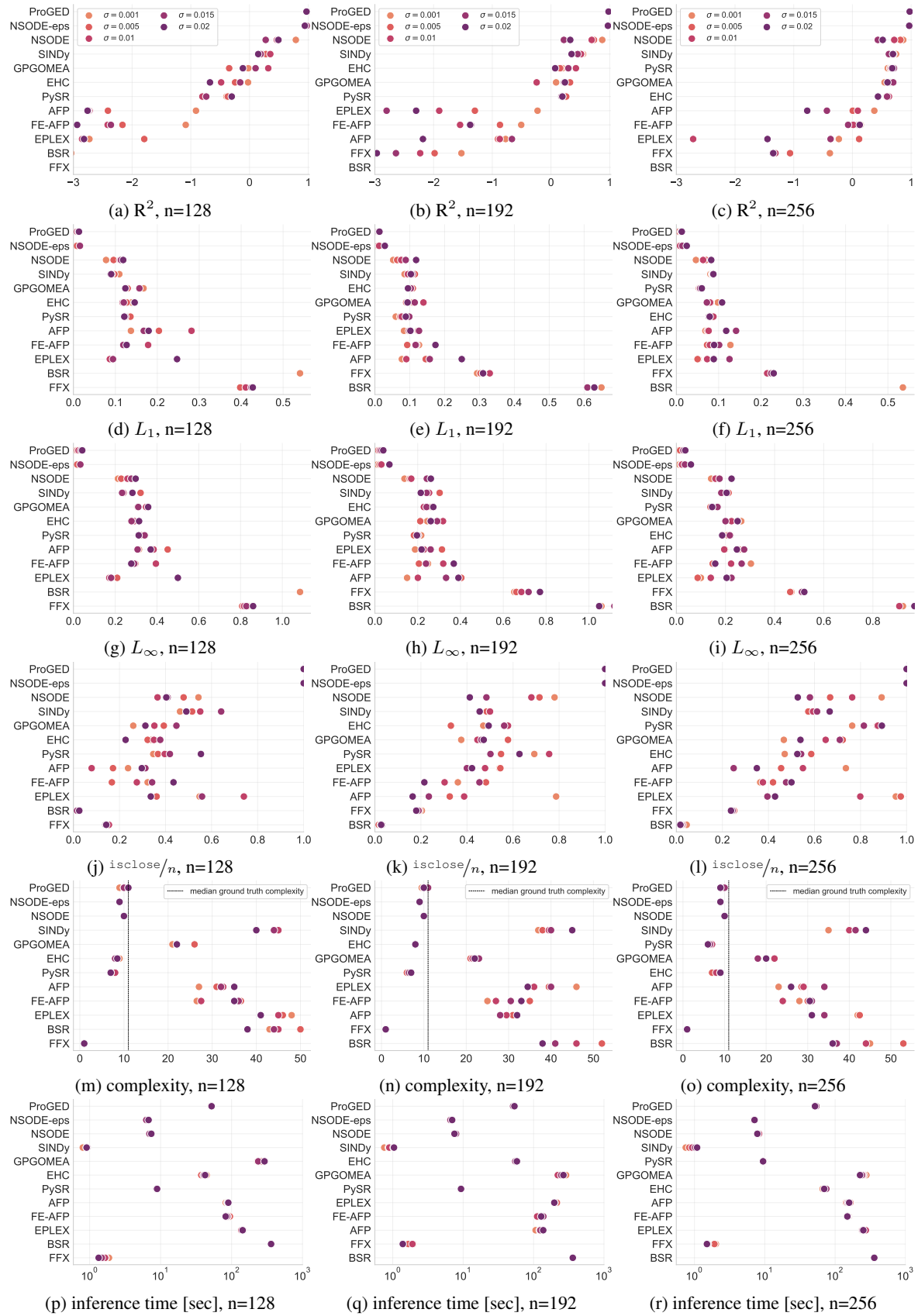
(g) L_∞, n=128 (h) L_∞, n=192 (i) L_∞, n=256

(j) $\texttt{isclose}/n$, n=128 (k) $\texttt{isclose}/n$, n=192 (l) $\texttt{isclose}/n$, n=256

(m) complexity, n=128 (n) complexity, n=192 (o) complexity, n=256

(p) inference time [sec], n=128 (q) inference time [sec], n=192 (r) inference time [sec], n=256

Figure 7: Interpolation. Median scores on **Textbook** for n irregularly sampled time points across different noise levels σ.



(a) R^2, n=128 (b) R^2, n=192 (c) R^2, n=256

(d) L_1, n=128 (e) L_1, n=192 (f) L_1, n=256

(g) L_∞, n=128 (h) L_∞, n=192 (i) L_∞, n=256

(j) $\texttt{isclose}/n$, n=128 (k) $\texttt{isclose}/n$, n=192 (l) $\texttt{isclose}/n$, n=256

(m) complexity, n=128 (n) complexity, n=192 (o) complexity, n=256

(p) inference time [sec], n=128 (q) inference time [sec], n=192 (r) inference time [sec], n=256

Figure 8: Interpolation. Median scores on **Large** for n irregularly sampled time points across different noise levels σ.

E.2. Extrapolation results

Performance evaluation on the extrapolation interval $[T, T_{\text{extra}}]$.

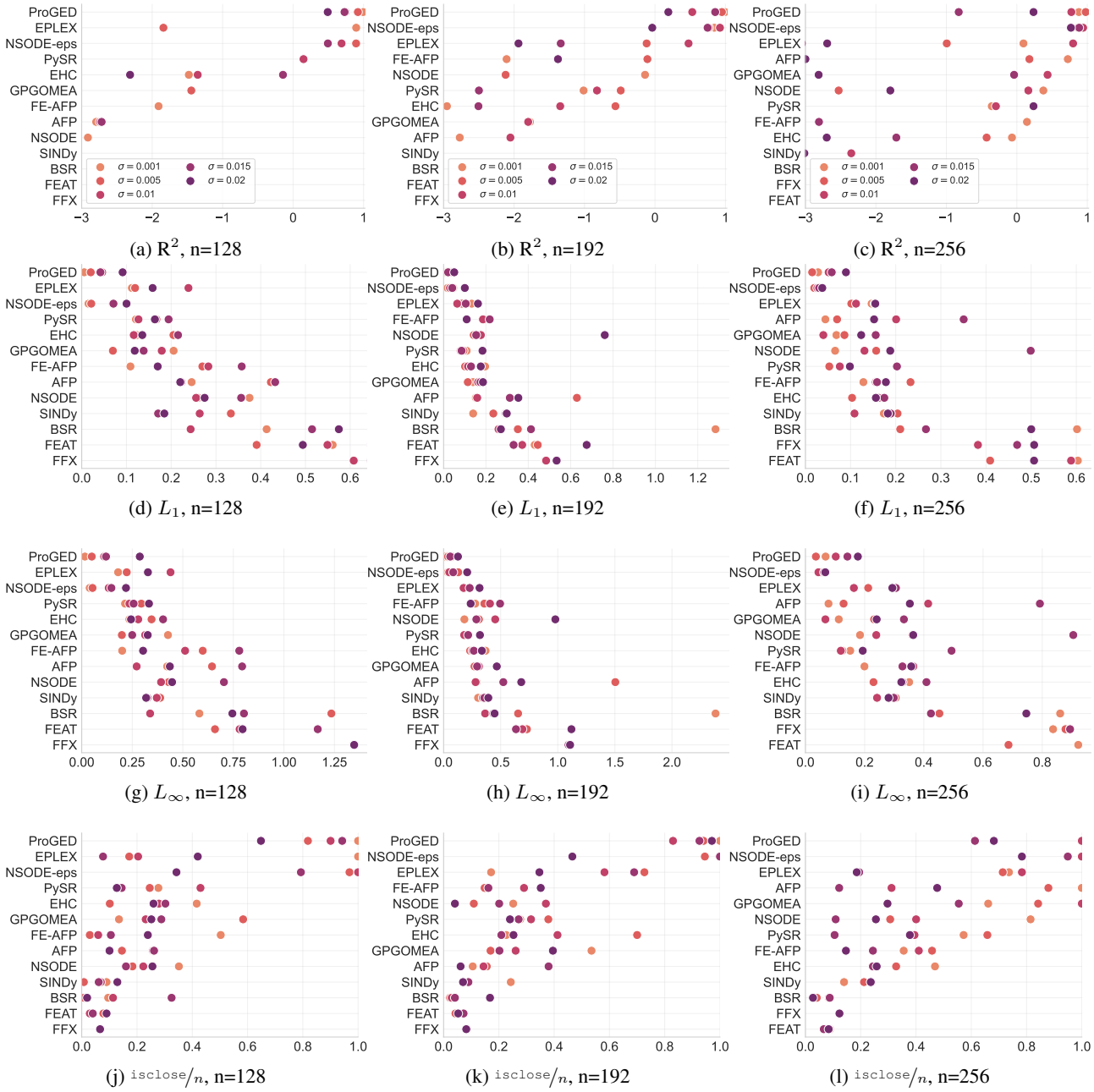
(a) R^2, n=128 (b) R^2, n=192 (c) R^2, n=256

(d) L_1, n=128 (e) L_1, n=192 (f) L_1, n=256

(g) L_∞, n=128 (h) L_∞, n=192 (i) L_∞, n=256

(j) $\texttt{isclose}/n$, n=128 (k) $\texttt{isclose}/n$, n=192 (l) $\texttt{isclose}/n$, n=256

Figure 9: Extrapolation. Median scores on **Classic** for n irregularly sampled time points across different noise levels σ.



(a) R^2, n=128 (b) R^2, n=192 (c) R^2, n=256

(d) L_1, n=128 (e) L_1, n=192 (f) L_1, n=256

(g) L_∞, n=128 (h) L_∞, n=192 (i) L_∞, n=256

(j) $\texttt{isclose}/n$, n=128 (k) $\texttt{isclose}/n$, n=192 (l) $\texttt{isclose}/n$, n=256

Figure 10: Extrapolation. Median scores on **Textbook** for n irregularly sampled time points across different noise levels σ.

(a) R^2, n=128 (b) R^2, n=192 (c) R^2, n=256

(d) L_1, n=128 (e) L_1, n=192 (f) L_1, n=256

(g) L_∞, n=128 (h) L_∞, n=192 (i) L_∞, n=256

(j) $\texttt{isclose}/n$, n=128 (k) $\texttt{isclose}/n$, n=192 (l) $\texttt{isclose}/n$, n=256

Figure 11: Extrapolation. Median scores on **Large** for n irregularly sampled time points across different noise levels σ.

F. Example trajectories

Below we provide a few selected trajectories alongside predictions obtained from NSODE-eps on dataset **Large** with noise of $\sigma = 0.01$ and $n = 192$ sampled time points. There are a few aspects to note: firstly, despite being autonomous scalar ODEs whose limit behavior is confined to convergence to an equilibrium or divergence, we can see that this function class can still exhibit rich and highly non-linear behavior before reaching this limit. This is perhaps most pronounced in Figure 12a and Figure 12c. Secondly, even though in our evaluation the two accuracy metrics R^2 and average `isclose` appear to be highly correlated we can see that they do not always capture the same phenomenon, compare e.g. in Figure 12b, Figure 12d and Figure 12f.

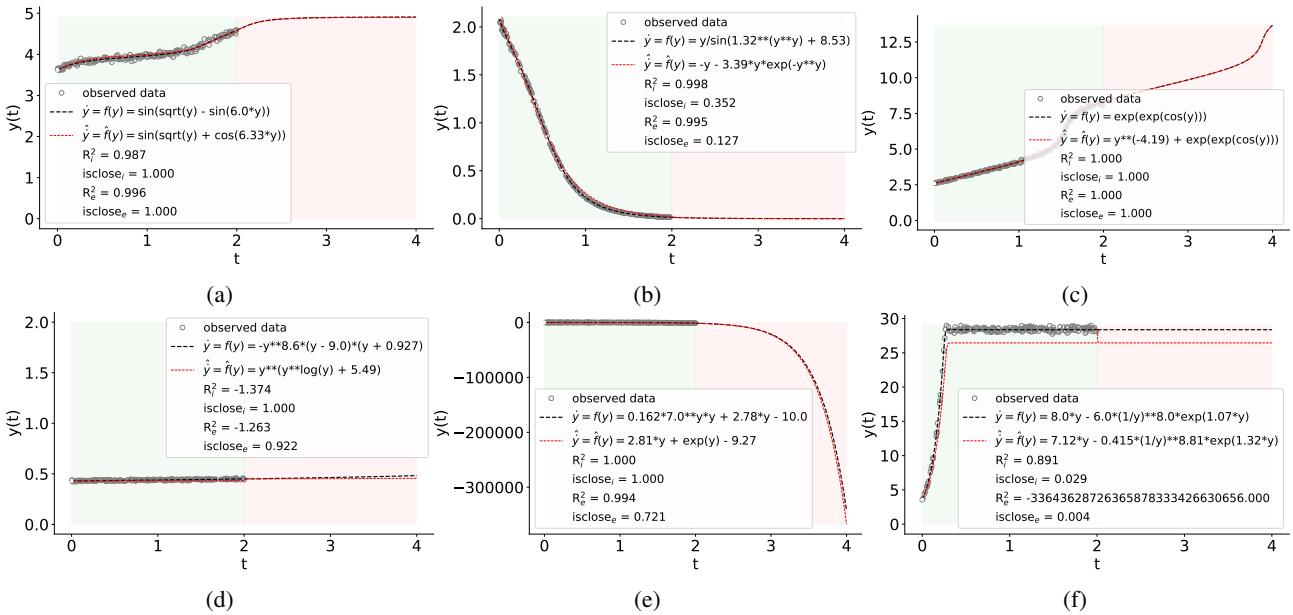


Figure 12: Different example trajectories and trajectories predicted by NSODE-eps with noise standard deviation of $\sigma = 0.01$ and $n = 192$ irregularly sampled points. Green and red area correspond to interpolation and extrapolation regimes.

G. Open Source Software acknowledgement

For this research project we heavily relied on available open source software packages which we list in Table 9.

Table 9: Overview of software packages we used in our work.

Name	Reference
Python	(van Rossum & Drake, 2009)
PyTorch	(Paszke et al., 2019)
Numpy	(Harris et al., 2020)
Pandas	(pandas development team, 2020; Wes McKinney, 2010)
Jupyter	(Kluyver et al., 2016)
Matplotlib	(Hunter, 2007)
Scikit-learn	(Pedregosa et al., 2011)
Seaborn	(Waskom, 2021)
SciPy	(Virtanen et al., 2020)
SymPy	(Meurer et al., 2017)
HuggingFace	(Wolf et al., 2020)
h5py	`https://www.h5py.org/`