ODEFORMER

Symbolic Regression of Dynamical Systems with Transformers

Authors: Stéphane d'Ascoli, Sören Becker, Alexander Mathis, Philippe Schwaller, Niki Kilbertus

Lancaster AI Reading Group

Cassandra Durr

May 6, 2026

TABLE OF CONTENTS

SECTION 1

Symbolic Regression

SYMBOLIC REGRESSION (SR)

SR: Generate mathematical expressions directly from (potentially noisy) data.

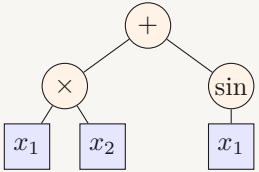
Functional SR

Estimating a function $f(\mathbf{x})$ symbolically from observed data, $(\mathbf{x}, f(\mathbf{x}))$.

Dynamical SR

Estimate differential equations $\frac{d\mathbf{x}_t}{dt} = f(\mathbf{x}_t)$ symbolically from observed trajectories $(t, \mathbf{x}_t)$.

$$f(x_1, x_2) = (x_1 \cdot x_2) + \sin(x_1)$$



SECTION 2

Model Components

METHOD

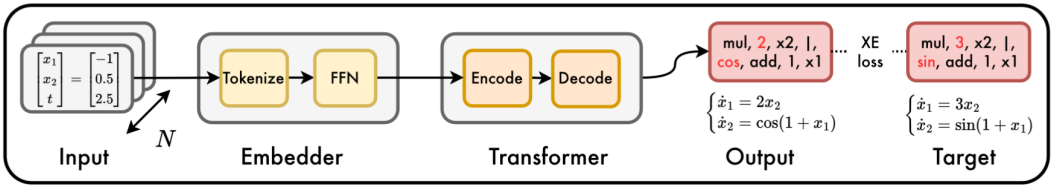


Figure 2: **Sketch of our method to train ODEFormer**. We generate random ODE systems, integrate a solution trajectory on a grid of N points $x \in \mathbb{R}^D$, and train ODEFormer to directly output the ODE system in symbolic form, supervising the predicted expression via cross-entropy loss.

Figure: (d'Ascoli et al. 2023)

SECTION 2.1

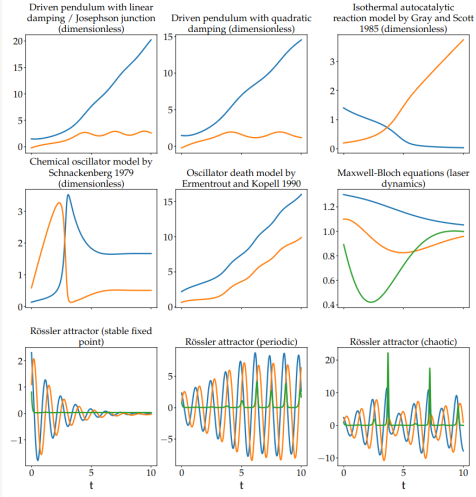
Data

Input data

- Trajectories: $(t_i, \mathbf{x}_i) \in \mathbb{R}^{D+1}$ for a D-dimensional ODE system ($i \in \{1, \ldots, N\}$)
- Goal is to learn:

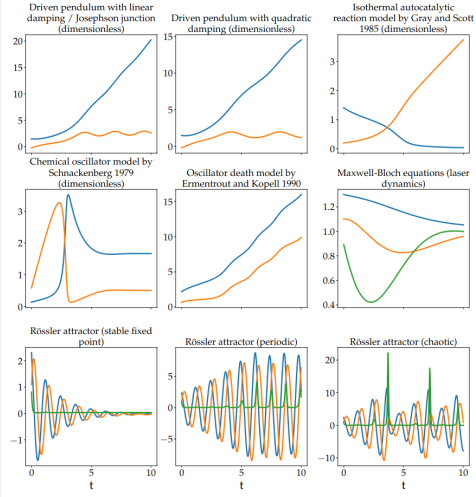
$$\frac{d\mathbf{x}}{dt} = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \vdots \\ \dot{x}_D \end{bmatrix} = \begin{bmatrix} f_1(x_1, x_2, ..., x_D) \\ f_2(x_1, x_2, ..., x_D) \\ \vdots \\ f_D(x_1, x_2, ..., x_D) \end{bmatrix}$$

- Train on noisy and irregularly sampled data



DATA DIMENSIONALITY & COMPLEXITY

- Allow a maximum ODE system dimension of 6.
  Q: How to generalise to higher-dimensional systems without retraining?
- In training, expressions are generated by sampling a maximum of 5 binary operators and 3 unary operators.
  Q: Bias towards sampling probability distribution in inference? How does the generation of ODE systems at pre-training bias the model at inference?



SECTION 2.2

Embedder

EMBEDDER

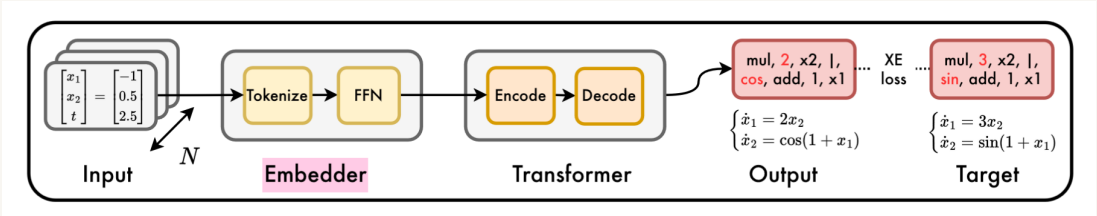


Figure: (d'Ascoli et al. 2023)

Tokenising numbers

- Symbolic expressions may contain floating point numbers.
- Tokenise continuous values into a fixed-size vocabulary.
- Example: 12.3586
  - Sign: $+$
  - Mantissa: 1236
  - Exponent: 10^{-2}

TOKENISATION

Tokenising numbers

- Symbolic expressions may contain floating point numbers.
- Tokenise continuous values into a fixed-size vocabulary.
- Example: 12.3586
  - Sign: $+$
  - Mantissa: 1236
  - Exponent: 10^{-2}

Questions/ Discussion

- **Precision**: Are there some systems for which the loss of precision will be most felt?
- **Mathematics**: Each token may not be represented in the training data. Without encoding mathematical knowledge, how will this model learn the 'value' of an infrequently seen token? How might we embed mathematical knowledge?

EMBEDDING NUMERICAL TRAJECTORIES

Data: $(t_i, \mathbf{x}_i) \in \mathbb{R}^{D+1}$ for a D-dimensional ODE system. For each data point $i \in \{1, \ldots, N\}$:

1. **Tokenise** to get token sequence $\in \mathbb{R}^{3(D+1)}$
2. **Embed** tokens into d_{emb} space and concatenate $\in \mathbb{R}^{3(D+1)d_{emb}}$
3. **Pad** system dimension $\in \mathbb{R}^{3(D_{\max}+1)d_{emb}}$ (*paper uses* $D_{\max} = 6$)
4. **Project** to a lower-dimension using a FFN (*2-layer with SiLU activations*) to get $\in \mathbb{R}^{d_{emb}}$

EMBEDDING NUMERICAL TRAJECTORIES

Data: $(t_i, \mathbf{x}_i) \in \mathbb{R}^{D+1}$ for a D-dimensional ODE system. For each data point $i \in \{1, \ldots, N\}$:

1. **Tokenise** to get token sequence $\in \mathbb{R}^{3(D+1)}$
2. **Embed** tokens into d_{emb} space and concatenate $\in \mathbb{R}^{3(D+1)d_{emb}}$
3. **Pad** system dimension $\in \mathbb{R}^{3(D_{\max}+1)d_{emb}}$ (*paper uses* $D_{\max} = 6$)
4. **Project** to a lower-dimension using a FFN (*2-layer with SiLU activations*) to get $\in \mathbb{R}^{d_{emb}}$

Questions/ Discussion

- It feels like the 2-layer FFN is doing quite a lot of heavy lifting – is there a better way to embed the input data?
- Is there a way to maintain system dimension throughout the model (vs. flattening the data)?

SECTION 2.3

Transformer

TRANSFORMER

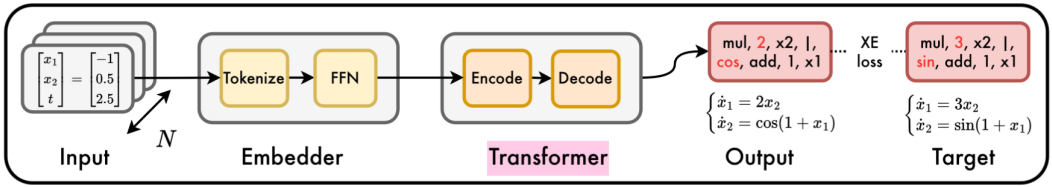


Figure: (d'Ascoli et al. 2023)

ENCODER

- Relatively shallow: 4 layers compared to 16 decoder layers
- No positional embeddings since the time component is included in the inputs
- Only concerned with numerical input trajectories, so its vocabulary only includes tokens for numbers

DECODER

Generating Symbolic Functions

- Extend vocabulary to include tokens for mathematical operators
- Separate different ODE dimensions by token '|'

Example: $f(x) = \cos(2.4242x)$

Represented as token sequence:

`[cos, mul, +, 2424, E-3, x]`

Questions/ Discussion

- How much compute is spent differentiating between similar numeric tokens vs generating unique and diverse functions?
- Would it make more sense to add a constant token for optimisation at a later stage? Or is this end-to-end approach superior?

DECODING STRATEGIES

Greedy search

- Picks the highest probability token at each step
- Computationally cheap
- Can get stuck in local optima (optimising locally for token $i + 1$)

DECODING STRATEGIES

Greedy search

- Picks the highest probability token at each step
- Computationally cheap
- Can get stuck in local optima (optimising locally for token $i + 1$)

Beam search

- Tracks the top k most probable sequences using cumulative probability
- At each step, keep only the k paths with the highest cumulative probability
- Optimises across the whole sequence rather than locally

DECODING STRATEGIES

Greedy search

- Picks the highest probability token at each step
- Computationally cheap
- Can get stuck in local optima (optimising locally for token $i + 1$)

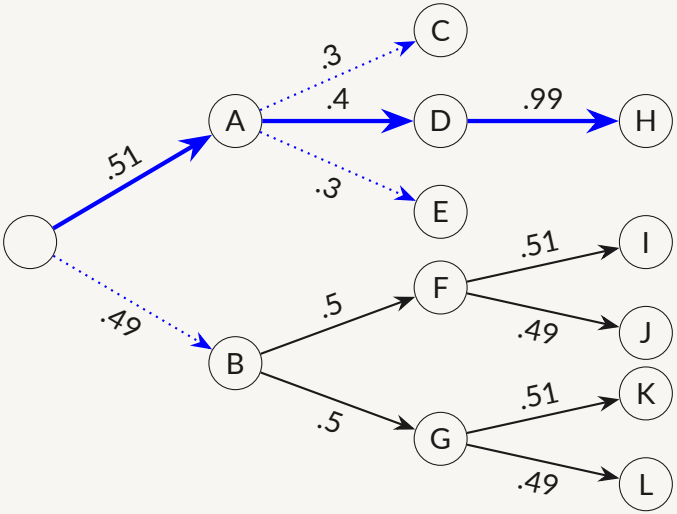
Beam search

- Tracks the top k most probable sequences using cumulative probability
- At each step, keep only the k paths with the highest cumulative probability
- Optimises across the whole sequence rather than locally

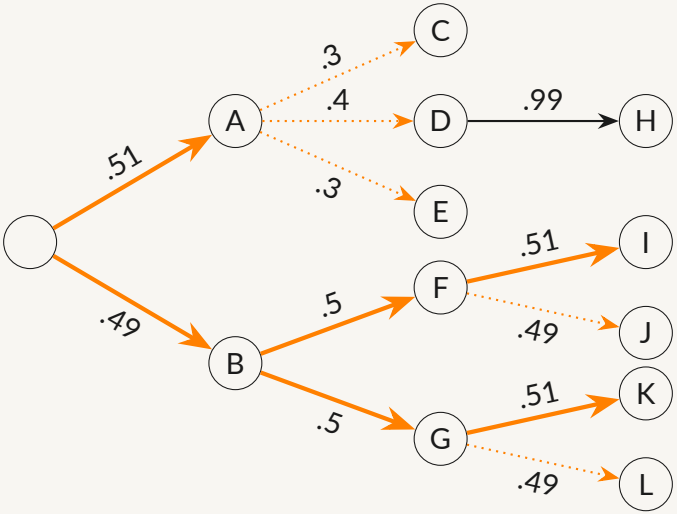
Beam sampling

- Tracks k sequences, but **samples** next tokens based on a probability distribution
- Injects randomness at each step to generate **diverse sequences**
- Stochasticity controlled by temperature hyperparameter

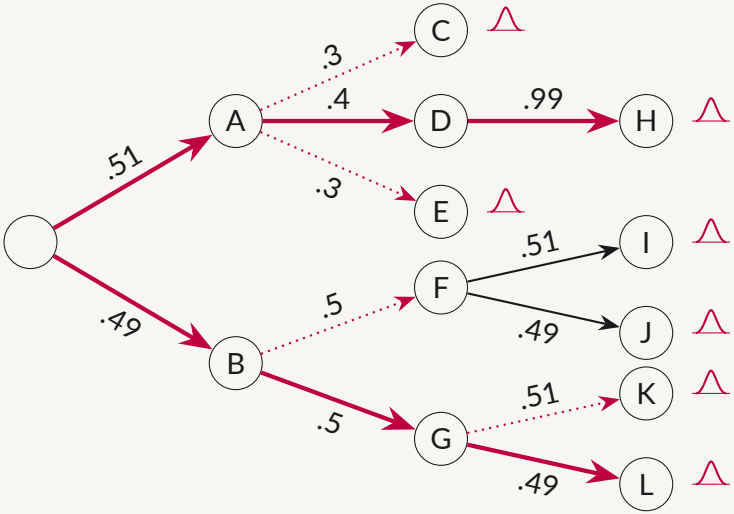
DECODING STRATEGIES: GREEDY SEARCH



DECODING STRATEGIES: BEAM SEARCH



DECODING STRATEGIES: BEAM SAMPLING



SECTION 2.4

Loss

LOSS

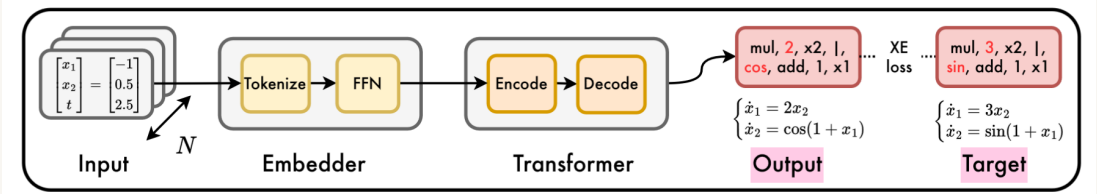


Figure: (d'Ascoli et al. 2023)

EVALUATION

Loss

- Training: cross-entropy loss (symbolic evaluation)
- Inference and refinement: Reconstruction loss (numerical evaluation)

Refinement

Add an optional, additional parameter optimisation step for the numeric tokens after the candidate equation is generated

Questions/ Discussion

- Symbolic vs numeric evaluation
- Mismatch between training loss (symbolic) and evaluation metrics (numerical)

SECTION 3

Results

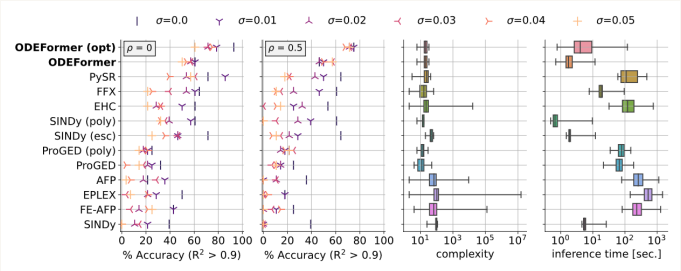
METHODS COMPARED

Table 1: **Overview of models.** f.d.: finite differences required, ode: method developed for dynamical SR, T: transformer-based, GP: genetic programming, MC: Monte Carlo, reg: regression

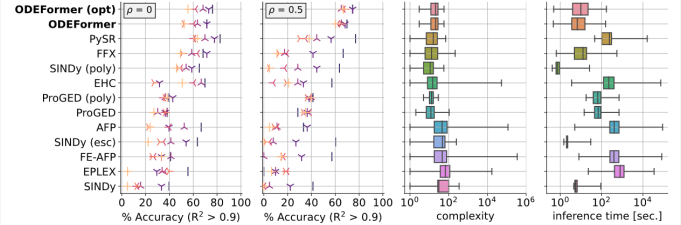
name	type	ode	f.d.	description	reference
ODEFormer	T	yes	no	seq.-to-seq. translation	ours
AFP	GP	no	yes	age-fitness Pareto optimization	[59]
FE-AFP	GP	no	yes	AFP with co-evolved fitness estimates	[59]
EHC	GP	no	yes	AFP with epigenetic hillclimbing	[60]
EPLEX	GP	no	yes	epsilon-lexicase selection	[15]
PySR	GP	no	yes	AutoML-Zero + simulated annealing	[20]
SINDy	reg	yes	yes	sparse linear regression	[37]
FFX	reg	no	yes	pathwise regularized ElasticNet regression	[61]
ProGED	MC	yes	no	MC on probabilistic context free grammars	[11]

Figure: (d'Ascoli et al. 2023)

RECONSTRUCTION RESULTS



(a) Reconstruction on Strogatz



(b) Reconstruction on ODEBench

GENERALISATION RESULTS

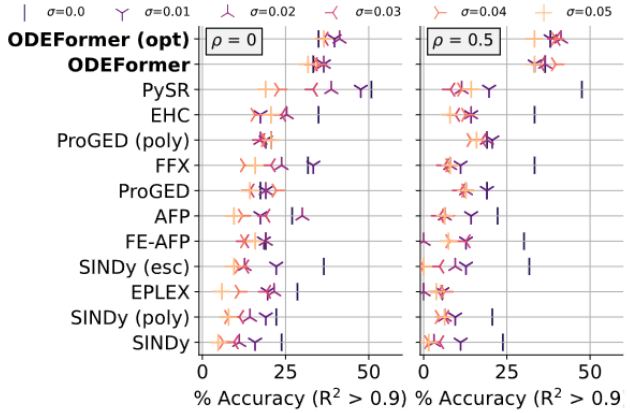


Figure 5: **Generalization on ODEBench.** We consider the same setting as in Figure 4.

SOME GOOD FITS

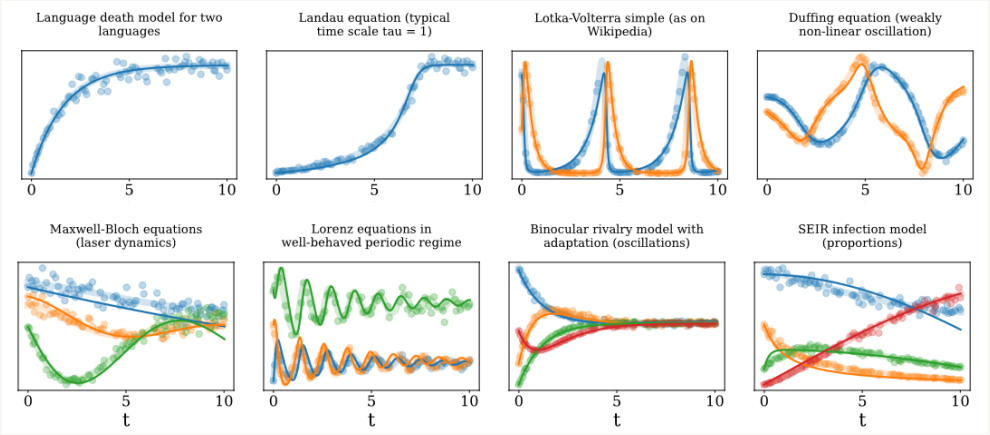


Figure: (d'Ascoli et al. 2023)

AND SOME BAD ONES...

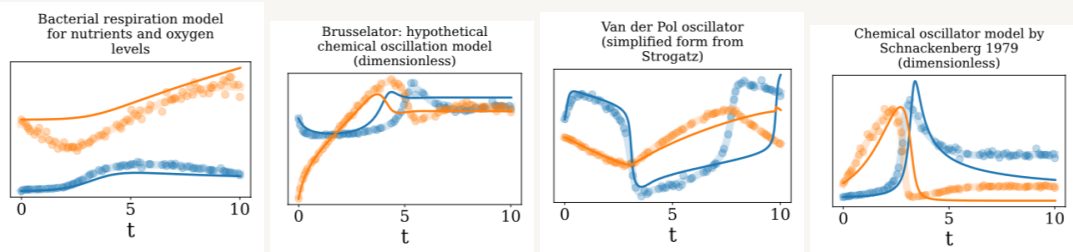


Figure: (d'Ascoli et al. 2023)

AND SOME BAD ONES...

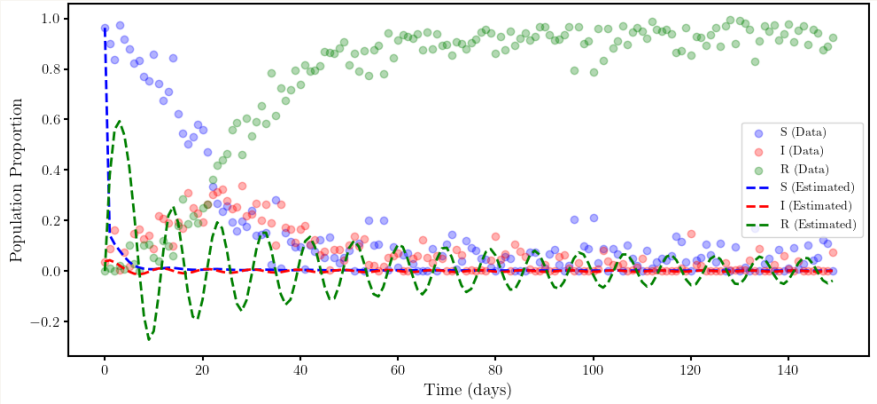


Figure: SIR Model (my dissertation)

SECTION 4

Future Directions

LIMITATIONS & PLANNED DIRECTIONS

- Limited to **fully observable systems** $\rightarrow$ proposed solution is to randomly mask variables during training by replacing their observations with a dedicated token (emulate unobserved variables)
- Limited to **first-order** ODEs $\rightarrow$ no proposed solution that avoids finite-differencing on noisy data
- Perform inference based on **single trajectories** only $\rightarrow$ allow multiple trajectories to 'improve identifiability'
- Poor performance on **chaotic** systems (similar to benchmarked systems)

REFERENCES

d'Ascoli, Stéphane et al. (2023). "ODEFormer: Symbolic Regression of Dynamical Systems with Transformers". In: *arXiv preprint arXiv:2310.05573*.

Thank you for your attention.
Any questions?

URL: https://arxiv.org/pdf/2310.05573