

A APPENDIX

A.1	Kolmogorov-Arnold Network	13
A.2	Technical implementation	14
A.3	Hardware specifications	14
A.4	Datasets generation and reproducibility specifics	14
A.5	Hyperparameters	15
A.6	Spline-Wise regression pseudo-code	15
A.7	Comparison with original Spline-Wise fitting algorithm	17
A.8	MAE Time Evolution	17
A.9	Symbolic Expressions Extracted from Synthetic Datasets	19
A.10	Ablation study of KAN without multiplication	19
A.11	Analysis with Noisy Inputs	19
A.12	Equations of Epidemiological Spreading	21
A.13	Country-specific coefficients fine-tuning	21
A.14	Plots of Epidemiological Spreading	21
A.15	Declaration on Generative AI	21

A.1 KOLMOGOROV-ARNOLD NETWORK

Kolmogorov-Arnold Networks (KANs) proposed by Liu et al. (2025) are a specific type of neural networks that have been recently proposed as valid alternatives to Multi-Layer Perceptrons (MLPs). Whereas MLPs are inspired by the universal approximation theorem, KANs are inspired by the Kolmogorov-Arnold representation theorem (Kolmogorov, 1961; Braun & Griebel, 2009), which states that if $f : [0, 1]^d \rightarrow \mathbb{R}$ is a multivariate continuous function on a bounded domain, then it can be written as:

$$f(\mathbf{x}) = f(x_1, x_2, \dots, x_d) = \sum_{q=1}^{2d+1} \Phi_q \left(\sum_{p=1}^d \phi_{q,p}(x_p) \right), \quad (7)$$

where $\phi_{q,p} : [0, 1] \rightarrow \mathbb{R}$, $\Phi_q : \mathbb{R} \rightarrow \mathbb{R}$. In other words, f can be reduced to a suitably defined composition of univariate functions, where the composition only involves simple addition. The underlying idea of KANs is to substitute weights and fixed activation functions of MLPs with learnable univariate activation functions on edges and sum aggregation on nodes.

The general definition of a KAN layer Φ_l with d_{in} -dimensional input and d_{out} -dimensional output consists of a matrix of univariate functions:

$$\Phi_l = \begin{bmatrix} \phi_{l,1,1}(\cdot) & \phi_{l,1,2}(\cdot) & \cdots & \phi_{l,1,d_{in}}(\cdot) \\ \phi_{l,2,1}(\cdot) & \phi_{l,2,2}(\cdot) & \cdots & \phi_{l,2,d_{in}}(\cdot) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_{l,d_{out},1}(\cdot) & \phi_{l,d_{out},2}(\cdot) & \cdots & \phi_{l,d_{out},d_{in}}(\cdot) \end{bmatrix} \quad (8)$$

where $\phi_{l,j,i}$ represents the learnable activation function applied to the i^{th} -feature of the input of the j^{th} -neuron at layer l . After computing all the $d_{in} \cdot d_{out}$ activation values, the output of the l^{th} layer $\mathbf{x}_l \in \mathbb{R}^{d_{out}}$ is obtained by summing along the first dimension of the matrix described in Eq. 8. Stacking multiple KAN layers results in an architecture with shape represented by an integer array:

$$[d_0, d_1, \dots, d_L],$$

where d_l represents the number of neurons of the l^{th} -layer and $d_0 = |\mathbf{x}|$. Each univariate function in Eq. 8 has trainable parameters that can be learned through backpropagation and gradient descent. Specifically, they are defined as splines with residual activations.

KANs are usually trained with a sparsity loss, which is an adaptation of the L1 norm of MLPs. However, this norm is directly defined on the learned activation functions. Formally, the L1 norm of an activation function ϕ is given by the average magnitude over its N_p inputs, that is:

$$|\phi|_1 \equiv \frac{1}{N_p} \sum_{s=1}^{N_p} |\phi(x^{(s)})|. \quad (9)$$

Then, the L1 norm of a KAN layer Φ with d_{in} inputs and d_{out} outputs is defined as:

$$|\Phi|_1 \equiv \sum_{i=1}^{d_{in}} \sum_{j=1}^{d_{out}} |\phi_{i,j}|_1, \quad (10)$$

that is, the sum of L1 norms of all the activation functions in Φ . Furthermore, an entropy term is added to the loss definition:

$$S(\Phi) \equiv - \sum_{i=1}^{d_{in}} \sum_{j=1}^{d_{out}} \frac{|\phi_{i,j}|_1}{|\Phi|_1} \log \left(\frac{|\phi_{i,j}|_1}{|\Phi|_1} \right). \quad (11)$$

Then, the final training loss \mathcal{L}_{total} is given by the prediction loss \mathcal{L}_{pred} plus the L1 and entropy regularization aggregated over all the layers:

$$\mathcal{L}_{total} = \mathcal{L}_{pred} + \lambda \left(\mu_1 \sum_{l=1}^L |\Phi_l|_1 + \mu_2 \sum_{l=1}^L S(\Phi_l) \right), \quad (12)$$

where μ_1, μ_2 , and λ are hyper-parameters that determine the impact of the corresponding loss terms.

One of the key characteristics of KANs is that they can be used to perform symbolic regression. Specifically, once the model is trained, we can prune inactive neurons by looking at their spline activation magnitudes ², and we can fix the remaining activation functions to symbolic formulas (e.g., \sin , \cos), so that the whole model can be described through a symbolic representation. This process enhances interpretability, since it overcomes the black-box nature typical of deep-learning models by providing as output a human-readable mathematical formulation of the learned function.

A.2 TECHNICAL IMPLEMENTATION

We implemented the model under consideration with Python 3.12.0, Pytorch 2.3.1, and PyG 2.3.1. For hyper-parameter tuning, we employed the Optuna package (Version 4.3.0). The Spline-Wise fitting procedure relies on the `curve_fit` method from `scipy` library for solving the non-linear least squares problem, while for the GP-based SR algorithm we used PySR 1.5.5. We used the `dopri5` solver from `torchdiffeq` library as a numerical integrator for computing the rollout metric MAE_{traj} , setting $\text{atol} = \text{rtol} = 10^{-5}$ for all models.

A.3 HARDWARE SPECIFICATIONS

We carried out the experiments on a Google Cloud `g2-standard-48` virtual machine, equipped with 48 vCPUs based on the Intel Cascade Lake CPU architecture and 192 GB of system memory. The setup was further accelerated by 4 NVIDIA L4 GPUs.

A.4 DATASETS GENERATION AND REPRODUCIBILITY SPECIFICS

Tab. 3 shows the general equations of the studied synthetic dynamical systems, while Tab. 1 reports the considered instantiations. Preliminary experiments with different dynamics parameters—as long as physically consistent—did not yield significant differences in terms of models’ learning capabilities, hence we set their magnitude with plausible values considered in the literature. The datasets are generated by numerically integrating these models with the Runge–Kutta method of order 5, implemented in the `solve_ivp` function from the `scipy` library, using absolute and relative tolerances of 10^{-12} to ensure high numerical precision. We simulate the dynamics on a Barabási–Albert (Barabási & Albert, 1999) graph with 70 nodes and attachment parameter $m = 3$, saving the solutions at $T = 2000$ regularly spaced time steps. We report in Tab. 4 the set of parameters to reproduce dataset generation.

Table 3: General equations of the considered dynamical processes.

Dynamics	Equation dx_i/dt
KUR	$\omega + K \sum_j A_{ij} \sin(x_j - x_i)$
EPID	$-\mu x_i + \beta \sum_j A_{ij} (1 - x_i) x_j$
BIO	$\alpha - \delta x_i - \kappa \sum_j A_{ij} x_i x_j$
POP	$-r x_i^b + \sigma \sum_j A_{ij} x_j^a$

Table 4: Parameters to reproduce the creation of the synthetic datasets.

Dynamics	Initial condition	T_{Start}	T_{End}
KUR	Uniform $[0, 2\pi]$	0	1
EPID	Uniform $[0, 1]$	0	2
BIO	Uniform $[0, 1]$	0	1
POP	Uniform $[-1, 1]$	0	10

²We refer to the original paper for details on the pruning procedure.

Table 5: Hyperparameter ranges of the MLPs in the GMLP-ODE models

Hyperparameter	Values
Hidden dimensions	[8, 64]
Activation function	{relu, softplus, tanh}
Dropout probability	[0.0001, 0.5]
Hidden layers	{1, 2}
Learning rate	[0.0005, 0.05]
Batch size	{16, 32, 64}

For the KUR dynamics, oscillator phases are initialized uniformly in $[0, 2\pi]$ to cover the full angular domain. For EPID and BIO dynamics, node states are chosen in $[0, 1]$ to represent normalized concentrations or infection probabilities. For POP, instead, we chose to initialize nodes in the interval $[-1, 1]$ to better expose the effect of the polynomial term x^a , as including both positive and negative values yields richer trajectories. Regarding the temporal horizons $T_{\text{Start}} T_{\text{End}}$, we set them to allow each dynamics to display its full evolution.

The intermediate validation set used to tune the hyper-parameters of the SR algorithms is obtained by simulating the dynamics on an additional Barabási–Albert graph with 100 nodes and the same attachment parameter used for the training graphs. The graphs used to generate the three OOD test sets are a BA graph with 70 nodes (analogous to the one used for training, but initialized with a different random seed), a Watts–Strogatz small-world graph with 50 nodes, and an Erdős–Rényi random graph with 100 nodes and edge probability 0.05. For validation and test datasets, we simulate the dynamics for $T = 1000$ steps.

Regarding the COVID-19 empirical dataset, we normalize the values to the range $(-1, 1)$ using a MinMax scaler. The scaler is fitted only on the training set (the first 80% of the data) to prevent data leakage. We perform the same pre-processing step also for H1N1 and SARS datasets before fine-tuning the coefficients of the learned symbolic formulas by neural models. During the evaluation phase, the same scaler is applied to transform the predicted values back to the original scale.

All data-generating code, along with its random seeds, is provided in the codebase.

A.5 HYPERPARAMETERS

This section lists the search spaces of the employed hyperparameters in the experimental analysis. Model selection is performed using the Optuna package, optimizing the MAE over 35 trials for synthetic dynamics, over 70 trials for dynamics with noise, and over 100 trials for COVID-19 data. All neural architectures are optimized using Adam for 1000 epochs with early stopping and patience parameters of 200 for synthetic dynamics and 300 for the real-world COVID dataset. Tabs. 5, 6 and 7 detail the hyperparameter ranges used for the neural-based models. For TPSINDy, we consider the default libraries of symbolic functions provided by the authors in the original implementation, including polynomial, trigonometric, fractional, and exponential terms.

Model selection of GP-based and Spline-Wise SR methods is performed via grid search according to the hyperparameter grids specified in Tabs. 8, 9, respectively.

A.6 SPLINE-WISE REGRESSION PSEUDO-CODE

Algorithm 1 describes the proposed Spline-Wise symbolic fitting procedure of KAN-based models. To ensure parsimony, in line 10 the coefficients with magnitudes below a threshold (ϵ) are pruned before complexity is computed. For example, the expression $x^3 + 10^{-5}x^2$ is considered to have a complexity of 1, not 4. As a measure of complexity, we use the `count_ops` function from `sympy` library, which measures the number of operations an expression contains.

Table 6: Hyperparameter ranges of the MLPs in the LLC models, where λ denotes the regularization parameter of the penalized loss function minimized during training.

Hyperparameter	Values
Hidden dimensions	[8, 64]
Activation function	{relu, softplus, tanh}
Hidden layers	{1, 2}
Learning rate	[0.0005, 0.05]
Batch size	{16, 32, 64}
Regularization λ	[0.0, 0.01]

Table 7: Hyperparameter ranges of the KANs in the GKAN-ODE models.

Hyperparameter	Values
Grid size	[5, 20]
Spline order	[1, 3]
Range limit	[-10, 10]
Hidden dimensions	[1, 6]
Regularization λ	[10^{-6} , 1.0]
Learning rate	[0.0005, 0.05]
μ_1	[0.1, 1.0]
μ_2	[0.1, 1.0]
Batch size	{16, 32, 64}

Table 8: Hyperparameter grid for the PySR algorithm.

Hyperparameter	Values
Number of iterations	{50, 100, 200}
Model selection	{Score, Accuracy}
Binary operators	{+, -, *, /}
Symbolic library \mathcal{F}	[exp, sin, neg, square, cube, abs, tan, tanh, ln, zero]

Table 9: Hyperparameter grid of the Spline-wise fitting algorithm. The *Model selection* parameter is used at line 28 of Algorithm 1, and defines whether to choose the function with the highest score (thus favoring simpler equations) or with the lowest log loss (thus favoring accuracy).

Hyperparameter	Values
Spline pruning threshold ρ	{0.01, 0.05, 0.1}
Coefficient pruning threshold ϵ	{0.001, 0.01, 0.1}
Model selection	{Score, Log loss}
Γ	[10^{-5} , 10^{-4} , 10^{-2} , 10^{-1} , 1]
Symbolic library \mathcal{F}	[identity, square, cube, exp, abs, sin, cos, tan, tanh, ln, zero]

Algorithm 1 Spline-wise Symbolic Regression for KAN

Require: Splines \mathcal{S} , function library \mathcal{F} , regularization grid Γ , training data (x, y) , coefficient pruning threshold ε , spline pruning threshold ρ

Ensure: Selected symbolic function f_ϕ^* for each $\phi \in \mathcal{S}$ and final symbolic formula f_{SW}

```

1:  $\mathcal{S}_{pruned} = \text{pruning}(\mathcal{S}, \rho)$ 
2: for each spline  $\phi \in \mathcal{S}_{pruned}$  do
3:   Collect inputs  $X_\phi$  and outputs  $Y_\phi$  from training data
4:    $\text{Results}_\phi = []$ 
5:   for each  $\gamma \in \Gamma$  do
6:      $\text{best\_state} = (\cdot)$ 
7:      $\text{best\_L} = \infty$ 
8:     for each candidate function  $f \in \mathcal{F}$  do
9:       Fit affine parameters  $\theta_{f,\phi}^* = (a, b, c, d)$ 
10:      Prune negligible coefficients  $< \varepsilon$  from  $f$ 
11:      Compute predictions  $\hat{Y}_{f,\phi} = f_\phi(X_\phi; \theta_{f,\phi}^*)$ 
12:       $\text{MSE} = \frac{1}{|X_\phi|} \sum (Y_\phi - \hat{Y}_{f,\phi})^2$ 
13:       $c = \text{Complexity}(f, \theta_{f,\phi}^*)$ 
14:       $L = \text{MSE} + \gamma \cdot c$ 
15:       $\ell = \log(\text{MSE})$ 
16:      if  $L < \text{best\_L}$  then
17:         $\text{best\_state} = (f, \theta_{f,\phi}^*, c, \ell)$ 
18:      end if
19:    end for
20:    Append  $\text{best\_state}$  to  $\text{Results}_\phi$ 
21:  end for
22:  Sort  $\text{Results}_\phi$  by complexity  $c$ 
23:   $\text{scores} = [\text{Results}_\phi[0][f], 0]$ 
24:  for each consecutive pair  $(c_1, \ell_1), (c_2, \ell_2)$  do
25:     $\Delta = (\ell_2 - \ell_1) / (c_2 - c_1)$ 
26:    Append  $-\Delta$  to  $\text{scores}$ 
27:  end for
28:  Select  $f_\phi^* =$  function with highest score
29: end for
30: Combine all  $f_\phi^*$  according to KAN's additive/multiplicative structure
31: Build final symbolic formula  $f_{SW}$ 
32: return  $\{f_\phi^*\}, f_{SW}$ 

```

A.7 COMPARISON WITH ORIGINAL SPLINE-WISE FITTING ALGORITHM

We compare the proposed algorithm for the Spline-Wise symbolic fitting with the original one introduced by the authors of KANs. Tab. 10 shows the complexity and MAE_{traj} of the best-validated symbolic expressions inferred by the original SW method. The results show that such a method is able to extract formulas that achieve very low MAE_{traj} , especially on EPID and BIO dynamics, but with very high structural complexity (thus less interpretable). In contrast, as shown in Fig. 1 and Tab. 2, our proposed approach consistently achieves a more favorable trade-off between accuracy and interpretability, maintaining low MAE_{traj} while yielding much more compact symbolic expressions. The hyperparameters grid employed for validating the original SW algorithm is shown in Tab. 11.

A.8 MAE TIME EVOLUTION

Fig. 3 shows the test MAE_{traj} over time obtained by the assessed models, both the trained neural-based architectures and the distilled symbolic expressions. We can observe that, on EPID, BIO and POP dynamics, GKAN-based models maintain the lowest error constantly over time, while for KUR, TPSINDy achieves the best performance.

Table 10: Performance and structural complexity of the best-validated Spline-wise symbolic formulas with the original SW fitting algorithm, on the four synthetic dynamical systems.

Dataset	Complexity	MAE _{traj}
KUR	8	$1.43 \cdot 10^{-3} \pm 2.39 \cdot 10^{-4}$
EPID	49	$1.40 \cdot 10^{-4} \pm 1.53 \cdot 10^{-5}$
BIO	81	$5.96 \cdot 10^{-5} \pm 4.82 \cdot 10^{-6}$
POP	24	$1.56 \cdot 10^{-2} \pm 8.28 \cdot 10^{-3}$

Table 11: Hyperparameter grid of the original Spline-Wise fitting algorithm.

Hyperparameter	Values
Spline pruning threshold ρ	$\{0.01, 0.05, 0.1\}$
Grid range	$\{(-10, 10), (-5, 5)\}$
Weight simple	$\{10^{-5}, 0.3, 0.7, 0.9\}$
Symbolic library \mathcal{F}	[identity, square, cube, exp, abs, sin, cos, tan, tanh, ln, zero]

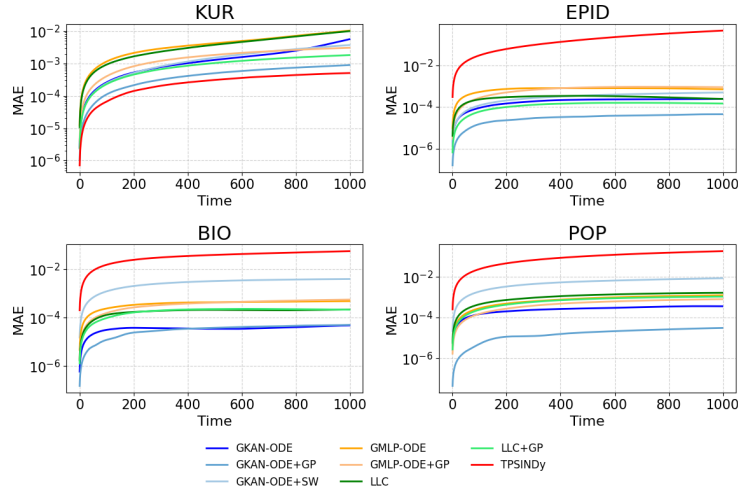


Figure 3: Evolution of test MAE_{traj} over time for each assessed model on synthetic dynamics. Values are averaged over the three test sets.

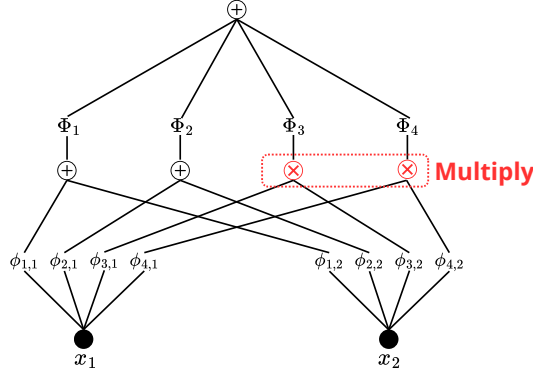


Figure 4: Representation of a KAN layer with the proposed multiplication enhancement (red) for a two-dimensional input ($d = 2$). ϕ are interpretable univariate splines.

A.9 SYMBOLIC EXPRESSIONS EXTRACTED FROM SYNTHETIC DATASETS

We show in Tab. 12 the learned symbolic expression on the four synthetic datasets by GMLP-ODE+GP, LLC+GP and TPSINDy methods. Despite the two neural-based models successfully extract the correct formulas' structures, the coefficients are slightly different from the ground truth, leading to a higher MAE_{traj} than GKAN-ODE+GP on every dynamics, as shown in Fig. 1.

Table 12: Learned symbolic expressions and their complexities across models and synthetic datasets.

Model	Dataset	Learned Expression	Complexity
GMLP-ODE + GP	KUR	$2.0009 + \sum_j A_{i,j}(-0.4971 \cdot \sin(x_i - x_j))$	5
	EPID	$-0.4990 \cdot x_i + \sum_j A_{i,j}(0.4976 \cdot x_j \cdot (1.0000 - x_i))$	6
	BIO	$-0.4970 \cdot x_i + 0.9987 + \sum_j A_{i,j}(-0.4989 \cdot x_i x_j)$	6
	POP	$-0.4998 \cdot x_i + \sum_j A_{i,j}(0.1973 \cdot x_j^3)$	5
LLC + GP	KUR	$1.9995 + \sum_j (-0.4986 \cdot \sin(x_i - x_j))$	5
	EPID	$-0.5012 \cdot x_i + \sum_j A_{i,j}(x_j \cdot (0.5005 - 0.5003 \cdot x_i))$	6
	BIO	$-0.4971 \cdot x_i + 0.9977 + \sum_j A_{i,j}(-0.4992 \cdot x_i x_j)$	6
	POP	$-0.4973 \cdot x_i + \sum_j A_{i,j}(0.1962 \cdot x_j^3)$	5
TP-SINDy	KUR	$2.0000 + \sum_j A_{i,j}(0.4994 \cdot \sin(x_j - x_i))$	5
	EPID	$-0.5679 + \sum_j A_{i,j}(0.2084 \cdot \exp(x_j - x_i))$	4
	BIO	$0.8670 + \sum_j A_{i,j}(-0.7113 \cdot x_i x_j)$	4
	POP	$-0.0162 + \sum_j A_{i,j}(0.0400 \cdot x_j + 0.0031 \cdot \sin(x_j))$	5

A.10 ABLATION STUDY OF KAN WITHOUT MULTIPLICATION

The architecture of a KAN layer with the proposed multiplicative enhancement is depicted in Fig. 4. In Fig. 5, we show the performance of GKAN-ODE models without multiplicative nodes (GKAN-ODE (no mult)) on the synthetic datasets. Despite the comparable or slightly worse MAE_{Traj} error of GKAN-ODE(no mult), formulas extracted with GP are comparable to those obtained by GKAN-ODE, while it is evident that the Spline-Wise fitting is unable to recognize the multiplicative term in EPID and BIO dynamics.

A.11 ANALYSIS WITH NOISY INPUTS

In the data-generating process, independent Gaussian noise is added to the node states at each time step, under three different signal-to-noise ratio (SNR) levels expressed in decibels (dB): 70 dB, 50 dB, and 20 dB. The performance of the assessed models in this setting is depicted in Fig. 6. The

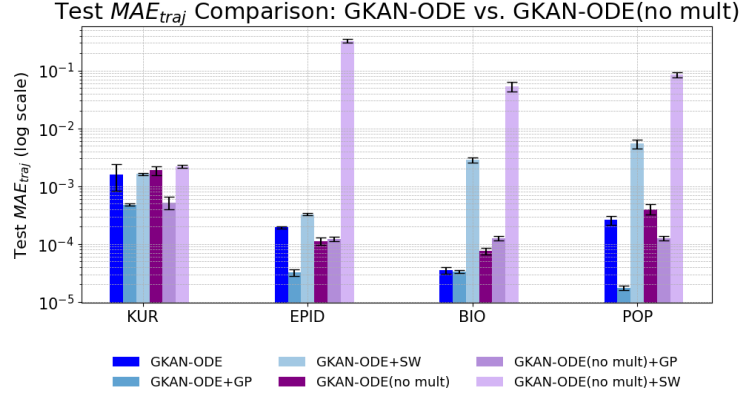


Figure 5: Performance comparison between GKAN-ODE models with and without multiplicative nodes.

methods are robust to noise up to 50 dB of SNR, particularly neural models, even though the quality of the expressions degrades with increasing values of noise. This is further exacerbated by the fact that we are estimating numerical derivatives, which are highly sensitive to noise and can amplify small fluctuations in the data.

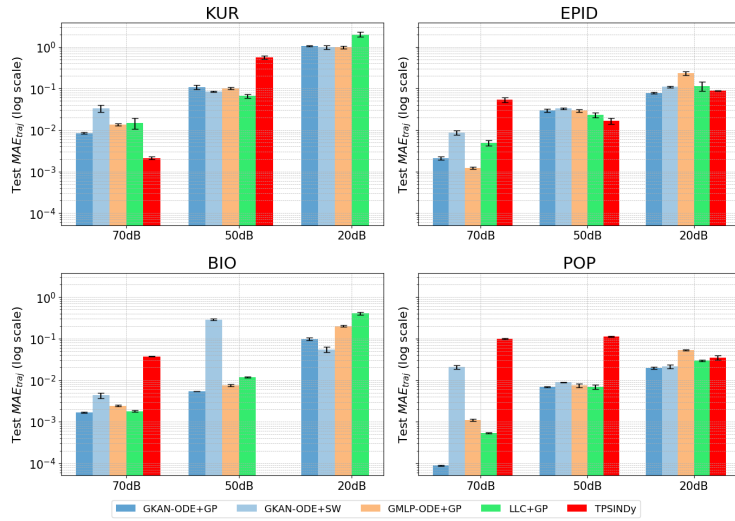


Figure 6: Performance of the extracted symbolic expression across various levels of SNR for each synthetic dataset. Missing values of TPSINDy are due to numerical divergences.

A.12 EQUATIONS OF EPIDEMIOLOGICAL SPREADING

Table 13: Symbolic expressions extracted from COVID-19 data as global dynamics, before country-specific fine-tuning. The LLC equation is re-derived from scratch, as the original work does not report all necessary coefficients needed for reproduction.

Model	Discovered Symbolic Expression	Complexity
TPSINDy	$a \cdot x_i + b \cdot \sum_j A_{ij} 1/(1 + e^{-(x_j - x_i)})$	7
LLC+GP	$a \cdot \tanh(x_i + b) + c \cdot \sum_j A_{ij} ((x_i - x_j) \cdot e^{-x_j})$	7
GMLP-ODE+GP	$a \cdot \ln(x_i + b) + \sum_j A_{ij} \ln(\tan(x_i + c)^2 + d)$	9
GKAN-ODE+GP	$ax_i + b + \sum_j A_{ij} (c \cdot e^{x_j})$	5
GKAN-ODE+SW	$a \cdot \tanh(b \cdot \tanh(cx_i + d) + e) - f \cdot \tanh(gx_i^3 + hx_i^2 - ix_i - j) + k$ $+ \sum_j A_{ij} (l \cdot \tanh(mx_i - n) - o \cdot \tanh(px_j - q) - r)$	30

A.13 COUNTRY-SPECIFIC COEFFICIENTS FINE-TUNING

To account for the heterogeneity of real-world epidemic dynamics, we fine-tune the coefficients of the generic symbolic structures discovered by neural-based models (detailed in Tab. 13) for each node. Specifically, we replace scalar constant terms in the symbolic equations with trainable parameters and optimize them via gradient descent. The optimization is performed by retraining the expressions on each node’s data on the first 80% of observations, using the subsequent 10% for validation, and leaving the final 10% for testing. Note that the LLC equation (and subsequent fine-tuning) is re-derived from scratch, as the original work does not report all necessary coefficients needed for reproduction. Instead, the TPSINDy formula is the one provided in the original paper. However, for a fair comparison with neural-based equations, we re-execute the fine-tuning algorithm used in the TPSINDy paper only on the first 90% of observations. This leads to a set of coefficients very similar to the original one, but that does not depend on the entire dataset, which is crucial when evaluating the generalization capabilities of a ML model.

A.14 PLOTS OF EPIDEMIOLOGICAL SPREADING

In Figs. 7 and 8, we show the performance of the learned symbolic expressions on COVID-19 data from Canada, Brazil, Turkey, and Serbia. Fig. 7 presents the predicted trajectories obtained through autoregressive integration, while Fig. 8 illustrates the results from short-term integration. As suggested by the performance comparison depicted in Fig. 2, neural-based models are able to capture the epidemiological spreading in both scenarios, while TPSINDy struggles on long term predictions.

A.15 DECLARATION ON GENERATIVE AI

The author(s) have employed Generative AI tools for proofreading and improving the readability of figures and tables.

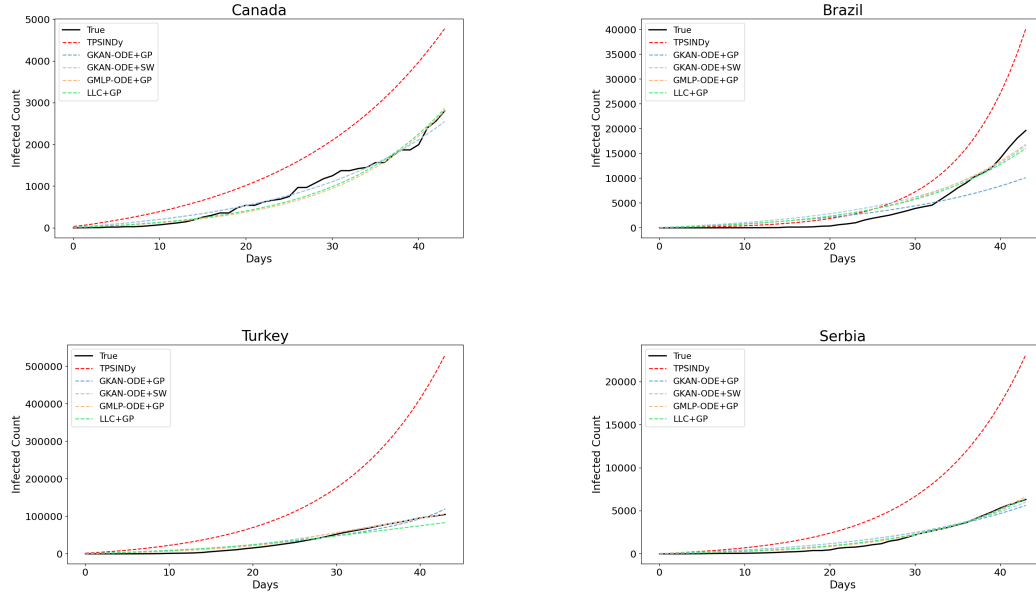


Figure 7: Predicted trajectories obtained by the long term (autoregressive) integration of the learned equations on COVID-19 data of Canada, Brazil, Turkey and Serbia.

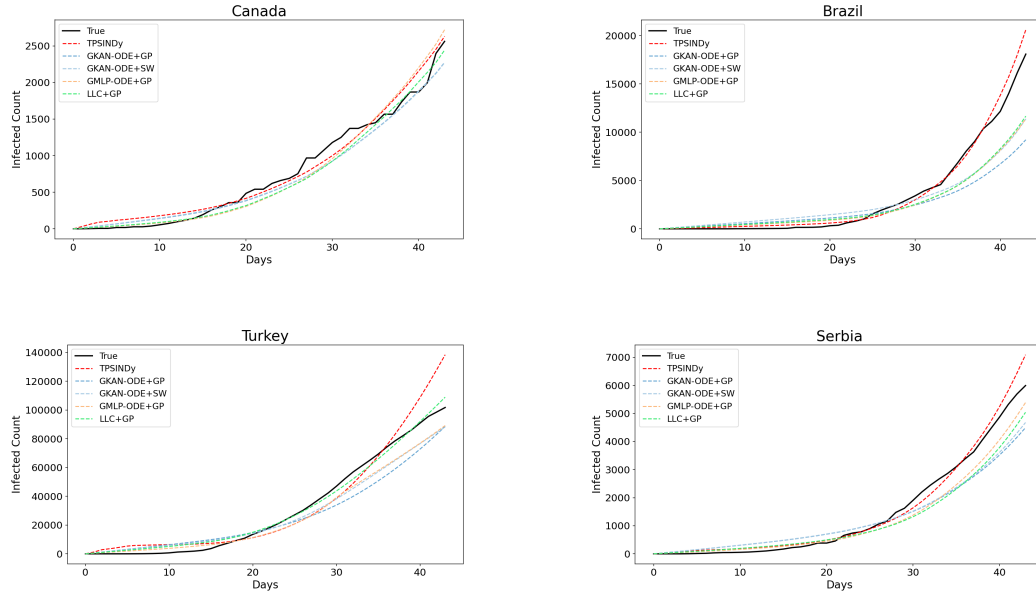


Figure 8: Predicted trajectories obtained by the short term integration of the learned equations on COVID-19 data of Canada, Brazil, Turkey and Serbia.