Micro-step Time-Series Regression: Insights from System Identification Using Symbolic Regression

Hengzhe Zhang[1], Alberto Tonda[2,3](✉), Qi Chen[1](✉), Bing Xue[1], Evelyne Lutton[2,3], and Mengjie Zhang[1]

[1] Centre for Data Science and Artificial Intelligence and School of Engineering and Computer Science, Victoria University of Wellington, PO Box 600, Wellington 6140, New Zealand

[2] UMR 518 MIA-PS, INRAE, Université Paris-Saclay, 91120 Palaiseau, France
{alberto.tonda,evelyne.lutton}@inrae.fr

[3] UAR 3611 Institut des Systèmes Complexes de Paris Île-de-France (ISC-PIF) CNRS, 75013 Paris, France

Abstract. Time-series forecasting is widely applied across various domains, yet most approaches rely on predefined time steps given by each problem. Based on observations from dynamic systems with known ground truth, we identify that large-step forecasts can lead to substantial errors due to insufficient modeling of continuous dynamics. To address this, we propose a micro-step time-series regression technique that decomposes predictions into smaller intervals, so that genetic programming-based feature construction can capture finer temporal patterns to improve the prediction performance. Specifically, we employ linear interpolation to allow the evolutionary feature construction process to learn from incremental changes, reducing the difficulty of time-series regression. Experiments on 100 datasets from the M4 forecasting benchmark demonstrate that micro-step regression significantly enhances prediction accuracy compared to traditional methods using raw time steps. Further analysis reveals that features trained on micro-step data evolve into simpler structures, promoting both generalization and interpretability.

Keywords: Time-Series Forecasting · Evolutionary Feature Construction · Genetic Programming · Dynamic Systems

1 Introduction

Most real-world phenomena develop over time [4]. To effectively model these dynamics, time-series regression techniques are widely used. Formally, given a time-series $\{y_t\}_{t=1}^{T}$, where y_t represents the observed value at time t, the objective of time-series regression is to predict future values based on past observations. Specifically, using evolutionary feature construction [13,25] for time-series regression, the goal is to search for an optimal set of features Φ, constructed via genetic programming (GP), along with a machine learning model f, such that:

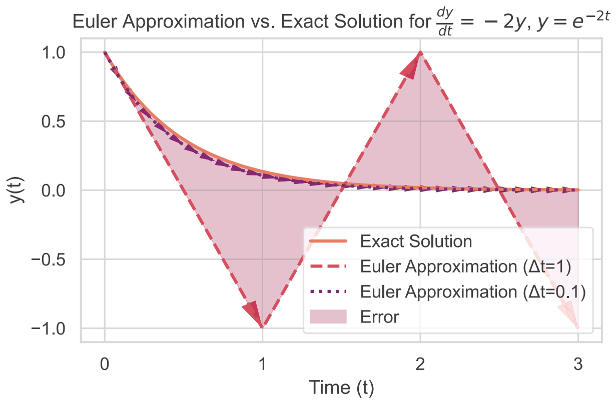


Fig. 1. Euler Approximation vs. Exact Solution. The arrow lines represent the first Euler approximation based on the ground truth, with each arrow indicating a prediction step based on the Euler Approximation.

$$\hat{y}_{t+1} = f(\varPhi(y_t, y_{t-1}, \ldots, y_{t-p+1})) \tag{1}$$

where $\hat{y}_{t+1}$ denotes the predicted value at time $t+1$, and $y_t, y_{t-1}, \ldots, y_{t-p+1}$ are the past p observations. Equation (1) can be straightforwardly extended to multi-step prediction: for a horizon of h future values, $\hat{y}_{t+1}, \hat{y}_{t+2}, \ldots, \hat{y}_{t+h}$, predictions can be made iteratively, using prior predictions as inputs in a recursive manner.

In time-series forecasting, non-stationarity often arises, meaning that statistical properties, such as $\mu_t = \mathbb{E}[y_t]$, change over time. A widely used approach to handle non-stationarity is the autoregressive integrated moving average (ARIMA) model [1], which addresses this issue through differencing. Rather than directly estimating $\hat{y}_{t+1}$, ARIMA with first-order differencing estimates the change $\varDelta y$ over a single time unit at each step t, with the next value calculated as $\hat{y}_{t+1} = y_t + \varDelta y$. The unit interval-whether a day, hour, or minute-varies depending on the application. From the perspective of system identification, this approach represents a first-order approximation of the system's dynamics, akin to the Euler method [8]. However, real-world systems often evolve continuously rather than discretely, meaning that a single time step may only approximate these gradual changes. This limitation can lead to substantial errors when predicting over longer intervals, as the model is forced to extrapolate over time spans where finer-scale dynamics are active but not fully captured.

In this paper, we observe that in time-series forecasting, even when the ground truth ordinary differential equation (ODE) $\frac{dy}{dt}$ is known, directly predicting the value at the next time step $t+1$ using the approximation $y_t + \frac{dy}{dt}$ can lead to significant errors. This issue is demonstrated in Fig. 1, where the ground truth ODE is $\frac{dy}{dt} = -2y$. Suppose a machine learning model successfully identifies the ground truth ODE. In this case, predicting the next value y_{t+1} based on the current value y_t becomes $y_{t+1} = y_t - 2y_t$. However, as shown in the figure, this approach yields inaccurate results due to the large step size, which prevents the approximation from capturing the system's gradual, continuous changes over the unit time interval. This limitation highlights the inadequacy of conventional time-series forecasting methods in accurately modeling the continuous evolution of dynamic systems.

Motivated by this challenge, we propose a micro-step time-series regression method based on a linear interpolation technique. Rather than predicting the value at $t + 1$ directly from the value at time step t, we decompose the time-series regression task into multiple smaller steps, allowing the model to make autoregressive predictions over micro intervals. This approach more effectively captures the continuous nature of the underlying dynamics, enhancing both prediction accuracy and robustness[1].

The key contributions of this paper are as follows:

– We analyze and demonstrate the impact of inappropriate step sizes in dynamic systems prediction. Specifically, we observe that even when the ground truth values are known, large step sizes can lead to inaccurate predictions, highlighting a fundamental limitation in standard time-series forecasting models.
– We propose a novel data-augmentation technique based on linear interpolation, designed to enrich the training data for fitness evaluation in GP. By augmenting the training set, evolutionary feature construction can capture fine-grained temporal dependencies, thereby allowing the time-series forecasting model to predict small, continuous changes instead of large, discrete jumps, achieving better prediction accuracy.

2 Background

2.1 Genetic Programming for Time-Series Regression

Genetic programming (GP) has been widely applied to time-series regression tasks, including applications like quality of service forecasting [6] and streamflow prediction [17]. GP is valued for its ability to evolve interpretable models that capture nonlinear and complex temporal dependencies [18]. Early work demonstrated that GP could compete with traditional methods such as Auto-Regressive Integrated Moving Average (ARIMA) [1] and Exponential Smoothing (ETS) [2] in time-series forecasting. Recent advancements have further enhanced GP by incorporating unit compatibility into the objective function [18] and integrating it into neuroevolution frameworks [19]. Additionally, GP has been applied as a routing model to direct different segments of time-series data to specialized base models [11]. Despite these innovations, most approaches rely on fixed time steps predefined by the problem. The impact of time-step granularity on prediction accuracy remains an open area for research.

2.2 Dynamic Systems and Time-Series Forecasting

A dynamic system describes how the values of state variables evolve over time, typically modeled by ordinary differential equations (ODEs). Formally, an ODE system with n state variables $y^{(1)}(t), y^{(2)}(t), \ldots, y^{(n)}(t)$ can be defined as:

[1] Source code: https://anonymous.4open.science/r/MicroStepSR/experiment/methods/MicroSR.py.

$$\begin{cases} \frac{dy^{(1)}}{dt} = -0.28 \cdot y^{(1)} \cdot y^{(3)} \\ \frac{dy^{(2)}}{dt} = 0.28 \cdot y^{(1)} \cdot y^{(3)} - 0.47 \cdot y^{(2)} \\ \frac{dy^{(3)}}{dt} = 0.47 \cdot y^{(2)} - 0.3 \cdot y^{(3)} \\ \frac{dy^{(4)}}{dt} = 0.3 \cdot y^{(3)} \end{cases}$$

$$y^{(1)}(t_0) = 0.6,\ y^{(2)}(t_0) = 0.3,$$
$$y^{(3)}(t_0) = 0.09,\ y^{(4)}(t_0) = 0.01$$

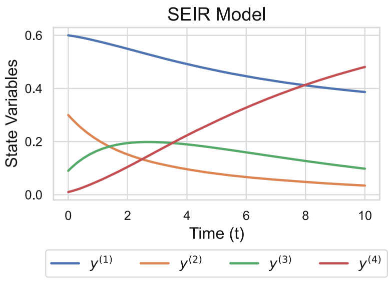


Fig. 2. Example of a 4-variable ODE system, and the corresponding trajectory of each state variable for the given initial conditions. The system in the example is the Susceptible-Exposed-Infectious-Recovered (SEIR) model used in epidemiology, set with parameters and initial conditions from the ODEBench system identification benchmark suite [5].

$$\frac{dy^{(i)}}{dt} = f_i(t, y^{(1)}, y^{(2)}, \ldots, y^{(n)}), \quad i = 1, \ldots, n, \tag{2}$$

where $\frac{dy^{(i)}}{dt}$ represents the rate of change of the i-th variable with respect to time t, and f_i defines the relationship of each state variable's derivative to time and the other state variables.

An ODE system can be solved given initial conditions $y^{(i)}(t_0)$ for $i = 1, \ldots, n$, a time step Δt, and a maximum time T, producing a trajectory that represents the values of each state variable over the interval $[t_0, T]$. An example of a 4-variable ODE system and its trajectory is shown in Fig. 2.

In practice, the first-order Euler method [8] can be used to approximate values over time. The value at the next time step t_{n+1} can be approximated as:

$$y_{n+1} = y_n + h \cdot f(t_n, y_n), \tag{3}$$

where h is the step size, y_n is the value at step n, and $f(t_n, y_n)$ is the derivative at step n. However, the accuracy of the first-order Euler method can be insufficient for systems with complex dynamics, motivating the exploration of finer-grained prediction techniques.

2.3 Genetic Programming for System Identification

System Identification (SI) involves the task of reconstructing a predictive model of a dynamic system [9], typically represented by a set of Ordinary Differential Equations (ODEs), starting from data points obtained from temporal trajectories. Symbolic Regression (SR), a technique capable of reconstructing free-form equations from data [22], is a natural choice for SI. Due to its flexible representation and gradient-free search mechanism, GP has been widely adopted as a symbolic regression method to solve the SI problem [10,24]. Nevertheless, several other techniques have also been applied to SI, ranging from general black-box neural networks [3] to physics-informed neural networks for specific applications [21], and transformer-based approaches that deliver human-readable ODEs [5].

While relatively small ODE benchmarks for SI have been available for a while, a recent publication [5] introduced a thorough system identification benchmark suite, ODEBench, with 63 different systems, along with experimental results for various SI techniques. The introduction of ODEBench now makes it possible to study system identification techniques in a more comprehensive manner.

3 Motivations and Modeling

The motivation for micro-step time-series regression stems from the field of system identification, particularly the analysis of how the interval between data points, Δ_t, influences forecasting accuracy on ODEBench benchmarks with known ground truth ODEs.

3.1 The ODEBench Benchmark Suite

ODEBench [5] is a Python-based benchmark suite designed for system identification of ODE systems. It includes 63 distinct combinations of systems and parameters, covering a variety of ODE configurations: 23 single-variable systems, 28 two-variable systems, 10 three-variable systems, and 2 four-variable systems. For each system, ODEBench provides both a training trajectory and a test trajectory, each generated from different initial conditions over the time interval $[0, 10]$, with uniformly sampled points spaced by Δ_t.

3.2 Influence of Δ_t When Ground Truth is Known

Similar to time-series regression, system identification can be transformed into a standard tabular learning format. A common approach is based on the Forward Euler method [8]. Specifically, given a state variable y, the goal of the Forward Euler method is to learn a symbolic model $F_y(\Delta_t, y)$ that predicts the incremental change in state over Δ_t in a single step, specifically $y_{t+\Delta_t} - y_t$. For a known ground truth ODE $y'(t)$, the model F_y can be expressed as $F_y(\Delta_t, y) = y'(t) \times \Delta_t$, derived from the mathematical relationship: $\frac{\partial F_y(\Delta_t,y)}{\partial \Delta_t}\Big|_{\Delta_t=0} = y'(t)$.

Since ODEBench provides the ground truth equations for each system, the ground truth incremental changes $y_{t+1} - y_t$ for each state variable can also be computed. Analyzing these ground truth values and the forecasts made by $F_y(\Delta_t, y)$ using the ground truth $y'(t)$ reveals that some cases yield considerably lower performance, as illustrated in Table 1. However, detailed examination shows that performance improves significantly when the default interval $\Delta_t \approx 0.06$ in ODEBench is reduced to $\Delta_t \approx 0.03$ or $\Delta_t \approx 0.015$. This improvement arises because the system evolves continuously over the interval between

Table 1. Impact of Reduced Discretization Interval (Δ_t) on R^2 Performance Metrics Across Various Systems and State Variables.

System Id	**State Variable**	**Trajectory**	R^2 **for** $\Delta_t \approx 0.06$	R^2 **for** $\Delta_t \approx 0.03$	R^2 **for** $\Delta_t \approx 0.015$
11	x_0	1	0.002478	0.769415	0.948090
		2	0.957637	0.990291	0.997697
26	x_0	1	0.531413	0.896984	0.976729
		2	0.838427	0.964003	0.991650
	x_1	1	0.670316	0.926977	0.983374
		2	0.884917	0.974147	0.993958
41	x_0	1	0.835624	0.961387	0.991028
		2	0.849260	0.924593	0.980166
	x_1	1	0.993183	0.998353	0.999599
		2	0.988084	0.997064	0.999260
48	x_0	1	0.897235	0.975271	0.994358
		2	0.976794	0.994000	0.998487
	x_1	1	0.973322	0.993513	0.998514
		2	0.996567	0.999107	0.999775
49	x_0	1	0.946908	0.987823	0.997125
		2	0.854351	0.962177	0.990543
	x_1	1	0.976213	0.994499	0.998698
		2	0.883269	0.968993	0.992211

y_t and $y_t + \Delta_t$. Predicting over a large interval Δ_t may fail to precisely approximate the integral: $\int_t^{t+\Delta_t} y'(t)\,dt$, leading to significant errors. By reducing Δ_t, predicting the next state with $y_t + F_y(\Delta_t, y)$ becomes better aligned with the continuous dynamics of the system, thus resulting in more accurate forecasts.

Based on the experimental results in Table 1, a key conclusion is that a large discretization interval can lead to poor predictive accuracy, even when the ground truth function is known. In such cases, a machine learning model may attempt to fit a complex function to capture all points. However, this complex function often could not represent the true underlying dynamics, leading to poor generalization. More concretely, in the context of time-series regression, if data is sampled at a weekly interval while the ground truth model represents a daily-evolving system, it may be difficult to learn an accurate model based on the weekly samples alone. Even if a model is found that fits all points in the training data, it may generalize poorly to unseen data.

4 Proposed Approach

4.1 Base Learner

In this paper, we aim to enhance traditional ARIMA-based time-series regression by integrating GP constructed features. Each GP individual consists of m GP trees, $\phi_1, \ldots, \phi_m$, where each tree represents a distinct constructed feature derived from the time-series data. These constructed features are applied to an ARIMA model (p, d, q) to make predictions, where p is the autoregressive order, d is the differencing degree for stationarity, and q is the moving average order.

Let $\{y_t\}$ denote the observed time-series at time t. In this work, we employ first-order differencing ($d = 1$) to address non-stationarity, defined as:

$$y'_t = y_t - y_{t-1}. \tag{4}$$

The model is designed to predict the incremental change y'_t rather than the raw next value y_t, allowing it to capture short-term variations more effectively. With GP-constructed features, the ARIMA-based model for predicting y'_t is defined as:

$$y'_t = c + \sum_{i=1}^{m} \beta_i \phi_i(\{y_{t-j} \mid j = 1, \dots, p\}, \{\epsilon_{t-k} \mid k = 1, \dots, q\}), \tag{5}$$

where c is a constant, β_i are feature weights, and $\phi_i(\{y_{t-j} \mid j = 1, \dots, p\}, \{\epsilon_{t-k} \mid k = 1, \dots, q\})$ represents the constructed features derived from past values y_{t-j} (for $j = 1, \dots, p$) and past errors ϵ_{t-k} (for $k = 1, \dots, q$). When predicting multiple steps ahead on unseen data, as in a long-horizon prediction $t + k$, the past errors used in predictions include those derived from future predicted points, such as ϵ_{t+k-1}. Since calculating errors for unseen data is challenging due to the absence of future labels y_{t+k-1}, we set $q = 0$ to omit the moving average term, focusing on autoregressive and differencing terms. To alleviate overfitting, we apply L2 regularization to the coefficients β_i, with a regularization strength of $\lambda = 1$.

4.2 Evolutionary Framework

The proposed algorithm is built upon a conventional multi-tree GP based evolutionary feature construction framework. The framework consists of the following steps:

- Population Initialization: A population of GP individuals is initialized, where each individual comprises multiple GP trees [14,26]. Each tree is initialized using the ramped-half-and-half method. The functional set includes mathematical operators (as detailed in Sect. 5.4), and the terminal set consists of lag features $y_{t-1}, \dots, y_{t-p}$.
- Fitness Evaluation: For each individual Φ, m features are constructed, and an ARIMA model is trained on these features to forecast the time-series values. Specifically, the data are augmented using a linear interpolation technique, expanding the time-series from T to $\hat{T} = T \times (k + 1)$, where k is the augmentation parameter. The details of this augmentation process are provided in Sect. 4.3. The training loss for each individual Φ on a specific training case t_j, representing a specific time step, is denoted by $L_{i,j}$. Here, $L_{i,j}$ measures the forecasting error of individual Φ_i on t_j in terms of prediction accuracy for that time step. Collectively, these errors form the loss matrix $\mathbf{L} \in \mathbb{R}^{n \times \hat{T}}$, where n is the number of individuals in the population and $\hat{T}$ is the number of training cases, providing a semantic information for selection.

- Parent Selection: Automatic ϵ-Lexicase selection [12] is applied to select individuals based on their diverse performance across training cases, leveraging the loss matrix $\mathbf{L} \in \mathbb{R}^{n \times \hat{T}}$ from a population with n individuals. Rather than aggregating the time-series forecasting errors, ϵ-Lexicase selection filters individuals sequentially by evaluating each training case in a random order. For a given training case t_j, only individuals meeting the condition $L_{i,j} \leq \min_k L_{k,j} + \epsilon_j$ are retained, where $\min_k L_{k,j}$ is the minimum loss on t_j across all individuals, and ϵ_j is set as the median absolute deviation. This filtering process continues through the sequence of training cases until only one individual remains, which is selected as the parent.
- Offspring Generation: Random subtree crossover and mutation are used to crossover and mutate the selected parents to generate offspring.
- Archive Maintenance: The best individual with the highest fitness is stored in an archive, which will be used for final predictions on unseen data. The archive is used to avoid the lose of the best individual during the evolutionary process.

The steps of fitness evaluations, parent selection, offspring generation, and archive maintenance are repeated iteratively until a stopping criterion is met.

4.3 Training for Micro-step Regression

To implement micro-step time-series regression, we augment the training data to enable the model to learn predict small incremental steps rather than predicting the value at step $t+1$ directly from step t. Specifically, we achieve this by linearly interpolating data points between step t and step $t+1$. Formally, given an augmentation parameter k, the interpolated values are generated as:

$$y_{t+\frac{i}{k+1}} = y_t + \frac{i}{k+1} \cdot (y_{t+1} - y_t), \quad i = 1, 2, \ldots, k. \tag{6}$$

This augmentation results in each time step t producing k interpolated points. For a time-series training set with T steps, the augmented time-series $\tilde{y}$ is thus represented as:

$$\tilde{y} = \{y_1, y_{1+\frac{1}{k+1}}, y_{1+\frac{2}{k+1}}, \ldots, y_2, \ldots, y_T\}. \tag{7}$$

The augmented series $\tilde{y}$ is then utilized in the fitness evaluation stage to compute fitness values, as described in Sect. 4.2.

4.4 Prediction Based on Micro-Step Regression

In micro-step time-series regression, the prediction is made for fractional steps rather than a full step. To forecast a future time step $\hat{y}_{T+h}$, the model first predicts $h \times (k+1)$ fractional increments and then aggregates these increments to obtain the final forecast:

$$\hat{y}_{T+h} = y_T + \sum_{i=1}^{h} \sum_{j=1}^{k+1} \hat{\Delta \tilde{y}}_{T+i,j}, \tag{8}$$

where y_T is the last observed actual value, and $\hat{\Delta}\tilde{y}_{T+i,j}$ represents the predicted incremental value at fractional step j within time step $T + i$.

5 Experimental Settings

5.1 Datasets

We conduct experiments on the M4 dataset [15], a widely used benchmark for time-series forecasting. The M4 dataset includes 414 hourly time-series from diverse domains such as finance, industry, and others. For simplicity, we use the first 100 time-series from these 414 in this paper.

5.2 Baseline Methods

The baseline method is training a GP method on raw features without decomposing the regression task into smaller steps. In addition to GP, the proposed method is also compared against several widely used machine learning algorithms: ElasticNet, Decision Trees (DT), Random Forests (RF), XGBoost, and Support Vector Regression (SVR). We also include ODEFormer [5] for comparison, as it represents a state-of-the-art deep learning approach for system identification. To ensure a fair comparison, all baseline algorithms, except ODEFormer, use lag features and a differencing mechanism consistent with those employed in the GP method. In this setup, ElasticNet can be viewed as a baseline ARIMA model augmented with L1 and L2 regularization. In contrast, ODEFormer is specifically designed to model system dynamics directly without relying on lagged features, avoiding the need to transform data into a tabular format with time-lagged features. Consequently, training R^2 scores are not reported for ODEFormer, and only the test MAPE is provided.

5.3 Evaluation Protocol

For evaluation, the training and test splits provided by M4 [15] are used. The samples are not randomly shuffled, as it is essential to maintain temporal order to avoid using future data to predict past values. To simulate a scenario where the underlying system changing interval is more fine-grained than the sampled interval, data are subsampled by selecting the first data point at each 6-hour interval from the hourly-level data. The six-hour interval is a commonly used setting in both medical [16] and physics domains [7]. In the standard M4 benchmark, the task is to predict the next 48 h. However, after subsampling, this prediction horizon becomes the next 8 data points. To ensure robustness, all experiments are repeated 30 times with different random seeds. A Wilcoxon signed-rank test [23] with a significance level of 0.05 is employed to compare the performance of the proposed method against the baseline.

Table 2. Parameter settings for GP.

Parameter	Value
Maximal Population Size	200
Number of Generations	100
Crossover and Mutation Rates	0.9 and 0.1
Initial Tree Depth	0–3
Maximum Tree Depth	10
Number of Trees	5
Elitism (Number of Individuals)	1
Functions	$+$, $-$, *, AQ, Square, Log, Sqrt, Max, Min, Sin, Cos, Abs, Negative

Table 3. Statistical comparison of training R^2 scores across 100 datasets. (Symbols $+/\sim/$- indicate the method in the row performs significantly better, similarly, or worse than the method in the column).

	GP	ElasticNet	DT	RF	XGB	SVR
Micro-GP	73(+)/25(∼)/2(-)	100(+)/0(∼)/0(-)	0(+)/0(∼)/100(-)	0(+)/0(∼)/100(-)	0(+)/0(∼)/100(-)	100(+)/0(∼)/0(-)
GP	–	100(+)/0(∼)/0(-)	0(+)/0(∼)/100(-)	0(+)/0(∼)/100(-)	0(+)/0(∼)/100(-)	100(+)/0(∼)/0(-)
ElasticNet	–	–	0(+)/0(∼)/100(-)	0(+)/0(∼)/100(-)	0(+)/0(∼)/100(-)	100(+)/0(∼)/0(-)
DT	–	–	–	100(+)/0(∼)/0(-)	100(+)/0(∼)/0(-)	100(+)/0(∼)/0(-)
RF	–	–	–	–	0(+)/0(∼)/100(-)	100(+)/0(∼)/0(-)
XGB	–	–	–	–	–	100(+)/0(∼)/0(-)

5.4 Parameter Settings

The parameter settings, shown in Table 2, are common for evolutionary feature construction [27]. To handle division safely, we use the analytical quotient (AQ) function [20], defined as $\mathrm{AQ}(x, y) = \frac{x}{\sqrt{y^2+1}}$, which avoids division by zero. Similarly, the logarithmic function is defined as $\log(1 + |x|)$ to prevent undefined values from negative inputs. To reduce runtime, we implement early stopping: if the fitness does not improve over 10 generations, the evolutionary process is terminated. For the ARIMA model, the lag parameter is set to 4, corresponding to a 24-h cycle given a 6-h sampling interval. The augmentation parameter k is set to 1, meaning that each pair of time steps is augmented with one interpolated point.

6 Experimental Results

6.1 Comparison on Training R^2

To evaluate performance on the training data, we use the R^2 metric to assess the model's ability to predict the value at the next time step $t + 1$ based on the current time step t, with all time steps belong to the training set. The R^2 metric

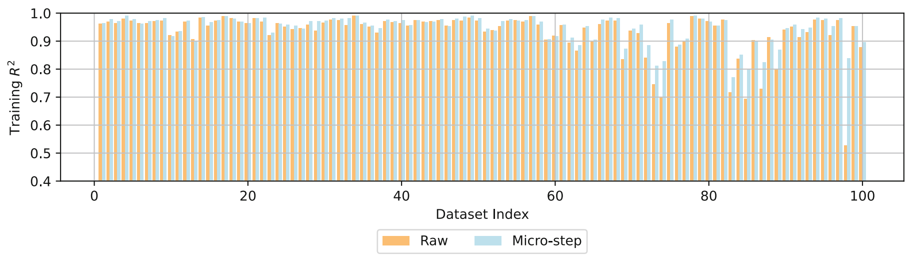


Fig. 3. Training R^2 scores across 100 time-series. Higher values indicate better performance.

is defined as $R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$, where n is the total number of data points, y_i is the actual value at time step i, $\hat{y}_i$ is the predicted value at time step i, and $\bar{y}$ is the mean of the actual values. The median training R^2 scores over the 30 runs across 100 time-series, using both micro-step regression and raw data, are shown in Fig. 3. A statistical comparison between GP and machine learning methods over the 100 time-series is presented in Table 3. The experimental results demonstrate that micro-step regression significantly improves the training R^2 of the evolutionary feature construction method compared with the baseline approach that directly uses the original series. This improvement indicates that data augmentation effectively decomposes the challenging, originally hard-to-fit dataset into easier-to-fit datasets, thus improving training accuracy. When comparing Micro-GP's training R^2 with other machine learning algorithms, results indicate that Micro-GP significantly outperforms ElasticNet across all datasets, underscoring the effectiveness of feature construction over a purely linear model. In contrast, Micro-GP yields lower training R^2 compared to tree-based models, including DT, RF and XGB. This discrepancy arises because tree-based models can precisely memorize the training points by recursively partitioning the feature space until only one point exists in each region. Consequently, these tree-based models achieve superior training performance.

6.2 Comparison on Test MAPEs

For the test data, the objective is not to predict only the next time step $t+1$ based on the previous time step t. Instead, the goal is to forecast over a horizon that extends beyond the training period. To evaluate model performance over this interval, we use the Mean Absolute Percentage Error (MAPE), which is defined as MAPE $= \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \times 100\%$, where n is the total number of predictions. The median MAPE over 30 runs across 100 time-series is presented in Fig. 4, along with a statistical comparison in Table 4. These results demonstrate that micro-step regression significantly improves the generalization performance of the evolutionary feature construction method compared to the standard approach applied to the original series. This improvement suggests that micro-step prediction is an effective strategy for enhancing time-series regression

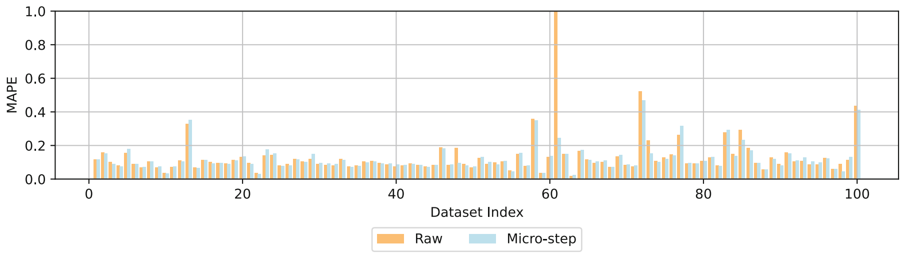
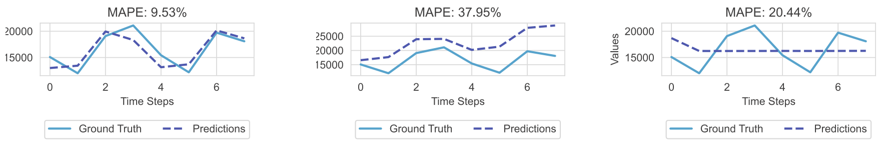


Fig. 4. Test MAPE across 100 series. MAPE values are capped at 1, with lower values indicating better performance. Accumulation of errors in time-series forecasting results in high MAPE for certain datasets.

Table 4. Statistical comparison of test MAPE across 100 datasets. (Symbols +/∼/- indicate the method in the row performs significantly better, similarly, or worse than the method in the column).

	GP	ElasticNet	DT	RF	XGB	SVR	ODEFormer
Micro-GP	21(+)/74(∼)/5(-)	32(+)/44(∼)/24(-)	51(+)/22(∼)/27(-)	35(+)/27(∼)/38(-)	38(+)/25(∼)/37(-)	96(+)/3(∼)/1(-)	98(+)/1(∼)/1(-)
GP	—	17(+)/48(∼)/35(-)	30(+)/48(∼)/22(-)	20(+)/43(∼)/37(-)	25(+)/34(∼)/41(-)	74(+)/23(∼)/3(-)	76(+)/22(∼)/2(-)
ElasticNet	—	—	52(+)/18(∼)/30(-)	48(+)/6(∼)/46(-)	52(+)/0(∼)/48(-)	99(+)/0(∼)/1(-)	100(+)/0(∼)/0(-)
DT	—	—	—	22(+)/16(∼)/62(-)	31(+)/12(∼)/57(-)	93(+)/2(∼)/5(-)	93(+)/2(∼)/5(-)
RF	—	—	—	—	48(+)/8(∼)/44(-)	96(+)/1(∼)/3(-)	98(+)/1(∼)/1(-)
XGB	—	—	—	—	—	94(+)/0(∼)/6(-)	96(+)/0(∼)/4(-)
SVR	—	—	—	—	—	—	53(+)/1(∼)/46(-)

accuracy, particularly when the underlying system exhibits fine-grained dynamics. Comparing the test MAPE results of Micro-GP with other machine learning algorithms, the performance varies across datasets. Specifically, Micro-GP significantly outperforms ElasticNet on 32 datasets but performs worse on 24 datasets. This outcome highlights that while evolutionary feature construction can enhance generalization, its effectiveness depends on the specific dataset and context. A similar pattern emerges in comparisons between Micro-GP and other algorithms: results are mixed across the 100 datasets, indicating that both GP-based and non-GP methods are suited to different types of datasets, and there is no universal algorithm that performs optimally across all datasets. When comparing the proposed method to pretraining-based modeling approaches, the proposed method demonstrates significantly better performance than ODEFormer. ODEFormer utilizes a Transformer-based architecture designed to capture relationships between data and dynamic equations derived from synthetic dynamic systems. However, this pretraining on synthetic data presents challenges when applied to real-world data, such as that in the M4 benchmark, which may exhibit patterns and complexities not present in the synthetic systems. As a result, ODEFormer struggles to generalize effectively to these real-world scenarios, leading to poor performance in the M4 benchmark.



(a) Micro-Step Regression (b) Raw Regression (c) ODEFormer [5]

Fig. 5. Prediction Results for Time-Series 21 in M4 Using Different Regression Techniques.

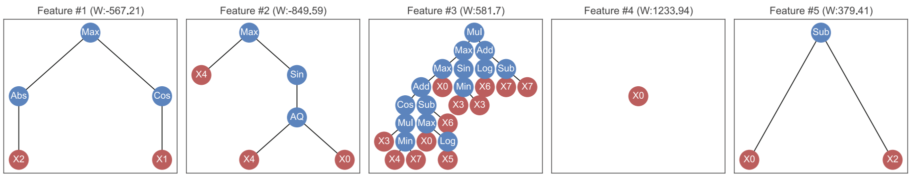


Fig. 6. Examples of Features constructed by GP based on Micro-Step Time-Series Regression.

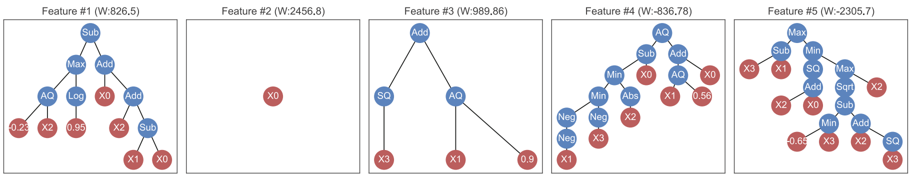


Fig. 7. Examples of Features constructed by GP based on Raw Time-Series Regression.

6.3 Further Analysis

Prediction Plot. To investigate the advantages of Micro-GP for time-series forecasting, we compare prediction results on time-series 21 in M4, shown in Fig. 5(a) and Fig. 5(b). These figures represent the prediction outcomes for models trained with Micro-Step Time-Series Regression and Raw Time-Series Regression, respectively. The model trained on micro-steps demonstrates strong generalization, achieving accurate multi-step predictions on unseen data. In contrast, while the model trained on raw features performs reasonably at the initial prediction step, its accuracy deteriorates significantly from the second step onward, indicating poorer generalization. When compared with ODEFormer, as shown in Fig. 5(c), ODEFormer fails to capture the underlying patterns in the data, highlighting the advantage of the GP-based method for modeling the dynamics of real-world systems.

Example Evolved Programs. To further examine why micro-step learning outperforms raw time-series regression, we analyze the coefficients of the constructed features in Fig. 6 and Fig. 7. Notably, there are no excessively large linear coefficients, suggesting that the observed poor generalization is not due to

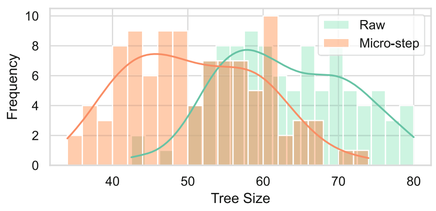


Fig. 8. Distribution of median model sizes across 30 runs on 100 time-series.

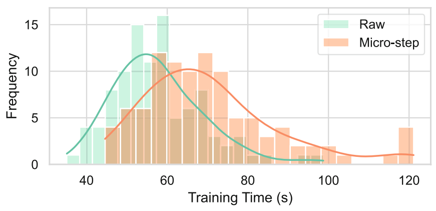


Fig. 9. Distribution of median training time across 30 runs on 100 time-series.

ill-conditioned coefficients. The primary distinction between models lies in the complexity of the constructed features. Specifically, features constructed from raw time-series data tend to be more complex than those constructed from micro-step data. This suggests that, to capture dynamics over larger time steps, GP have evolved more intricate features, which may lead to overfitting on the training data. In contrast, when learning on smaller steps, changes are more gradual, enabling GP to evolve simpler, more interpretable features that fit the data effectively. Consequently, these simpler features are less prone to overfitting, enhancing the model's generalization on unseen data.

6.4 Comparisons on Model Size and Training Time

Model Size (Number of Nodes). As analyzed, decomposing the problem into smaller steps can simplify the time-series regression task, leading to better training and generalization performance. A comparison of model sizes between these two strategies is presented in Fig. 8. As shown, the evolved trees on micro-step data are smaller than those on raw data, indicating that learning GP features on micro-step data is more efficient and results in more compact models. By contrast, learning directly from raw data requires more complex GP trees to capture changes over larger time steps, which significantly increases the search space and complicates the learning process. Therefore, from a search efficiency perspective, the micro-step time-series regression technique offers more advantageous.

Training Time. Regarding training time, a comparison using micro-step data versus raw data is presented in Fig. 9. The results indicate that employing micro-step data does not significantly increase training time, even with the doubling of training data due to augmentation. This is likely due to the reduction in model complexity achieved through micro-step regression. While the amount of training data increases, the less complex GP trees resulting from micro-step regression offset the additional computational complexity, leading to only a modest increase in training time.

7 Conclusions

In this paper, we identified that large step sizes adversely affect performance in dynamic system identification tasks, even when the ground truth ODEs are known. To mitigate this issue, we proposed a micro-step time-series regression technique that augments training data with interpolated points between observed time steps. This approach allows the model to predict smaller, incremental changes rather than large, discrete jumps, effectively decomposing the regression task into simpler sub-problems. This decomposition allows GP to evolve simpler yet effective features that capture meaningful patterns in the data, leading to improved accuracy in time-series regression and the evolution of more compact models. Experimental results demonstrate that the proposed method significantly enhances generalization performance compared to models trained directly on raw time-series data, achieving superior results in over 21 of the 100 evaluated series. Moreover, the method outperformed pre-trained dynamic system modeling approaches, such as ODEFormer, highlighting the advantages of evolutionary feature construction for time-series regression. For future work, a promising direction is to develop a method for automatically determining the optimal step size in micro-step time-series regression, which could enhance the adaptability and applicability of the proposed method to real-world problems.

References

1. Box, G.E., Jenkins, G.M., Reinsel, G.C., Ljung, G.M.: Time Series Analysis: Forecasting and Control. Wiley (2015)
2. Brown, R.: Statistical Forecasting for Inventory Control, vol. 2, pp. 443–473. McGraw-Hill (1959)
3. Chen, T.Q., Rubanova, Y., Bettencourt, J., Duvenaud, D.: Neural ordinary differential equations. In: Bengio, S., Wallach, H.M., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3–8 December 2018, Montréal, Canada, pp. 6572–6583 (2018)
4. Cramer, S., Kampouridis, M., Freitas, A.A., Alexandridis, A.: Stochastic model genetic programming: deriving pricing equations for rainfall weather derivatives. Swarm Evol. Comput. **46**, 184–200 (2019)
5. d'Ascoli, S., Becker, S., Schwaller, P., Mathis, A., Kilbertus, N.: ODEFormer: symbolic regression of dynamical systems with transformers. In: The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, 7–11 May 2024. OpenReview.net (2024)
6. Fanjiang, Y.Y., Syu, Y., Huang, W.L.: Time series QoS forecasting for web services using multi-predictor-based genetic programming. IEEE Trans. Serv. Comput. **15**(3), 1423–1435 (2020)

7. Galtieri, J., Reno, M.J.: Intelligent sampling of periods for reduced computational time of time series analysis of PV impacts on the distribution system. In: 2017 IEEE 44th Photovoltaic Specialist Conference (PVSC), pp. 2975–2980. IEEE (2017)
8. Gaucel, S., Keijzer, M., Lutton, E., Tonda, A.: Learning dynamical systems using standard symbolic regression. In: Lecture Notes in Computer Science, pp. 25–36. Springer, Heidelberg (2014)
9. Iba, H., deGaris, H., Sato, T.: A numerical approach to genetic programming for system identification. Evol. Comput. **3**(4), 417–452 (1995)
10. Kronberger, G., Kammerer, L., Kommenda, M.: Identification of dynamical systems using symbolic regression. In: Computer Aided Systems Theory–EUROCAST 2019: 17th International Conference, Las Palmas de Gran Canaria, Spain, 17–22 February 2019, Revised Selected Papers, Part I 17, pp. 370–377. Springer (2020)
11. Kuranga, C., Pillay, N.: Genetic programming-based regression for temporal data. Genet. Program Evolvable Mach. **22**(3), 297–324 (2021). https://doi.org/10.1007/s10710-021-09404-w
12. La Cava, W., Helmuth, T., Spector, L., Moore, J.H.: A probabilistic and multi-objective analysis of lexicase selection and ε-lexicase selection. Evol. Comput. **27**(3), 377–402 (2019)
13. La Cava, W., Moore, J.H.: Learning feature spaces for regression with genetic programming. Genet. Program Evolvable Mach. **21**(3), 433–467 (2020). https://doi.org/10.1007/s10710-020-09383-4
14. La Cava, W., Silva, S., Danai, K., Spector, L., Vanneschi, L., Moore, J.H.: Multidimensional genetic programming for multiclass classification. Swarm Evol. Comput. **44**, 260–272 (2019)
15. Makridakis, S., Spiliotis, E., Assimakopoulos, V.: The M4 competition: results, findings, conclusion and way forward. Int. J. Forecast. **34**(4), 802–808 (2018)
16. Matowe, L.K., Leister, C.A., Crivera, C., Korth-Bradley, J.M.: Interrupted time series analysis in clinical research. Ann. Pharmacother. **37**(7–8), 1110–1116 (2003)
17. Mehr, A.D., Gandomi, A.H.: MSGP-LASSO: an improved multi-stage genetic programming model for streamflow prediction. Inf. Sci. **561**, 181–195 (2021)
18. Murari, A., Peluso, E., Spolladore, L., Rossi, R., Gelfusa, M.: Upgrades of genetic programming for data-driven modeling of time series. Evol. Comput. **31**(4), 401–432 (2023)
19. Murphy, J., Desell, T.: Minimizing the EXA-GP graph-based genetic programming algorithm for interpretable time series forecasting. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion, pp. 1686–1690 (2024)
20. Ni, J., Drieberg, R.H., Rockett, P.I.: The use of an analytic quotient operator in genetic programming. IEEE Trans. Evol. Comput. **17**(1), 146–152 (2012)
21. Raissi, M., Perdikaris, P., Karniadakis, G.: Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. J. Comput. Phys. **378**, 686–707 (2019)
22. Schmidt, M., Lipson, H.: Distilling free-form natural laws from experimental data. Science **324**(5923), 81–85 (2009). https://doi.org/10.1126/science.1165893
23. Virgolin, M., Alderliesten, T., Witteveen, C., Bosman, P.A.: Improving model-based genetic programming for symbolic regression of small expressions. Evol. Comput. **29**(2), 211–237 (2021)
24. Winkler, S., Affenzeller, M., Wagner, S., Kronberger, G., Kommenda, M.: Using genetic programming in nonlinear model identification. In: Identification for Automotive Systems, pp. 89–109 (2012)

25. Zhang, H., Chen, Q., Xue, B., Banzhaf, W., Zhang, M.: Modular multi-tree genetic programming for evolutionary feature construction for regression. IEEE Trans. Evol. Comput. (2023)
26. Zhang, H., Chen, Q., Xue, B., Banzhaf, W., Zhang, M.: A semantic-based hoist mutation operator for evolutionary feature construction in regression. IEEE Trans. Evol. Comput. (2023)
27. Zhang, H., Zhou, A., Chen, Q., Xue, B., Zhang, M.: SR-forest: a genetic programming based heterogeneous ensemble learning method. IEEE Trans. Evol. Comput. (2023)