

Article

A Neural Input Optimization Framework for Structure Search in Dynamical Systems

Ricardo A. Calix *, Ajaykumar Rejith and Tae-Hoon Kim

Department of Computer Information Technology, Purdue University Northwest, Hammond, IN 46323, USA; arejoth@pnw.edu (A.R.); tae.hoon.kim@pnw.edu (T.-H.K.)

* Correspondence: rcalix@pnw.edu; Tel.: +1-(219)-989-2013

Abstract

The world consists of many dynamical and chaotic systems whose underlying functions can be highly nonlinear. Traditional engineering and scientific methods emphasize simplifying problems, sometimes to the point that valuable information may be lost. Instead, understanding of these systems may require analysis of the underlying dynamics. In the past, the technique of discovering and analyzing strange attractors has yielded some success. However, much work is still needed. Machine learning (ML)-based models such as deep neural networks which are based on nonlinear functions may provide a great set of techniques to help in the discovery of structure in dynamical systems. In this paper, we propose an ML-based approach to search for structure in dynamical systems using Neural Input Optimization (NIO). We discuss our methodology and present promising results. Quantitative results help to validate our proposed NIO methodology for structure discovery.

Keywords: dynamical systems; Neural Input Optimization; chaos; machine learning; strange attractors

1. Introduction

The world consists of many dynamical and chaotic systems. Weather forecasting being a classical example of this. The standard engineering and scientific methods emphasize simplifying problems by making assumptions. Sometimes to the point that valuable information may be lost. Instead, modeling dynamical systems may require studying as many issues as possible related to the problem. Considering many chaotic aspects of a problem can be very challenging because the behavior of underlying functions may be nonlinear. Chaos theory and the study of strange attractors can sometimes provide valuable insights when analyzing dynamical systems. However, much work is still needed. Machine learning (ML)-based models such as deep neural networks which are based on nonlinear functions may provide a great set of techniques to help in the discovery of structure in dynamical systems. In this paper, we propose an ML-based approach to search for structure in dynamical system using Neural Input Optimization (NIO). We discuss our methodology and present promising results.

Dynamical systems and chaos modeling have a wide application to many very important problems in our world today. These problems broadly range from species evolution, disease spread, fluid dynamics, healthcare applications such as medical arrhythmia diagnosis, to airplane design, steel blast furnace modeling, and many more.

A lot of progress has been made since the mid-20th century thanks to Edward Lorenz [1], Mandelbrot [2], Robert Shaw [3], and many more. One key contribution



Academic Editors: António Lopes,
Yeliz Karaca and Sergio Adriani
David

Received: 30 June 2026

Revised: 27 July 2026

Accepted: 29 July 2026

Published: 31 July 2026

Copyright: © 2026 by the authors.
Licensee MDPI, Basel, Switzerland.
This article is an open access article
distributed under the terms and
conditions of the [Creative Commons
Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

in dynamical systems research was the introduction of the strange attractor to discover structure in chaotic dynamical system. Lorenz [1] had one of the biggest contributions in this respect with his famous work. A strange attractor is a way of representing data from a dynamical system in a state space as the behavior of the system evolves over time. This new way of looking at the evolving dynamics of the system can help us learn new characteristics about the system. This is best exemplified by the Lorenz strange attractor (Figure 1, below). While methods to discover strange attractors, or just structure in dynamical systems, have been proposed, there are still many challenges that need to be addressed. Many existing approaches still remain too domain-specific and are computationally very expensive, motivating the development of more data-driven methodologies such as NIO. Our proposed NIO methodology addresses these limitations by providing an input optimization-based general framework for structure search without requiring highly domain-specific formulations, in an efficient manner.

In this work, we propose that machine learning models can now be leveraged to help in the search and characterization of dynamical systems for information and structure search in chaotic systems. We propose a methodology for structure search in complex or dynamical systems based on Neural Input Optimization (NIO) [4]. To illustrate our methodology, we use the well-known Lorenz equations to present the proposed methodology. We evolve a dynamical system via the Lorenz equations from initial conditions and compare the produced trajectories via Neural Input Optimization.

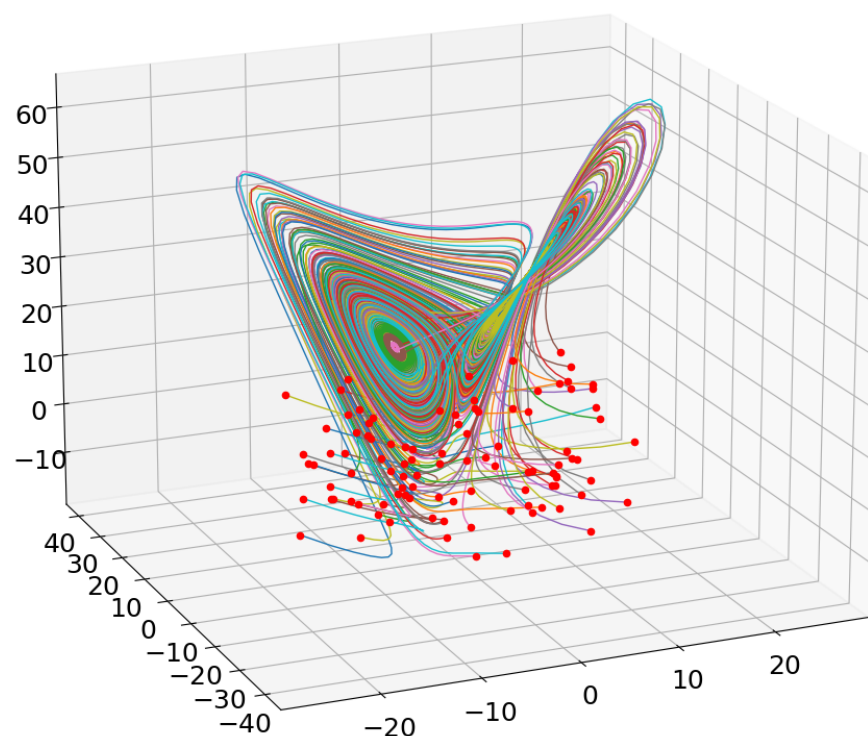


Figure 1. Lorenz strange attractor.

Neural Input Optimization (NIO) represents a class of ML algorithms that focus on optimizing the inputs to a model instead of the weights of the model itself. We use NIO to discover initial conditions that result in trajectories that may deviate considerably if perturbed by a small delta value. For contrast, we compare these to a new set of initial conditions discovered by NIO where the optimization objective is to minimize the trajectory divergence when adding the same delta perturbation. With these two sets of discovered initial conditions, we proceed to characterize the two sets for structure search using descriptive statistics, simple projections, and kernel density maps. This provides a

powerful visualization approach that allows the researcher to optimize, visualize, modify parameters, and iterate. This iterative process, for structure search, represents one of the core features of our proposed methodology. Finally, we compare the two sets via trajectory divergence metrics.

We propose that Neural Input Optimization provides a useful framework for exploring the structure of chaotic dynamical systems by systematically searching the space of initial conditions. Inspired by Robert Shaw's seminal work on the dripping faucet as a dynamical system [5], we envision this approach as a machine learning tool for scientific exploration, analogous in spirit to how an oscilloscope or microscope enables the exploration of physical phenomena. Experimental results are presented and discussed. The remainder of this paper is organized as follows: Section 2 addresses the related literature review, Section 3 introduces the methodology, Section 4 discusses the results, and Section 5 provides some conclusions and future work.

2. Literature Review

In this section we discuss previous work relevant to this paper. First, we cover the literature on dynamical systems, and then we focus on the literature on related machine learning approaches.

2.1. Dynamical Systems

Dynamical systems and chaos modeling have a wide application to many very important problems in our world today. These problems broadly range from weather forecasting, species evolution, disease spread, fluid dynamics, healthcare application such as tumor growth analysis, to airplane design, steel blast furnace modeling, and many more [6]. Dynamical Systems consist of modeling frameworks that help to represent complex systems. In general, they can be thought of as evolving systems over time. Given a set of equations that represent an underlying physical behavior, a dynamical system can be evolved from initial conditions through these equations. In the use case used in this work we consider the evolution of fluids in the atmosphere (i.e., the Lorenz equations). We start with initial conditions that go into the equations to produce the next state in the time-dependent trajectory. From there, each new predicted state becomes an input to the set of equations which are used to predict the next state. As such, a model can be evolved in this fashion over a time interval (T).

Many have contributed to the field of dynamical systems [6]. Of note are M. Feigenbaum [7], B. Mandelbrot [2], Edward Lorenz [1], and R. Shaw [3]. Feigenbaum did important work on Universality and discovered that chaotic systems had common structure and patterns such as in ratios that persisted across chaotic systems [7]. Lorenz did very important work on discovering strange attractors in chaotic systems related to weather systems and this work was very influential [1]. A strange attractor is usually a graphical structure found in what was believed to be a 100 percent chaotic system. Usually, in a state space or phase space, which for visualization purposes has three dimensions to represent states of a dynamical system. The key idea is that almost any system [3,5] where some of its data, or behavior, may seem chaotic and random can, in fact, have structure over certain features which can then be represented in this state space. The amazing thing is that this behavior can have structure (see Lorenz figure) and the resulting strange attractor can help researchers to study a problem from an entirely different perspective. This also has implications on information theory given that now noise may be broken up and re-classified into random noise and chaos information [3,8].

Works that have used machine learning with dynamical systems include [9–12]. In [11,12] for instance, the authors used ML to improve and accelerate Computational

Fluid Dynamics (CFD) modeling. Machine learning has also been used to discover governing equations directly from dynamic data. Brunton et al. proposed a sparse regression framework that identified nonlinear equations for systems such as the Lorenz attractor. This helped to demonstrate that machine learning can recover underlying physical models rather than just making predictions [9]. Weng et al. investigated “reservoir computing” as an alternative to traditional dynamical equations for modeling chaotic systems such as the Lorenz system. Their results showed that trained models can preserve key dynamic properties. This helped to illustrate how machine learning can represent chaotic behavior when analytical equations may not be available [10].

2.2. Neural Input Optimization

Neural Input Optimization (NIO) represents a class of ML algorithms that focus on optimizing the inputs to a model instead of the weights of the model itself. Generally speaking, NIO can be used as a generative methodology that combines the key aspects of gradient descent methodologies from neural networks with that of traditional constraint-based optimization methods. Constraint-based optimization has long been used in solving engineering and science problems. Essentially, NIO is a type of inverse machine learning approach where, given the desired output of a model, we use stochastic gradient descent to find the corresponding inputs. This approach has traditionally been used in the fields of physics, materials science and chemistry. The classic example of an inverse optimization approach consists of developing new materials. For instance, inverse approaches via optimization can be used to determine what materials and levels of energy are needed to produce a specific alloy with a certain degree of brittleness or some other characteristic. This formulation provides a simple template on how to define a Neural Input Optimization problem. Molesky et al., for instance, describes an example of how to formulate an Inverse problem [13]. In their work, the authors discuss inverse designs related to photonics engineering which involved either gradient-based approaches or genetic algorithms. Inverse modeling has also been used in the design of molecules. In [14], the authors discuss how they used Inverse modeling to produce molecules that meet specific properties from given starting points. In [15], the authors use a frozen neural network model and inversely optimize the binary inputs for automating the discovery of optimal designs in nano-photonics materials. Wang et al. [16] provide a good review on the use of inverse optimization for materials science. Essentially, one advantage of using stochastic gradient descent methods and inverse modeling approaches is that they provide efficient search space frameworks that can reduce the burden when using expensive and difficult simulations [16].

Specifically, Neural Input Optimization can therefore be defined as a technique in Inverse modeling where the goal is to use stochastic gradient descent to optimize the values of the latent input vectors. This is especially helpful when a model is already trained and the weights can be frozen. Other works that have used NIO for input optimization include [4,17]. Gatys et al. [17] is a very important work related to Neural Input Optimization. It is a landmark paper referred to commonly as Gatys’s Neural Art Style Transfer. This work uses a similar approach to NIO to optimize input images. Famously, it is used to transfer the painting styles from famous art works to common everyday images. Recently, NIO was applied for cyber security and secure token generation [4]. In that work, NIO was used to generate discrete values, such as characters, rather than continuous regression values.

3. Methodology

In this section, the data, modeling methodologies, and performance metrics are presented and discussed. The proposed methodology focuses on searching for structure in dynamical systems. To illustrate our methodology, we use the well-known Lorenz dynamical system.

ical system and equations. We evolve a dynamical system via the Lorenz equation from initial conditions and compare the produced trajectories via Neural Input Optimization.

Neural Input Optimization (NIO) represents a class of ML algorithms that focus on optimizing the inputs to a model instead of the weights of the model itself. We use NIO to discover initial conditions that result in trajectories that may deviate considerably if perturbed by a small delta value. For contrast, we compare these to a new set of initial conditions discovered by NIO where the optimization objective is to minimize the trajectory divergence when adding the same delta perturbation. With these 2 sets of discovered initial conditions, we proceed to characterize the 2 sets for structure search and discovery using descriptive statistics, simple projections, and kernel density maps. This provides a powerful visualization approach that allows the researcher to optimize, visualize, modify parameters, and iterate.

3.1. Neural Input Optimization

As described in Calix et al. [4], the basic concept behind NIO is very interesting and simple. You train a model and freeze its weights. You then create a random input that requires gradient optimization. Iteratively, you feed the random input to the model to produce an output. The output is compared to the constraints via the loss function. This is your error. You then backpropagate the error back to the input. Because the input is differentiable, the input values (weights) are updated. With enough training, the input values will update to the right input that produces the desired output. Algorithm 1 (below) describes the high-level overall methodology.

Algorithm 1: Neural Input Optimization (NIO) for Sensitive Initial Condition Search

Input: Optimization steps T , perturbation magnitude ϵ
Output: Optimized initial condition $\hat{x}_0$

- 1 Initialize candidate initial condition $\mathbf{x}_0$ with random values;
- 2 **for** $t \leftarrow 1$ **to** T **do**
- 3 Generate perturbed initial condition: $\mathbf{x}'_0 \leftarrow \mathbf{x}_0 + \epsilon$;
- 4 Simulate trajectory $\mathcal{T}_1$ from $\mathbf{x}_0$;
- 5 Simulate trajectory $\mathcal{T}_2$ from $\mathbf{x}'_0$;
- 6 Compute trajectory separation:

$$D = \|\mathcal{T}_2 - \mathcal{T}_1\|$$
- 7 Compute optimization objective:

$$\mathcal{L} = -D$$
- 8 Backpropagate $\mathcal{L}$ and update $\mathbf{x}_0$ using gradient descent;
- 9 **end**
- 10 Return optimized initial condition $\hat{x}_0$;

The key idea is that a latent input vector $\mathbf{z}$ is optimized via gradient descent to generate inputs $\mathbf{x}$ such that $f_\theta(\mathbf{x})$ satisfies the user-defined output constraints. NIO needs a differentiable ML pipeline which can consist of a complicated model with many layers and weights such as a large language model (LLM), or it can be as simple as the $\mathbf{z}$ latent vector going through some ReLU and softmax activation functions. In fact, no model is necessarily needed as the gradient optimization and backpropagation only affect the inputs themselves.

Formally, we can define the idea of backpropagating to the latent “z” input values as follows (Equation (1)). We initialize $\mathbf{z}$ randomly and iteratively update it via gradient descent:

$$\mathbf{z} \leftarrow \mathbf{z} - \eta \nabla_{\mathbf{z}} \mathcal{L}_{\text{total}} \quad (1)$$

The $\mathbf{z}$ value can be used directly as the model input $\mathbf{x}$, or it can serve as the input to another function that computes $\mathbf{x}$. The latter approach can be useful, for example, when enforcing input constraints or clamp values.

In the present work, the optimized latent variable $\mathbf{z}$ corresponds directly to the Lorenz initial condition $\mathbf{x}_0$. Therefore, Equation (1) is implemented by updating the initial condition through gradient descent while propagating gradients through the differentiable Lorenz simulator.

3.2. Initial Conditions with Neural Input Optimization for Trajectory Divergence

In this section, we formally describe the main optimization objective used in our proposed methodology. Let x_0 denote an initial condition of the Lorenz system and let ϵ be a small perturbation. Concretely, we then define the optimization objective as follows:

$$S(x_0) = \sum_{t=0}^T \|T(x_0 + \epsilon, t) - T(x_0, t)\| \quad (2)$$

where

- $T(\cdot, t)$ denotes the Lorenz simulator evaluated at timestep t ;
- x_0 is the original initial condition;
- $x_0 + \epsilon$ is a perturbed initial condition;
- $\|\cdot\|$ denotes the squared Euclidean distance or Norm between trajectory states;
- Both MSE and the Euclidean norm were evaluated; the Euclidean norm was used for all results presented in this manuscript.

To explore sensitive regions of the state space, Neural Input Optimization solves

$$\max_{x_0} S(x_0) \quad (3)$$

thereby discovering initial conditions whose trajectories diverge the most under small perturbations. This essentially performs a maximization of the loss function. For comparison, we repeat the exercise for the minimization case. After both optimizations (Max and Min), we find 2 sets of initial conditions. One set by maximizing the divergence of the trajectories and one by minimizing the divergence of the trajectories. We refer to these 2 sets as the Min set and the Max set. Together, these two sets define the extremes of the search space and can help characterize the behavior of the dynamical system.

3.3. Wing Selection Objective

In addition to searching for trajectory divergent initial conditions, Neural Input Optimization (NIO) was used to identify initial states that generate trajectories associated with a specific lobe of the Lorenz attractor. Let $x(t)$ denote the first state variable of the Lorenz system. To encourage trajectories to remain on the desired lobe, the following loss function was employed:

$$L = \frac{1}{TN} \sum_{t=1}^T \sum_{i=1}^N \max(0, -s x_i(t)). \quad (4)$$

where $s \in \{-1, +1\}$ specifies the target lobe. When $s = +1$, trajectories with negative values of $x(t)$ are penalized, encouraging trajectories to remain on the right lobe. Con-

versely, when $s = -1$, trajectories with positive values of $x(t)$ are penalized, encouraging trajectories to remain on the left lobe. The max function used is simply ReLU and the 2 summation symbols divided by TN simply imply a mean calculation.

3.4. Projection and Density Maps

Once the 2 sets of initial conditions have been obtained, the next step is to calculate descriptive statistics, and plot simple scatter, projection, and density maps. This step helps for visualization purposes and as an iterative feedback that the researcher can look at to adjust the parameters of the NIO search.

Each NIO solution corresponds to a point in the three-dimensional Lorenz state space:

$$p_i = (x_i, y_i, z_i)$$

To visualize the distribution of sensitive and insensitive states, we project the three-dimensional state space onto two-dimensional planes.

The formulation of the XY projection is as follows:

$$\Pi_{xy}(x, y, z) = (x, y)$$

The formulation of the XZ projection is as follows:

$$\Pi_{xz}(x, y, z) = (x, z)$$

The formulation of the YZ projection is as follows:

$$\Pi_{yz}(x, y, z) = (y, z)$$

In practice, the projection plots are obtained by simply discarding one coordinate:

- XY projection removes z ;
- XZ projection removes y ;
- YZ projection removes x .

Although mathematically simple, these projections revealed structures that were difficult to observe in the full three-dimensional state space. The projection plots provide a qualitative view of the discovered states. To quantify where NIO may prefer to place sensitive initial conditions, we estimate probability densities for both the Max and Min sets of initial conditions using Kernel Density Estimation (KDE).

Suppose the Max set contains

$$(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)$$

The density estimate is

$$\rho_{\max}(x, y) = \frac{1}{N} \sum_{i=1}^N K_h((x, y) - (x_i, y_i))$$

where K_h is a Gaussian kernel. Notice that the Gaussian kernel looks similar to the standard Gaussian function but is modified to operate on distances rather than deviations from a mean:

$$K_h(\mathbf{r}) = \frac{1}{2\pi h^2} \exp\left(-\frac{|\mathbf{r}|^2}{2h^2}\right)$$

Similarly, for the Min points:

$$\rho_{\min}(x, y) = \frac{1}{M} \sum_{j=1}^M K_h((x, y) - (u_j, v_j))$$

Once the density estimates for Max and Min have been calculated, we can proceed to obtain the key visualization by computing the density difference map as follows:

$$D(x, y) = \rho_{\max}(x, y) - \rho_{\min}(x, y)$$

This density difference map can help highlight regions where NIO may preferentially discover sensitive initial conditions and may help highlight the contrast between Max and Min sets. As such, the density difference maps transform a collection of optimized initial conditions into an interpretable sensitivity map of the Lorenz state space.

Rather than simply reporting a list of coordinates, the workflow becomes

NIO → Sensitive States → Projection Plots → Density Maps → Interpretation

3.5. Lorenz Dynamical System, Data, and Features

No external dataset is required in this study, as all data are generated directly through numerical simulation of the Lorenz dynamical system. The Lorenz system consists of three state variables that describe the evolution of an idealized atmospheric convection process [18]. These variables characterize the state of the system at any given time and are used by Neural Input Optimization (NIO) to search for initial conditions that produce highly divergent or highly stable trajectories. The input and output variables used in this study are summarized in Table 1.

Table 1. Lorenz system variables.

Category	Variable	Definition
Input	x_0	Initial convective intensity
Input	y_0	Initial horizontal temperature variation
Input	z_0	Initial vertical temperature variation
Output	$x(t)$	Convective intensity at time t
Output	$y(t)$	Horizontal temperature variation at time t
Output	$z(t)$	Vertical temperature variation at time t

The Lorenz system is governed by three connected nonlinear differential equations [18]. The parameters of the Lorenz system are σ , ρ , and β , which correspond to the Prandtl number, Rayleigh number, and a geometric scaling factor, respectively. In this work, the standard chaotic configuration of the Lorenz system is employed with $\sigma = 10$, $\rho = 28$, and $\beta = 8/3$. This is a common arrangement, as described in [18]. These parameter values produce the well-known butterfly attractor and chaotic behavior used to evaluate the proposed NIO methodology.

The Lorenz system is governed by the following set of connected nonlinear differential equations:

$$\frac{dx}{dt} = \sigma(y - x) \quad (5)$$

$$\frac{dy}{dt} = x(\rho - z) - y \quad (6)$$

$$\frac{dz}{dt} = xy - \beta z \quad (7)$$

where x , y , and z are the system state variables, σ is the Prandtl number, ρ is the Rayleigh number, and β is a geometric scaling factor.

A common way to describe sensitivity to initial conditions is through the Lyapunov exponent, which measures how quickly nearby trajectories separate over time [18]. During the initial phase of chaotic motion, the separation between two trajectories with initial distance ε grows approximately as

$$d(T) \approx \varepsilon e^{\lambda T} \quad (8)$$

where λ is the largest Lyapunov exponent [19]. This exponential growth does not continue indefinitely, however, as the trajectories eventually become distributed over the attractor [20]. This behavior motivates our evaluation across multiple time horizons (300, 500, 900, and 1300 timesteps), allowing us to identify the horizon at which NIO produces the greatest trajectory separation.

3.6. Performance Metrics

Unlike traditional supervised learning problems, there are no annotated target variables (ground truth) available for evaluation. Instead, the objective of Neural Input Optimization (NIO) is to identify initial conditions that maximize or minimize trajectory divergence under small perturbations.

For this study, performance is measured using a trajectory divergence score. This metric quantifies the separation between two trajectories that originate from nearly identical initial conditions. Larger divergence scores indicate greater sensitivity to initial conditions, while smaller scores indicate more stable system behavior. The effectiveness of NIO is evaluated by comparing the divergence scores obtained from the maximization and minimization objectives. Ideally, the maximization objective should produce larger divergence scores than the minimization objective.

All experiments used the standard Lorenz parameters $\sigma = 10$, $\rho = 28$, and $\beta = 8/3$. Additional parameters include: timestep of $dt = 0.01$, perturbation $\epsilon = 0.01$, learning rate of 0.05, 1000 optimized initial conditions, 1000 optimization epochs, initial conditions constrained to $x \in [-20, 20]$, $y \in [-30, 30]$, and $z \in [0, 50]$. Kernel Density Estimation (KDE) was performed with default SciPy `gaussian_kde` function (Scott's Rule).

4. Results

In this section, the results are presented and discussed. In general, performance was good, and the models learned quickly for both the minimization and maximization case. All optimization experiments were completed in less than one hour on a standard CPU workstation. As shown in the following figures, we can notice that the losses are going up for the maximization case and they are going down for the minimization case.

Figure 2 shows the minimization optimization loss values.

Figure 3 shows the maximization optimization loss values over training epochs. In the interest of providing more conclusive results, we repeated all experiments with two different loss objectives. One loss objective looked at the full trajectory, and the other loss objective looked at just the last step in the trajectory. Both loss objectives provided similar results in terms of the subsequent figures. As can be seen in Figure 3, for the maximization case, the last step optimization objective has better convergence than when using the entire trajectory objective.

In the next section we present and discuss some statistical results and the projection and density map plots.

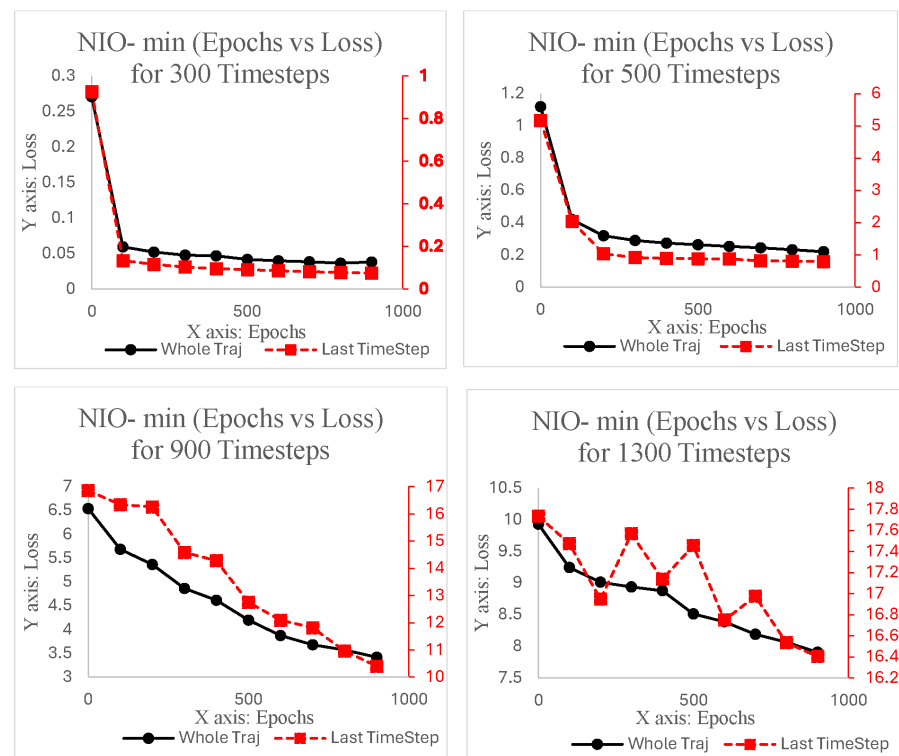


Figure 2. NIO minimization losses.

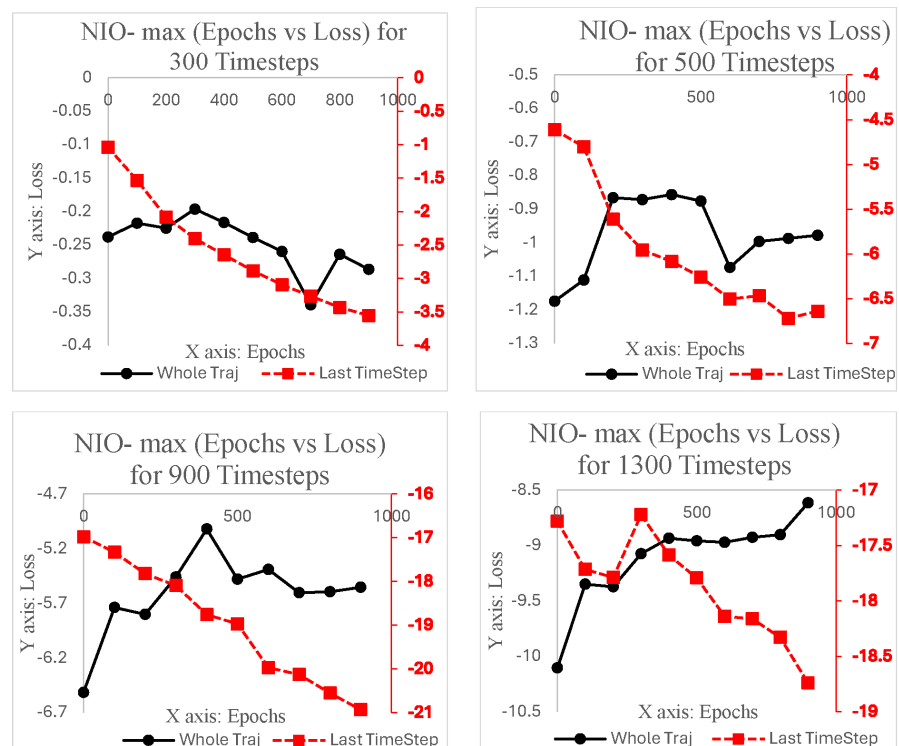


Figure 3. NIO maximization losses.

4.1. Projection and Density Map Plots

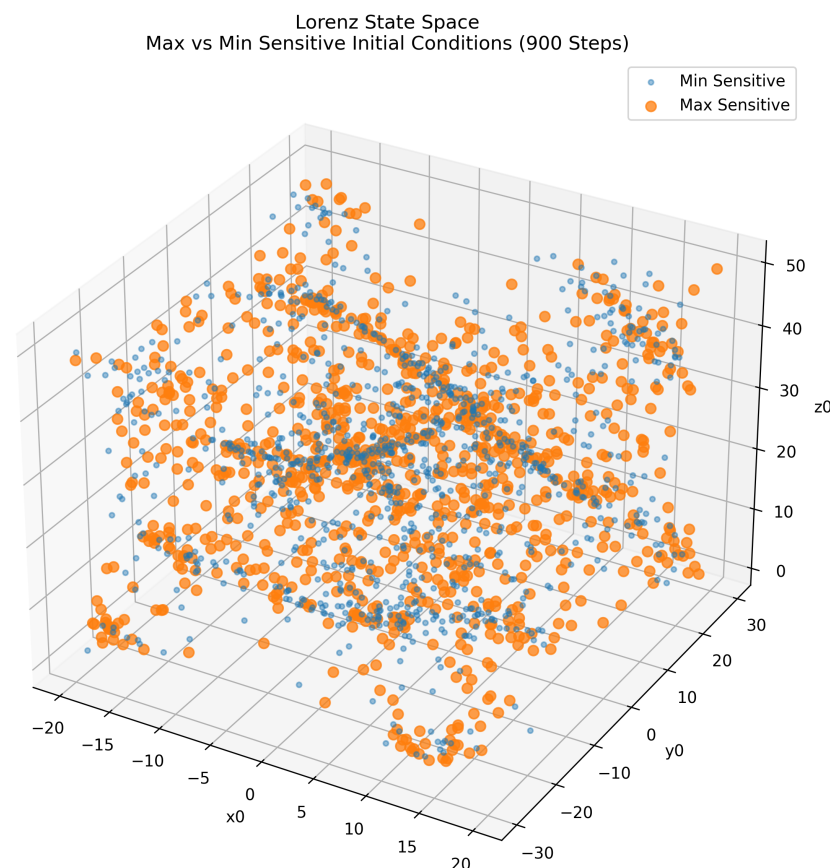
In this section, we focus on descriptive statistics of the obtained initial conditions and also on plotting the data. The descriptive statistics can be seen in the following table (Table 2). Additionally, we did some experiments with initial conditions that were chosen randomly for additional contrast. The descriptive statistics for the random case can also be seen in the table below.

Table 2. Descriptive statistics.

Variable	Mean	St. Dev	Min	Max	Objective
x0	0.10	11.81	−19.77	19.77	Max
y0	−0.09	16.93	−29.45	29.63	Max
z0	24.84	14.28	0.73	49.86	Max
x0	0.09	10.83	−19.81	19.53	Min
y0	0.09	16.13	−28.89	28.43	Min
z0	25.10	13.90	0.80	49.73	Min
x0	−0.04	11.43	−19.87	19.97	Random
y0	−0.93	17.24	−29.83	29.84	Random
z0	24.78	14.42	0.03	49.98	Random

From Table 2 we can observe that descriptive statistics are similar across all three sets, suggesting that summary statistics alone are insufficient. The underlying structure must therefore be explored by further visualizations and by evolving the discovered initial conditions into trajectories. In the previous table, the column “Objective” just indicates if the values were related to a maximization objective, a minimization objective, or if they were random.

Figure 4 shows the 3D State Space contrasting the initial conditions from both the Max and the Min case.

**Figure 4.** 3D state space.

As can be seen in the previous and following figures, underlying structure begins to emerge in the visualizations. These plots become even more useful when used iteratively to visualize the impact of changes to parameters and loss objectives within the proposed framework.

Figure 5 shows the projection plots described in the methodology. These are simply two-dimensional views obtained by projecting the three-dimensional initial conditions onto pairs of variables.

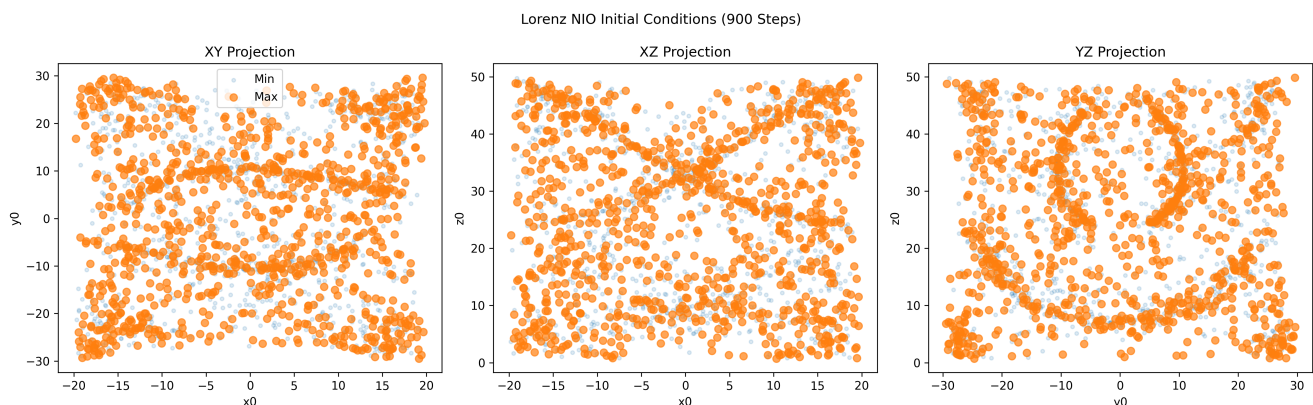


Figure 5. Projection plots.

Figure 6 shows the density difference maps. As described in the methodology, these plots highlight where the Max and Min point sets differ across the state space. Each density difference map is computed from the two-dimensional projections (XY, XZ, and YZ) of the Max and Min initial condition sets.

KDE difference maps are used to compare the distributions of two sets of populations. Values near zero indicate similar population concentrations, while positive and negative values indicate regions where one population is more concentrated than the other.

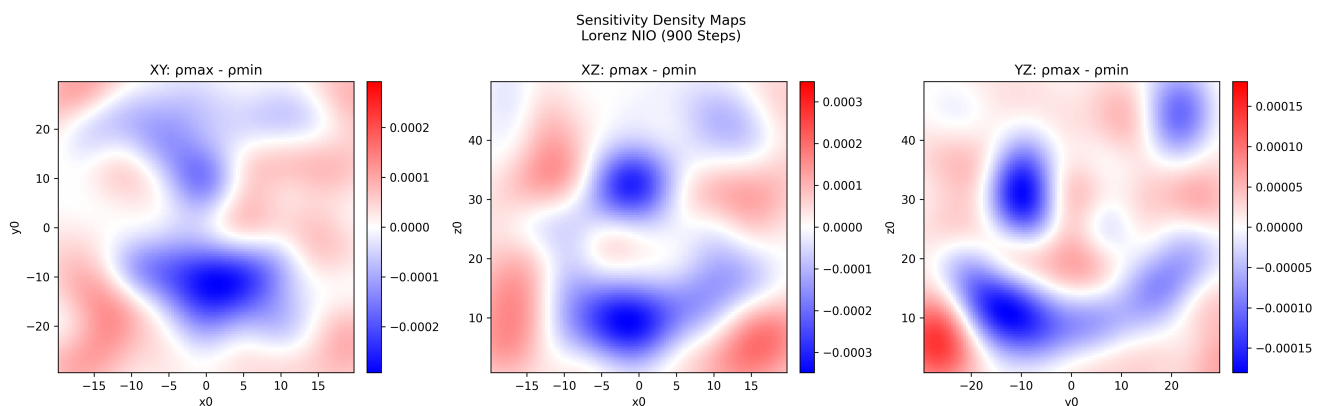


Figure 6. Density map plots.

4.2. Max to Min vs. Max to Random Plots

In this section, we show results of repeating the experiment but using random initial conditions instead of minimized trajectory divergence points for comparison with the Max set points. The scatter plots can be seen in Figure 7.

Of the four scatter plots in Figure 7, we can observe that three plots have some structure while one is completely random. The plots in Figure 7 represent the type of visual insight we hope NIO can discover. As domain experts interpret these graphical insights, they may help direct future research.

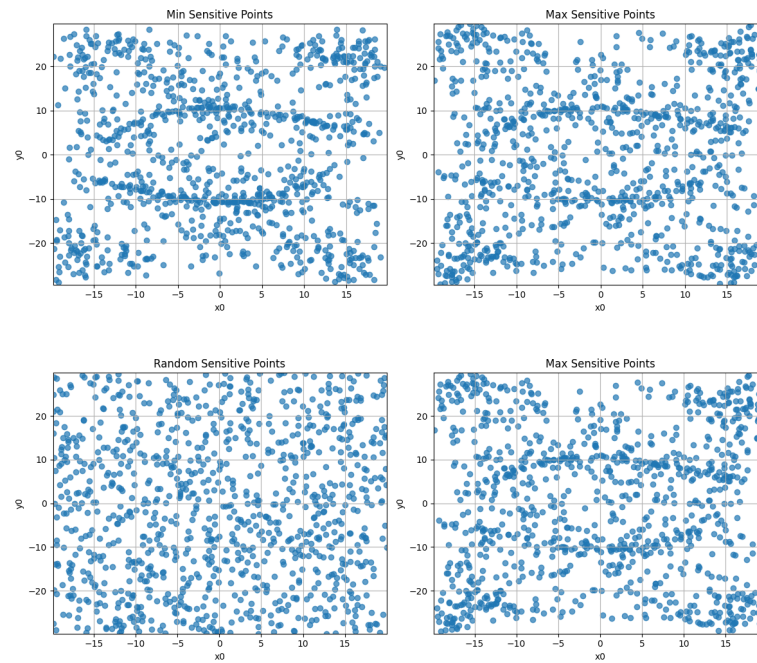


Figure 7. Max to Min vs. Max to random scatter plots.

The same density difference maps can also be used to contrast the Random initial conditions with the discovered Max and Min initial conditions, as shown in Figure 8.

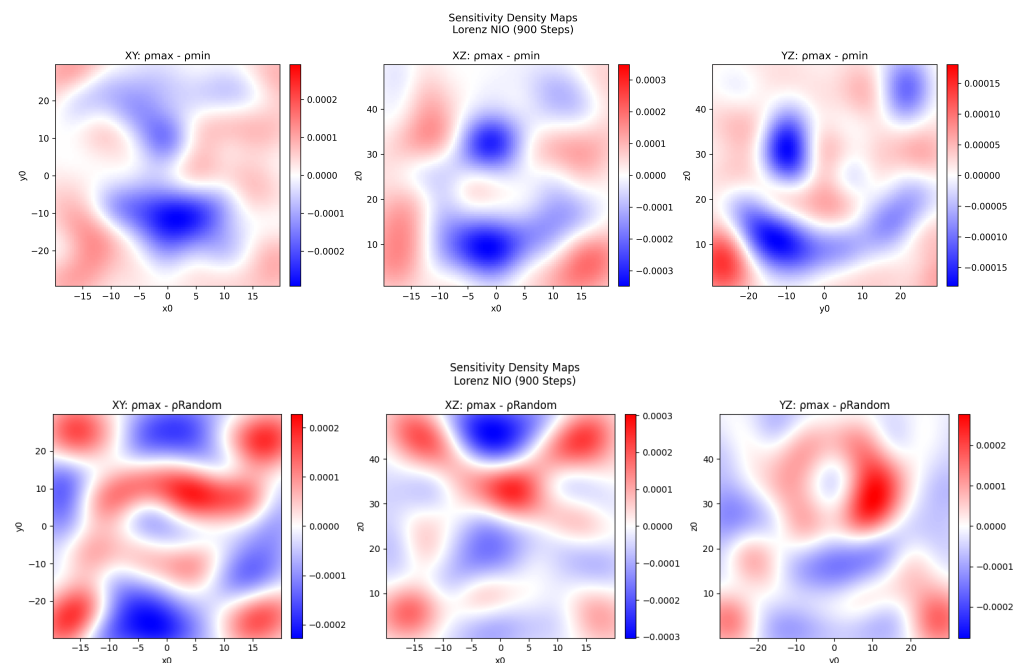


Figure 8. Max to Min to random density maps.

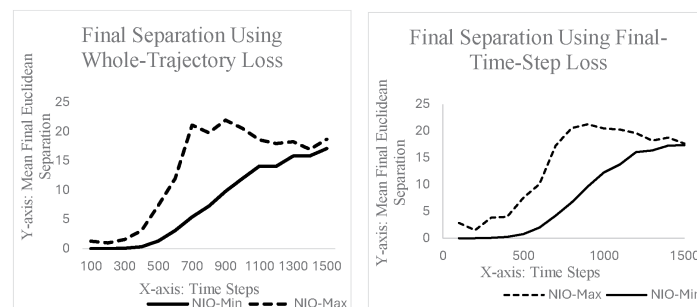
4.3. Performance Metrics and Trajectory Divergence Score

Both the whole-trajectory and last-timestep divergence metrics were studied. Table 3 reports a measure of separation between a trajectory and a perturbed version of itself after optimization. The measure of separation was evaluated in two ways: (1) by averaging the distance over the entire trajectory and (2) by considering only the separation at the final timestep.

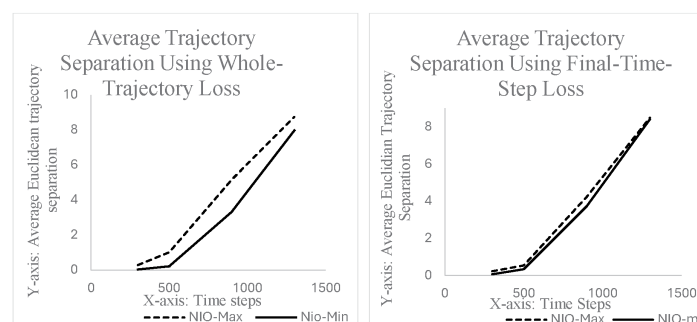
Table 3. Distance separation of trajectories.

Timestep	NIO-Max	NIO-Min	Diff	Loss Type	Separation of
300	0.285054	0.03434	0.250714	Full Trajectory	Whole Trajectory
500	1.011808	0.214497	0.797311	Full Trajectory	Whole Trajectory
900	5.160543	3.332232	1.828311	Full Trajectory	Whole Trajectory
1300	8.745911	7.974634	0.771277	Full Trajectory	Whole Trajectory
300	1.575575	0.091667	1.48391	Full Trajectory	Last Timestep
500	7.413527	1.313761	6.099766	Full Trajectory	Last Timestep
900	21.98312	9.821088	12.162032	Full Trajectory	Last Timestep
1300	18.29838	15.83575	2.46263	Full Trajectory	Last Timestep
300	0.224368	0.061597	0.162771	Last Step	Whole Trajectory
500	0.531563	0.332925	0.198638	Last Step	Whole Trajectory
900	4.238712	3.723658	0.515054	Last Step	Whole Trajectory
1300	8.488544	8.405669	0.082875	Last Step	Whole Trajectory
300	3.829242	0.063259	3.765983	Last Step	Last Timestep
500	7.541448	0.780186	6.761262	Last Step	Last Timestep
900	21.285336	9.634127	11.651209	Last Step	Last Timestep
1300	18.245144	16.381557	1.863587	Last Step	Last Timestep

Table 3 is summarized in the following two plots (Figures 9 and 10). As can be seen from the plots and the table, when evaluated using the last-timestep separation metric (Figure 9), the maximization objective exhibits its greatest separation at approximately 900 timesteps. This observation is qualitatively consistent with the Lyapunov-inspired motivation discussed in Equation (8).

**Figure 9.** Distance separation of last timestep.

Figures 9 and 10 show distance separation trajectories based on respective maximum or minimum discovered initial conditions. When looking at only the last timestep (Figure 9), the separation seems to be more pronounced when compared to looking at the average separation for a whole trajectory (Figure 9).

**Figure 10.** Distance separation of whole trajectory.

4.4. Results for Wing Selection Objective

In this section, we show the results for the wing selection objective. For the wing selection objective, the optimization was performed over a finite horizon of $T = 100$ timesteps. As can be seen in the Figure 11, NIO is able to find points that favor the left wing.

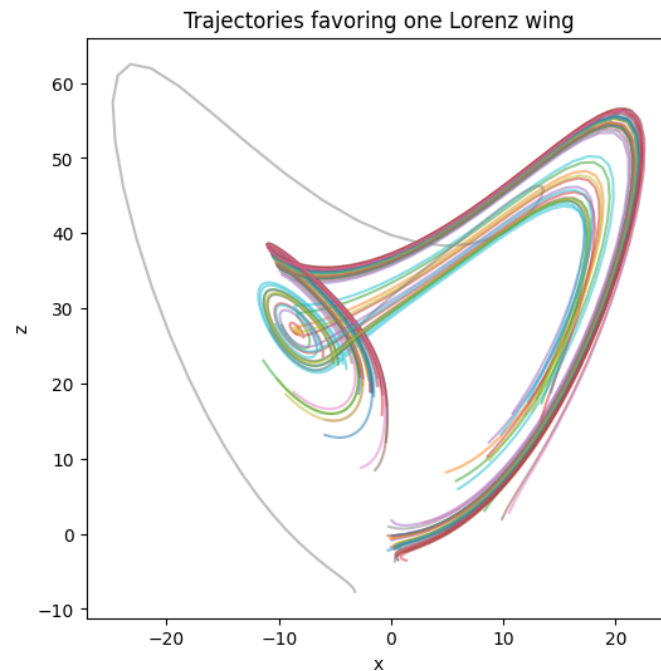


Figure 11. Left wing objective.

Figure 12 shows the case where NIO is able to find points that favor the right wing. These results show that by controlling the loss functions with NIO, we are able to find initial conditions that lead to desired behavior outcomes from the trajectories.

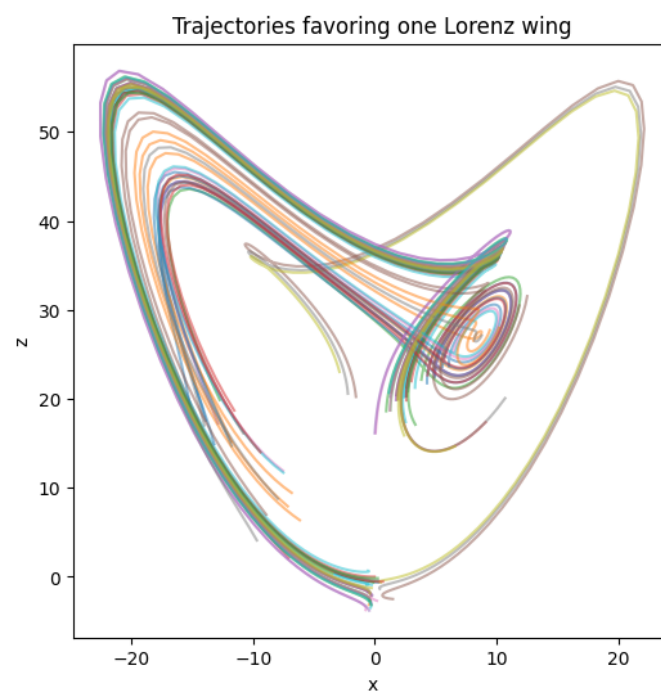


Figure 12. Right wing objective.

The code and relevant files for this work can be found on our GitHub repository located at [21].

5. Conclusions

In this work, we proposed a Neural Input Optimization (NIO) framework for structure search in dynamical systems. We demonstrated the methodology using the Lorenz system and showed that it can identify sets of initial conditions that satisfy different optimization objectives. Specifically, we considered two classes of loss functions: (1) a loss that identifies initial conditions leading to trajectories concentrated on either the left or right lobe of the Lorenz attractor, and (2) a loss that identifies initial conditions producing either maximum or minimum trajectory divergence. We also observed that the maximum separation occurred at around 900 steps in our trajectory simulation. These results suggest that NIO is a promising approach for scientific discovery and is well-suited for dynamical systems modeling. This work represents an initial investigation of NIO using the Lorenz system and equations, providing a foundation for future research involving more advanced dynamical systems and nonlinear evolution equations. We believe that our proposed data-driven NIO framework is flexible and can be applied to multiple problems by simply modifying the loss objective, as demonstrated by the wing selection objective and the trajectory divergence objective. Future work will focus on applying this methodology to different use cases such as dynamical systems in healthcare, materials science, and steelmaking.

Author Contributions: Conceptualization, R.A.C. and T.-H.K.; methodology, R.A.C. and T.-H.K.; software, R.A.C. and A.R.; validation, R.A.C. and A.R.; formal analysis, all authors; data creation, A.R.; writing—original draft preparation, R.A.C.; writing, all authors; visualization, R.A.C.; supervision, R.A.C. and T.-H.K.; project administration, T.-H.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The data used in this work is available on GitHub at https://github.com/arejith/NIO_StructureSearch_DynamicSystem (accessed on 28 June 2026).

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Lorenz, E.N. Deterministic Nonperiodic Flow. *J. Atmos. Sci.* **1963**, *20*, 130–141. [CrossRef]
2. Mandelbrot, B.B. *The Fractal Geometry of Nature*; W. H. Freeman and Company: New York, NY, USA, 1982.
3. Shaw, R. Strange Attractors, Chaotic Behavior, and Information Flow. *Z. Für Naturforschung A* **1981**, *36*, 80–112. [CrossRef]
4. Calix, R.A.; Kim, T.-H. Constraint-Based Neural Input Optimization (NIO) for Secure Token Generation. In *Proceedings of the IEEE 2025 Cyber Awareness and Research Symposium (CARS)*; IEEE: Piscataway, NJ, USA, 2025; pp. 1–6.
5. Shaw, R. The Dripping Faucet as a Model Chaotic System. Ph.D. Thesis, Aerial Press, Santa Cruz, CA, USA, 1984.
6. Gleick, J. *Chaos: Making a New Science*; Viking Press: New York, NY, USA, 1987.
7. Feigenbaum, M.J. Quantitative Universality for a Class of Nonlinear Transformations. *J. Stat. Phys.* **1978**, *19*, 25–52.
8. Eckmann, J.-P.; Ruelle, D. Ergodic Theory of Chaos and Strange Attractors. *Rev. Mod. Phys.* **1985**, *57*, 617–656. [CrossRef]
9. Brunton, S.L.; Proctor, J.L.; Kutz, J.N. Discovering Governing Equations from Data by Sparse Identification of Nonlinear Dynamical Systems. *Proc. Natl. Acad. Sci. USA* **2016**, *113*, 3932–3937. [CrossRef] [PubMed]
10. Weng, T.; Yang, H.; Zhang, J.; Small, M. Modeling Chaotic Systems: Dynamical Equations vs Machine Learning Approach. *Commun. Nonlinear Sci. Numer. Simul.* **2022**, *114*, 106452. [CrossRef]
11. Kochkov, D.; Smith, J.; Alieva, A.; Wang, Q.; Brenner, M.; Hoyer, S. Machine learning-accelerated computational fluid dynamics. *Proc. Natl. Acad. Sci. USA* **2021**, *118*, e2101784118. [CrossRef] [PubMed]
12. Usman, A.; Rafiq, M.; Saeed, M.; Nauman, A.; Almqvist, A.; Liwicki, M. Machine Learning Computational Fluid Dynamics. In *Proceedings of the 2021 Swedish Artificial Intelligence Society Workshop (SAIS), Luleå, Sweden, 14–15 June 2021*; IEEE: Piscataway, NJ, USA, 2021; pp. 1–4. [CrossRef]

13. Molesky, S.; Lin, Z.; Piggott, A.Y.; Jin, W.; Vuckovic, J.; Rodriguez, A.W. Inverse Design in Nanophotonics. *Nat. Photonics* **2018**, *12*, 659–670. [[CrossRef](#)]
14. Sanchez-Lengeling, B.; Aspuru-Guzik, A. Inverse Molecular Design Using Machine Learning: Generative Models for Matter Engineering. *Science* **2018**, *361*, 360–365. [[CrossRef](#)] [[PubMed](#)]
15. Park, J.; Xu, Z.; Park, G.M.; Luo, T.; Lee, E. Inverse Binary Optimization of Convolutional Neural Network in Active Learning Efficiently Designs Nanophotonic Structures. *Sci. Rep.* **2025**, *15*, 15187. [[CrossRef](#)] [[PubMed](#)]
16. Wang, J.; Wang, Y.; Chen, Y. Inverse Design of Materials by Machine Learning. *Materials* **2022**, *15*, 1811. [[CrossRef](#)] [[PubMed](#)]
17. Gatys, L.A.; Ecker, A.S.; Bethge, M. Image Style Transfer Using Convolutional Neural Networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*; IEEE: Piscataway, NJ, USA, 2016; pp. 2414–2423.
18. Thompson, J.M.T.; Stewart, H.B. *Nonlinear Dynamics and Chaos*; John Wiley & Sons: New York, NY, USA, 2002.
19. Wolf, A.; Swift, J.B.; Swinney, H.L.; Vastano, J.A. Determining Lyapunov Exponents from a Time Series. *Phys. D Nonlinear Phenom.* **1985**, *16*, 285–317. [[CrossRef](#)]
20. Guckenheimer, J.; Holmes, P. *Nonlinear Oscillations, Dynamical Systems, and Bifurcations of Vector Fields*; Springer: New York, NY, USA, 1983.
21. Rejith, A. Neural Input Optimization (NIO): Structure Search in Dynamical Systems. GitHub Repository. Available online: https://github.com/arejith/NIO_StructureSearch_DynamicSystem (accessed on 28 June 2026).

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.