Accelerating Numerical Solvers for Large-Scale Simulation of Dynamical System via NeurVec

Zhongzhan Huang\textsuperscript{1,*}, Senwei Liang\textsuperscript{2,*}, Hong Zhang\textsuperscript{3}, Haizhao Yang\textsuperscript{4}, and Liang Lin\textsuperscript{1,†}

\textsuperscript{1}School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou, China
\textsuperscript{2}Department of Mathematics, Purdue University, West Lafayette, IN, USA
\textsuperscript{3}Mathematics and Computer Science Division, Argonne National Laboratory, Lemont, IL, USA
\textsuperscript{4}Department of Mathematics, University of Maryland College Park, College Park, MD, USA
\textsuperscript{†}Correspondence should be addressed to: linliang@ieee.org
\textsuperscript{*}these authors contributed equally to this work, are listed with alphabetical order
\textsuperscript{♣}Technical report

ABSTRACT

Ensemble-based large-scale simulation of dynamical systems is essential to a wide range of science and engineering problems. Conventional numerical solvers used in the simulation are significantly limited by the step size for time integration, which hampers efficiency and feasibility especially when high accuracy is desired. To overcome this limitation, we propose a data-driven corrector method that allows using large step sizes while compensating for the integration error for high accuracy. This corrector is represented in the form of a vector-valued function and is modeled by a neural network to regress the error in the phase space. Hence we name the corrector neural vector (NeurVec). We show that NeurVec can achieve the same accuracy as traditional solvers with much larger step sizes. We empirically demonstrate that NeurVec can accelerate a variety of numerical solvers significantly and overcome the stability restriction of these solvers. Our results on benchmark problems, ranging from high-dimensional problems to chaotic systems, suggest that NeurVec is capable of capturing the leading error term and maintaining the statistics of ensemble forecasts.

Introduction

Dynamical systems are widely used to characterize the time dependence of the physical states and to model phenomena that change with time.\textsuperscript{1–3} Studying the temporal evolution of dynamical systems and their statistics can help uncover the physics behind the dynamics and predict future states of the systems.\textsuperscript{3} Typically, a time-dependent $d$-dimensional state $u(t)$ is depicted by a system of ordinary differential equations (ODEs):

$$\frac{du}{dt} = f(u), \quad u(0) = c_0, \quad (1)$$

where $c_0$ represents an initial condition. This system arises in many science and engineering fields such as climate change,\textsuperscript{4,5} air pollution,\textsuperscript{6,7} stable financial systems,\textsuperscript{8} power grid management,\textsuperscript{9,10} transportation networks,\textsuperscript{11,12} and medical analysis and drug discovery.\textsuperscript{13–15} To obtain a numerical solution of (1), one may employ an integration method\textsuperscript{16,17} (Fig. 1d) given by the iterative formula

$$u_{n+1} = u_n + S(f, u_n, \Delta t_n), \quad u_0 = c_0, \quad n = 0, 1, \cdots, \quad (2)$$

where $S$ represents a numerical scheme (for example, $S(f, u_n, \Delta t_n) := f(u_n)\Delta t_n$ when the Euler method\textsuperscript{18} is used), $\Delta t_n$ is the step size of the $n$th time step, and $u_n \in \mathbb{R}^d$ is an approximated solution at time $\sum_{i=0}^{n} \Delta t_i$. When approximating a solution at a specific time given an initial condition, we readily customize accuracy and speed via tuning integration strategies (e.g., different scheme $S$ and step size $\Delta t_n$ selection). However, many real-world applications\textsuperscript{19–22} require simulating many trajectories. In particular, large-scale simulation (Fig. 1a), which produces forecasts on a set of initial conditions simultaneously, is more useful for these applications. Compared with a single simulation, ensemble-based large-scale simulation is a computationally challenging problem but plays a critical role in a variety of demanding applications. For illustration, we present a few scenarios of such simulation.

Fast simulation. Since late 2019, the epidemic of COVID-19 has raged around the world, hitting the global health and economy.\textsuperscript{23} Scientists need to perform simulations of virus propagation under different circumstances.\textsuperscript{24,25}
These predictions provide the scientific reference for governments to make quick responses and control policies.\cite{26,27} The virus, such as the Delta and Omicron variants, spreads rapidly and mutates frequently.\cite{28} A slow simulation may lead to a delay in decision-making and worsen the situation.

**Synchronous simulation.** Particle systems are a graphical technique that simulates complex physical effects (such as smoke,\cite{29} water flow,\cite{30} and object collision.\cite{31}). This is widely used in applications in games, movies, and animation.\cite{32} These applications involve synchronously simulating thousands of particles at one time. Yet supporting the real-time simulation of these particle motions with satisfactory visual enjoyment is expensive.

**Reliable model.** Weather forecasting is beneficial for making a proper plan for production and living.\cite{33,34} A single forecast of the weather model essentially suffers from considerable errors introduced by the highly simplified model formulation and the chaotic nature of the atmosphere evolution equations. In order to avoid a misleading single forecast, ensemble forecasting\cite{35–37} presents a range of possible future weather states through conducting simulations from multiple initial conditions and models.

To meet the demands of these applications, we need to develop a fast solver that is capable of simultaneously simulating the dynamical system over a large batch of initialization data. The advances in processors, such as graphics processing units (GPUs),\cite{38} tensor processing units,\cite{39} and natural graphics processing units,\cite{40} provide the
possibility of accelerating the numerical computation via parallel computing of batch data. However, most hardware implements restrictive SIMD-based (single instruction, multiple data) models.\textsuperscript{41} The numerical method that needs individual processing of each trajectory is not appropriate for SIMD processors directly. For example, the adaptive time-step integrator (e.g., the Runge-Kutta–Fehlberg method\textsuperscript{42}) determines a step size at each step based on an estimate of the local error, making the iterative computation in Eq. (2) asynchronous for each trajectory in the batch and affecting the efficiency of parallel computing. On the other hand, we may control the step size to be the same at each step for all trajectories by adding one dimension to Eq. (2), for example, \(u_{n+1} \in \mathbb{R}^{N \times d}\) with \(N\) representing the batch size. Controlling the step size requires considering a combined ODE system and estimating the error on all batch elements.\textsuperscript{43} The step size is limited by the largest local truncation error in a batch, making it difficult to use a large step size especially when the batch size is large.\textsuperscript{43,44} If the step size is always small in each step, it becomes slow for integration. Therefore, SIMD processors prefer a fixed time-step integrator (i.e., \(\Delta t := \Delta t_1 = \Delta t_2 = \cdots\)),

\[
u_{n+1} = u_n + S(f, u_n, \Delta t), \quad u_0 = c_0, \quad n = 0, 1, \cdots.
\]

However, a fixed step size integrator encounters a trade-off\textsuperscript{14,45} on step size between accuracy and computational efficiency: a large step size has a fast simulation but leads to a less accurate solution, while a small step size has a slow simulation but achieves a more accurate solution (see Table 1 for the comparison of evaluation time and theoretical error between the traditional solvers with fine or coarse step size). This trade-off limits the feasibility of large-scale simulation if high accuracy is required.

To break through this speed-accuracy trade-off, in this paper we propose an open-source and data-driven corrector, called neural vector (NeurVec), which enables integration with coarse step size while maintaining the accuracy of fine step size in large-scale simulations. We empirically demonstrate that NeurVec is capable of overcoming the stability restriction of explicit integration methods for ODEs. The deployment of NeurVec comprises offline training and inference (Fig. 1b). During offline training, NeurVec is trained with the accurate solution, while during inference NeurVec is employed to the solver to compensate for the error caused by the coarse step size. Our results on benchmark problems, ranging from high-dimensional problems to chaotic systems, show that NeurVec is capable of learning the error term and accelerating the large-scale simulation of dynamical system significantly. Also, we have found that NeurVec can overcome the stability restriction of explicit integration methods for ODEs. Previous works also consider accelerating the solution estimation via deep learning, but they emphasize adopting pure data-driven approaches,\textsuperscript{3,45,46} without using any explicit formula of the equation. Because of chaos\textsuperscript{47–49} (solution is sensitive to small perturbations) and stiffness\textsuperscript{14,50} (solution is unstable unless a sufficiently small step size is used), the pure data-driven method still suffers from large errors in prediction, especially for long-term prediction.\textsuperscript{51}

**Framework of NeurVec**

The corrector NeurVec, in the form of a vector function, is directly added to the estimated solution to compensate for the error caused by the use of the coarse step size (Fig. 1c). To learn the complicated error distribution on the phase space, we adopt a neural network, a universal approximator for continuous functions,\textsuperscript{52,53} to model this corrector. Specifically, NeurVec, a neural network parameterized by \(\Theta\), maps from the state \(\mathbb{R}^d\) to the error correction \(\mathbb{R}^d\) (see the Methods section) and is added to the iterative formula of the solver with \(k\) times the step size \((k \Delta t)\), namely,

\[
\hat{u}_{k(n+1)} = \hat{u}_{kn} + S(f, \hat{u}_{kn}, k \Delta t) + \text{NeurVec}(\hat{u}_{kn}; \Theta), \quad \hat{u}_0 = c_0, \quad n = 0, 1, \cdots.
\]

With NeurVec, we just need to estimate the solution on every \(k\) steps instead of step by step as in Eq. (3). NeurVec is trained from the more accurate solutions with fine step size \(\Delta t\) to characterize the error caused by the use of the coarse step size \(k \Delta t\). The parameter \(\Theta\) in NeurVec can be optimized by minimizing the mean squared difference between the predicted error and the error of the solver with the coarse step size:

\[
\min_{\Theta} \frac{1}{G} \sum_{n=1}^{G} \left\| \text{NeurVec}(u_{kn}; \Theta) - (u_{k(n+1)} - u_{kn} - S(f, u_{kn}, k \Delta t)) \right\|_{2}^{2},
\]

where \(G\) is the number of training samples. Table 1 displays a comparison of evaluation time and theoretical error between the traditional solver and NeurVec. We use \(\epsilon\) to denote the runtime ratio of NeurVec to the scheme \(S\). NeurVec inevitably increases the relative time complexity for each step by \(\epsilon\) since an additional computation module is used. When \(k > (1 + \epsilon)\), NeurVec with the coarse step size \(k \Delta t\) is faster than the solver with the fine step size \(\Delta t\), while achieving comparable accuracy. Moreover, the runtime increment \(\epsilon\) of NeurVec can be lessened. For example, the more complicated scheme \(S\) increases the time complexity, and built-in parallel computing in Pytorch,\textsuperscript{54}
We demonstrate the performance of NeurVec on widely used numerical solvers with consistent performance improvement. To characterize the solution error of NeurVec, we consider the Euler method, a simple ODE solver, as a proof of concept. The global truncation error of the Euler method linearly grows with the step size, namely, \( O(\Delta t) \) when the step size is \( \Delta t \) and \( O(k\Delta t) \) when the step size is \( k\Delta t \). In our theory, we show that NeurVec of sufficient width can achieve an error of \( O(\Delta t) \) when the step size is \( k\Delta t \), which breaks the accuracy-speed trade-off.

| Method                          | Step size          | Evaluation time       | Theory error (Euler scheme) |
|--------------------------------|--------------------|-----------------------|-----------------------------|
| Fixed step size solver (fine step size) | \( \Delta t \) | \( O(1/\Delta t) \) | \( O(\Delta t) \)          |
| Fixed step size solver (coarse step size) | \( k\Delta t \) | \( O(1/(k\Delta t)) \) | \( O(k\Delta t) \)         |
| NeurVec (coarse step size)       | \( k\Delta t \) | \( O((1+\epsilon)/(k\Delta t)) \) | \( O(\Delta t) \)          |

**Table 1.** Comparison of evaluation time and theoretical error (based on the Euler scheme) among the numerical solvers with fine \((\Delta t)\) or coarse step size \((k\Delta t)\) and NeurVec \((k\Delta t)\). Here \( \epsilon \) denotes the ratio of the runtime of NeurVec to that of scheme \( S \) for one step. The fixed step size solvers suffer from the accuracy-speed trade-off on the step size. NeurVec learns from the solutions of fine step size. Then NeurVec is applied to the solver and integrates with the coarse step size \((k\Delta t)\) but still has the theoretical accuracy of the fine step size, \( O(\Delta t) \).

**Results**

We verify the capabilities of NeurVec in two aspects: (1) NeurVec is capable of stabilizing and accelerating the simulation on widely used numerical solvers with consistent performance improvement; and (2) NeurVec can be applied effectively to various benchmark problems, ranging from high-dimensional simulation to chaotic dynamics.

To illustrate the performance of NeurVec, we employ a simple network structure, a one-hidden-layer fully connected neural network, to model NeurVec, where the number of the hidden neurons is 1,024 and a rational function is used (see the Methods section for details). The training and inference of NeurVec are all performed on a single GeForce RTX 3080 GPU with a memory of 10 gigabytes. The simulations in the training and testing sets are uniformly sampled every time interval \( \eta \). We first introduce the training dataset to train NeurVec and the testing dataset for evaluation. The training and testing simulations are uniformly sampled every time interval \( \eta = 2e−1 \). The initial states are sampled randomly from uniform distribution \( \pi := \mathbb{U}([-2.5, 2.5]^d \times [-2.5, 2.5]^d) \). We set the dimension \( d = 20 \) so the dimension of the state is 40. Given a scheme \( S \), the training dataset is generated by \( S \) with \( \Delta^F T = 1e−3 \). The reference simulations in the testing set are generated by RK4 with sufficiently small step size \( 1e−4 \) (see the supplementary material).

Next, we demonstrate the performance of NeurVec in terms of accuracy and speed. NeurVec learns from the simulations of \( \Delta^F T = 1e−3 \) and is applied to the numerical solver with \( \Delta^C T = 2e−1 \). We characterize accuracy by the MSE between the reference and the simulated solution. In the short-term simulation on the time interval...
Figure 2. Application of NeurVec on different numerical solvers. a, The mean square error (MSE) between the reference solution and the numerical solutions with different configurations (step size $\Delta C_t = 1e-1$, $\Delta C_t = 2e-1$, $\Delta F_t = 1e-3$, and NeurVec ($\Delta NV_t = 2e-1$)) on the spring-chain system, averaged over 10.5k simulations. The reference solution is obtained by using the 4th-order Runge-Kutta with step size $1e-4$. NeurVec is trained on the simulations of $\Delta F_t = 1e-3$. The numerical solution of $\Delta C_t = 2e-1$ becomes unstable using Euler or the improved Euler formula, while NeurVec ($\Delta NV_t = 2e-1$) achieves a stable solution with accuracy comparable to that of $\Delta F_t = 1e-3$. b, The (normalized) runtime of the numerical solver with $\Delta F_t = 1e-3$, $\Delta C_t = 2e-1$, and NeurVec ($\Delta NV_t = 2e-1$). The runtime of $\Delta C_t = 2e-1$ is benchmarked to one unit. NeurVec ($\Delta NV_t = 2e-1$) has accuracy similar to that of $\Delta F_t = 1e-3$ and is over 150 times faster.

[0,17], the numerical solutions of the coarse step size ($\Delta C_t \geq 1e-1$) incur considerable simulation error and become unstable if Euler and the improved Euler are used (Fig. 2a). By contrast, NeurVec ($\Delta NV_t = 2e-1$) achieves a stable solution with accuracy comparable to that of the fine step size $\Delta F_t = 1e-3$ (Fig. 2a), which means that NeurVec can overcome the stability restriction. Moreover, we verify the long-term performance of NeurVec. On time intervals [180,200] (Fig. 3a) and [570,600] (Fig. 3b), NeurVec ($\Delta NV_t = 2e-1$) has the consistent trend of error change with time and still maintains accuracy of the same magnitude as $\Delta C_t = 1e-3$. These observations indicate that NeurVec learns the error distribution from the fine step size dataset and compensates for errors caused by the use of the coarse step size, demonstrating that NeurVec is compatible to these solvers. To better display the runtime, we benchmark the runtime of $\Delta C_t = 2e-1$ as one unit. The use of NeurVec increases for a certain runtime ($\epsilon \leq 0.38$) for a single step (compared with $\Delta C_t = 2e-1$), but NeurVec has accuracy comparable to that of $\Delta F_t = 1e-3$, which needs 200 steps for integrating over the time interval $2e-1$. The runtime of $\Delta F_t = 1e-3$ is much higher than that of NeurVec ($\Delta C_t = 2e-1$) ($P$ value $\ll 0.001$ under two-sided t-tests), and NeurVec enables these numerical methods to have more than 150× speedup on the spring-chain systems (Fig. 2b).

NeurVec on chaotic dynamical systems.

We verify the effectiveness of NeurVec on challenging chaotic systems, including the Hénon–Heiles system, elastic pendulum, and K-link pendulum. The chaotic system is sensitive to perturbation to the initial state, and small errors are increased exponentially by the dynamics. For all of these examples, we generate the testing set using RK4 with step size $1e-4$ while the training set is generated with $\Delta F_t = 1e-3$. The initial conditions are randomly and uniformly sampled on a range of values (see the supplementary material). NeurVec is applied to RK4.

(1) Hénon–Heiles system. The Hénon–Heiles system is a Hamiltonian system\textsuperscript{58} that describes the motion of a
body around a center on the \(x-y\) plane. Let \((q_x, q_y)\) and \((p_x, p_y)\) denote the positions and momenta of a particle, respectively. The ODE is given by

\[
\frac{d}{dt}(q_x, q_y, p_x, p_y) = (p_x, p_y, -q_x - 2\lambda q_x q_y, -q_y - \lambda (q_x^2 - q_y^2)).
\] (7)

The Hamiltonian (energy) function \(\mathcal{H}\), defined by

\[
\mathcal{H}(q_x, q_y, p_x, p_y) = \frac{1}{2}(p_x^2 + p_y^2) + \frac{1}{2}(q_x^2 + q_y^2) + \lambda(q_x^2 q_y - \frac{q_y^2}{3}),
\] (8)

must be conserved during the time evolution. This property is used as an additional metric to evaluate the accuracy of our method. We characterize the energy error by the absolute difference between the energy of the simulated trajectory and the initial energy. The datasets are generated with initial energy between \([0, 12]\) and \(-1 < q_x < 1, -0.5 < q_y < 1\) such that the equipotential curves of the system form an inescapable interior region and exhibit chaotic behavior\(^5\). The simulations are uniformly sampled every time interval \(\eta = 5e-1\).

We find that NeurVec \((\Delta^\text{NV} t = 5e-1)\) vastly improves the accuracy of the ODE solvers, achieves almost the same accuracy as RK4 with \(\Delta^F t = 1e-3\) on the time interval \([0, 42.5]\) (Fig. 4a), and works well for a much larger time interval \([450, 500]\) (Fig. 4c). Furthermore, NeurVec with \(\Delta^\text{NV} t = 5e-1\) almost maintains the same system energy as does the reference method with \(\Delta^F t = 1e-3\) (Fig. 4d). To illustrate the error correction capability of NeurVec, we visualize three trajectories of the first three components \((q_x, q_y, p_x)\) in Fig. 4e. For Examples 1–3 of Fig. 4e, NeurVec with \(\Delta^\text{NV} t = 5e-1\) produces orbits similar to those of the reference method with \(\Delta^F t = 1e-3\) while having an energy error of the same magnitude. Furthermore, the reference method with \(\Delta^C t = 5e-1\) yields a larger energy error and pathwise difference. Integrating with \(\Delta^F t = 1e-3\) over the time interval \(\eta = 5e-1\) takes 500 steps, so it is not surprising that the runtime for \(\Delta^F t = 1e-3\) is much larger than NeurVec with \(\Delta^\text{NV} t = 5e-1\). Based on our test, NeurVec \((\Delta^\text{NV} t = 5e-1)\) reaches more than \(390\times\) speedup over the reference method with \(\Delta^F t = 1e-3\) (Fig. 4b).

(2) \textbf{Elastic pendulum.} The elastic pendulum describes a point mass connected to a spring swinging freely (Fig. 5a), which may exhibit chaotic behavior under the force of gravity and spring\(^6\). We denote \(\theta\) as the angle between the spring and the vertical line and \(r\) as the length of the spring. \(\dot{\theta}\) and \(\dot{r}\) correspond to the time derivative of \(\theta\) and \(r\), respectively. The motion of this system is governed by the ODE,

\[
\frac{d}{dt}(\theta, r, \dot{\theta}, \dot{r}) = \left(\dot{\theta}, \dot{r}, \frac{1}{r}(-g \sin \theta - \dot{\theta} \dot{r}), r \dot{\theta}^2 - \frac{k}{m}(r - l_0) + g \cos \theta\right),
\] (9)
We provide three examples of trajectories projected on the coordinates $q_x, q_y, p_x, p_y$, and the corresponding energy error on $[0, 50]$. More examples can be found in the supplementary material.

where $k, m, l_0$, and $g$ are spring constant, mass, original length, and gravity constant, respectively. The initial length of $r$ is $r(0) = l_0 = 10$. $\tilde{r}$ and $\tilde{\theta}$ are initialized by constant 0, and $\theta$ is randomly sampled from the uniform distribution $\mathcal{U}([0, \pi])$. The simulations in the training and testing sets are uniformly sampled every time interval $\eta = 1e-1$. NeurVec is trained on the simulation generated by $\Delta^F t = 1e-3$. NeurVec ($\Delta^{NV} t = 1e-1$) has accuracy of the same order as does $\Delta^F t = 1e-3$ on both short-term prediction (time interval $[0, 8.5]$) (Fig. 5b) and long-term prediction (time interval $[25, 50]$) (Fig. 5d). NeurVec ($\Delta^{NV} t = 1e-1$) is much faster than $\Delta^F t = 1e-3$ ($P$ value $\ll 0.001$ under two-sided t-tests), reaching about 70× speedup (Fig. 5c).

(3) **K-link pendulum.** A K-link pendulum is a body suspended from a fixed point (Fig. 5a) with $K$ rods and $K$ bobs so that the body can swing back and forth under gravity.\(^{61}\) The system exhibits chaotic behavior. For simplification, the length of each rod and the mass of each bob are set to 1, and the gravity constant $g$ is set to 9.8. Let variables $\theta := (\theta_1, \theta_2, \cdots, \theta_K)$, where $\theta_i$ is the angle between the $i$th rod and the vertical axis. The system is governed by the ODE

$$
\frac{d}{dt}(\theta, \dot{\theta}) = (\dot{\theta}, A^{-1} \mathbf{b}).
$$

**(Figure 4. Performance comparison on the Hénon–Heiles system.** a, MSE with varied time on the time interval $[0,42.5]$ under different configurations (step size $\Delta^C t = 5e-1$, $\Delta^C t = 2e-1$, $\Delta^F t = 1e-3$, and NeurVec ($\Delta^{NV} t = 5e-1$)). b, MSE with varied time on the longer time interval $[450,500]$. The upper and lower bounds of the light color indicate the maximal and minimal error, respectively. c, The (normalized) runtime of the numerical solver with $\Delta^F t = 1e-3$, $\Delta^C t = 5e-1$, and NeurVec ($\Delta^{NV} t = 5e-1$). d, Energy error with varied time on $[0,50]$. e, We provide three examples of trajectories projected on the coordinates $(q_x, q_y, p_x, p_y)$, and the corresponding energy error on $[0,50]$. More examples can be found in the supplementary material.)
we compare the evaluation time and solution error among solver with fine or coarse step size and NeurVec.

We validate the performance of NeurVec on producing consistent statistical observations for ensemble forecasting.

which is in accordance with the periodic variation of the spring during its extend-retract. However, let

The ability to enable a large step size for a set of sampled initial conditions is critical for real applications such as weather forecasting. We visualize the time series histogram of the testing set for variables (a) \( r \) and \( \dot{r} \) in the elastic pendulum (9) and (b) \( q_1 \) and \( p_1 \) in the Hénon–Heiles system (7) in Fig. 6. The time series histogram is generated by dividing axes into 800×100 bins and counting the curves that cross the bins.

For the elastic pendulum, we find that starting from a time \( T \geq 25 \), the statistical difference of \( \dot{r} \) and \( r \) between \( \Delta F t = 1e-3 \) and \( \Delta C t = 1e-1 \) becomes larger. When the step size is \( \Delta F t = 1e-3 \), \( \dot{r} \) exhibits periodic behavior, which is in accordance with the periodic variation of the spring during its extend-retract. However, let \( \Delta C t = 1e-1 \), \( \dot{r} \) and \( r \) show a trend of approaching specific values, and the change range gradually narrows. On the other hand, the simulations with NeurVec (\( \Delta NV t = 1e-1 \)) have a pattern similar to that of \( \Delta F t = 1e-3 \). We have a similar observation for \( q_1 \) and \( p_1 \) in Hénon–Heiles system (Fig. 6b). Therefore, we conclude that NeurVec produces more accurate solutions compared with the reference method with large step size, enabling better and more consistent statistical observation.
Figure 6. Time series histogram. We visualize the time series histogram of a test set for variables (a) \( r \) and \( \dot{r} \) in the elastic pendulum (9) and (b) \( q_y \) and \( p_y \) in the Hénon–Heiles system (7). The color represents the number count (the lighter color and the larger frequency). The solutions generated by the solver with coarse step size exhibits a trend of convergence to a specific value, while solutions of the solver with fine step size are distributed within a range, and NeurVec with coarse step size produces a histogram visually identical with that of the solver with fine step size. This result shows that NeurVec has a more accurate solution than does the solver with fine step size.

Error visualization on the phase space.

Neural networks are generally considered black-box functions and lack interpretability. In this section we explore what the neural network in NeurVec learns in order to explain the good performance of NeurVec. We consider solving a 1-link pendulum with the Euler method. Our consideration for testing NeurVec on this system is based on the following motivations. First, the dimension of the state is 2, which facilitates error visualization on phase space. Second, we derive the error term of the Euler method explicitly through the Taylor formula:

\[
\mathbf{u}(t + \Delta t) - (\mathbf{u}(t) + \mathbf{f}(\mathbf{u})\Delta t) = \frac{1}{2}(\nabla \mathbf{f})\mathbf{f}(\mathbf{u})\Delta t^2 + \mathcal{O}(\Delta t^3),
\]

where \( \nabla \mathbf{f} \) is the Jacobian matrix of \( \mathbf{f} \). The second-order term \( \frac{1}{2}(\nabla \mathbf{f})\mathbf{f}(\mathbf{u})\Delta t^2 \) is the leading error term of the Euler method, which is supposed to be captured by NeurVec from data of fine step size. Denote \( R_{NV}(\mathbf{u}) := \|\text{NeurVec}(\mathbf{u})\|^2 \) as the norm of error learned by NeurVec and \( R_{EL}(\mathbf{u}) := \left\| \frac{1}{2}(\nabla \mathbf{f})\mathbf{f}(\mathbf{u})\Delta t^2 \right\|_2^2 \) as the norm of the leading error term of the Euler method. To train NeurVec for the Euler method, we generate the dataset by randomly sampling the initial conditions of \( \theta \) and \( \dot{\theta} \) from uniform distributions \( U([0, \pi/2]) \) and \( U([0, 0.5]) \), respectively, and then use the Euler method to simulate the data with \( dt = 1e-3 \). We train NeurVec with coarse step size \( dt = 1e-1 \).

We found that the learned error \( R_{NV} \) (Fig. 7a) is visually consistent with the leading error \( R_{EL} \) of the Euler method (Fig. 7b). The squared difference \( R_{Diff} = \left\| \frac{1}{2}(\nabla \mathbf{f})\mathbf{f}(\mathbf{u})\Delta t^2 - \text{NeurVec}(\mathbf{u})\right\|_2^2 \) is up to order \( \mathcal{O}(10^{-6}) \), and a small part of the difference near the boundary is relatively large (Fig. 7c). Through the training data of high accuracy,
NeurVec captured the leading error term of the numerical solver. NeurVec may even capture the higher-order error terms, which enable the use of a coarse step size.

**Theoretical analysis.** We analyze the runtime and the global error in the solution approximated with NeurVec. Let $0 = t_0 < t_1 < \cdots < t_{pk} = T$ be uniform points on $[0, T]$ and $\Delta t = \frac{T}{pk}$.

We compare the runtime of fine and coarse step size. If the step size $\Delta t$ is used, then the number of steps for integration is $\frac{T}{\Delta t}$. If the step size is $k\Delta t$, then the number of steps needed is $\frac{T}{k\Delta t}$. $\epsilon$ is the ratio of the runtime of NeurVec to that of scheme $S$ for one step. Hence, when NeurVec is used to integrate with step size $k\Delta t$, we need $\epsilon \times 100\%$ extra time for each step; and the time becomes $\Theta(\frac{T(1+\epsilon)}{k\Delta t})$.

Next we study the error of solvers with fine or coarse step size. For simplification, we focus on the Euler solver and characterize the global discretization error (difference between the true solution and the estimated solution) at the time $T$. When the Euler scheme is used,

$$u_{n+1} = u_n + \Delta t f(u_n), \quad u_0 = c_0, \quad n = 0, 1, \cdots, kp - 1. \quad (12)$$

**Proposition 0.1.** We assume that (1) $f$ is Lipschitz continuous with Lipschitz constant $L$ and (2) the second derivative of the true solution $u$ is uniformly bounded by $M > 0$, namely, $\|u''\|_\infty \leq M$ on $[0, T]$. Then, using (12), we have

$$|u_{kp} - u(T)| \leq \frac{M \exp(2TL)}{2L} \Delta t.$$

For the proof, see the work of Atkinson et al.62. Proposition 0.1 shows that the Euler method converges linearly. The error is $O(\Delta t)$ when the step size $\Delta t$ is used. Similarly, the error becomes $O(k\Delta t)$ when the step size is $k\Delta t$.

We next derive the error for the Euler method with step size $k\Delta t$ using NeurVec. The iterative formula is given by

$$\hat{u}_{k(n+1)} = \hat{u}_{kn} + f(\hat{u}_{kn})(k\Delta t) + \text{NeurVec}(\hat{u}_{kn}; \Theta), \quad \hat{u}_0 = c_0, \quad n = 0, 1, \cdots, p - 1. \quad (13)$$

We can use the following loss function to identify the learnable parameter $\Theta$ in NeurVec. $V_n$ denotes the residual error for each term.

$$\text{LS} = \frac{1}{p} \sum_{n=0}^{p-1} \left\| \frac{u_{k(n+1)} - u_{kn}}{k\Delta t} - f(u_{kn}) - \frac{\text{NeurVec}(u_{kn}; \Theta)}{k\Delta t} \right\|^2 = \frac{1}{p} \sum_{n=0}^{p-1} \|V_n\|^2 \quad (14)$$

In the next theorem we characterize the error of NeurVec ($k\Delta t$) by the quality of the training data and the neural network training error. In addition to the assumption in Proposition 0.1, we assume NeurVec is Lipschitz continuous and the Lipschitz constant is of order $k\Delta$. This assumption is reasonable based on the following motivation. According to Taylor expansion $v(t + \Delta t) = v(t) + v'(t)\Delta t + o(\Delta t)$, from our objective we expect that NeurVec $\sim o(k\Delta t)$. 

Figure 7. Numerical error visualization on the phase space of 1-link pendulum. a, The square sum of leading (second-order) error term of the Euler method, denoted by $R_{EL}$. The error is calculated by using the true dynamics $f$. b, The square sum norm of error compensation learned by NeurVec, denoted by $R_{NV}$. c, The difference between the leading error term and NeurVec.
The premise of our method to accelerate the numerical method is that NeurVec enables the use of coarse step size with Lipschitz constant $k\Delta t L_{NV}$, which is independent of $\theta$. Then the error is

$$ |\hat{u}_{kp} - u(T)| \leq \frac{M \exp(2TL)}{2L} \Delta t + \frac{\sqrt{T} \exp(T(L + L_{NV}))}{\sqrt{L + L_{NV}}} (LS)^{1/2}. \tag{15} $$

The first term in the right-hand side of (15) comes from the error of the training data, Euler simulation with step size $\Delta t$, while the second term is the training error of NeurVec. A series of works\textsuperscript{63–65} utilize a neural tangent kernel to prove the global convergence of a neural-network-based least squares method. Under the assumptions of training data distribution, when the width of one hidden layer network is sufficiently large, gradient descent converges to a globally optimal solution for the quadratic loss function. We might assume that the training error $LS \rightarrow 0$ as the increasing update iteration. Then in (15), $|\hat{u}_{kp} - u(T)| \sim O(\Delta t)$.

### Discussion

To address the speed-accuracy trade-off in large-scale simulations of dynamical systems, we proposed NeurVec, a neural-network-based corrector, to compensate for the error caused by the use of coarse step size for numerical solvers. Through extensive experiments and preliminary theoretical evidence, we show that NeurVec is general and can be applied to widely used explicit integration methods and learn the error distribution through simulations with fine step size. However, NeurVec has the following limitations.

1. The premise of our method to accelerate the numerical method is that NeurVec enables the use of coarse step size. Therefore, NeurVec needs to be trained well enough, or it may not capture the error distribution on the phase space. Our numerical results reveals that NeurVec may have a slightly larger error than that of the fine step size. The main contribution of this paper is to provide a methodology based on deep learning to accelerate numerical methods while ignoring tedious training techniques. For example, we use only the simplest fully connected neural network, as shown in Fig. 8, with conventional training techniques (such as least squares loss in Eq. (5)) in deep learning. Hence, the performance of NuerVec could be improved by using more advanced learning architectures and training algorithms, for example, an attention mechanism,\textsuperscript{66–68} neural network structure search,\textsuperscript{69–71} and large-scale pretrained models.\textsuperscript{72}

2. We model NeurVec via deep learning, so a neural network must be trained, a process that can require considerable costs, such as training and data acquisition costs. Fortunately, with rapid advances in high-performance computer hardware, training time is generally acceptable. For example, all experiments in this paper were trained on a consumer-grade personal computer (CPU: AMD Ryzen 9 5950X, GPU: GeForce RTX 3090 with 24 GB of memory), and the training time does not exceed 72 hours. In addition, we could further compress the training time through multicard training, which is commonly used in the artificial intelligence community. When NuerVec is trained and deployed as shown in Fig. 1e, it can be used without additional training before each use, so the cost of training is practically constant. With regard to the cost of data acquisition, first we need to have enough hard drives to support data acquisition. The price of consumer-grade hard drives is decreasing year by year, so the cost of storage is acceptable to most institutions, laboratories, and even individuals. Next, as shown in Fig. 1e, it unavoidably takes a long time to acquire the required data in a given simulation environment. Moreover, some applications, such as weather forecasting and traffic forecasting, already store a sufficient amount of historical data, which can be used for training the NeurVec.

In addition, we may extend NeurVec in several ways.

- **Generalizing NeurVec.** We implemented NeurVec using the simulation data with fixed system parameters and fixed step size. Recently, the neural network, as a universal approximator, shows promising results on learning the nonlinear continuous operator. Motivated by operator learning,\textsuperscript{45,46} we may add additional dimensions to the input of NeurVec, such as the step size for integration and the physical parameters in the system. Then NeurVec can be trained with more diverse simulation data, such that NeurVec can be used with different $dt$ for systems with varied physical parameters (such as $k$ and $m$ in the spring-chain system (6)).

- **Continual model update.** Neural networks may sometimes have inaccurate predictions when encountering abnormal situations. Therefore, we may need to maintain and update NeurVec regularly to learn from new data. The simplest strategy is to retrain the network from scratch, but it needs considerable computing resources to train and memory resources to store the data. To address such a problem, we can fine-tune NeurVec via incremental learning\textsuperscript{73} for a small amount of newly collected training data to achieve low-cost model updates.
• **Numerical solvers and cutting-edge problems.** Four forward numerical solvers and four kinds of problems were considered. We may extend NeurVec to other types of numerical solvers, such as backward methods and implicit methods. These methods are mainly aimed at improving the simulation accuracy, but the simulation cost for one step may be large. Furthermore, since NeurVec is open source and easy to use, we believe that it can be applied to more cutting-edge applications in industry or scientific problems.

**Methods**

**Datasets.** We summarize the simulated training and testing datasets used in the main text in Table 2. For each dataset we integrate with the step size $\delta$ using the numerical solver over $N$ random initializations. We obtain the discrete solutions every $\delta$ up to the model time $T$. Next, we sample the solution every time interval $\eta$ ($\eta$ is a multiple of $\delta$).

| Problem                  | Type     | Dim | Num  | Step size $\delta$ | Method            | Duration $T$ |
|--------------------------|----------|-----|------|---------------------|-------------------|--------------|
| Spring-chain (Euler)     | Train    | 40  | 60k  | 1e-3                | Euler             | 20           |
| Spring-chain (Improved Euler) | Train    | 40  | 60k  | 1e-3                | Improved Euler    | 20           |
| Spring-chain (RK3)       | Train    | 40  | 60k  | 1e-3                | RK3               | 20           |
| Spring-chain (RK4)       | Train    | 40  | 60k  | 1e-3                | RK4               | 20           |
| Spring-chain             | Test     | 40  | 10.5k| 1e-4                | RK4               | 20           |
| 1-link pendulum          | Train    | 2   | 1k   | 1e-3                | RK4               | 10           |
| 2-link pendulum          | Train    | 4   | 300k | 1e-3                | RK4               | 10           |
| 2-link pendulum          | Test     | 4   | 7k   | 1e-4                | RK4               | 10           |
| Hénon–Heiles (Train)     | Test     | 4   | 70k  | 1e-4                | RK4               | 50           |
| Hénon–Heiles (Test)      | Train    | 4   | 100k | 1e-3                | RK4               | 50           |
| Elastic pendulum         | Train    | 4   | 300k | 1e-3                | RK4               | 50           |
| Elastic pendulum         | Test     | 4   | 14k  | 1e-4                | RK4               | 50           |

Table 2. Summary of the simulated datasets used in the main text.

**Numerical solvers.** We introduce four numerical solvers used in our paper: the Euler method, improved Euler method, and 3rd- and 4th-order Runge–Kutta methods. These solvers have different $S(f, u_n, \Delta t_n)$ in the iterative formula (2). (1) The Euler method can be written as

$$S(f, u_n, \Delta t_n) = \Delta t_n f(u_n).$$

(16)

It has an explicit geometric interpretation—it uses a series of line segments to approximate the solution of the equation. It is first-order accurate since its local truncation error is $O(\Delta t^2)$ and the global error is $O(\Delta t)$.

(2) The improved Euler method\(^4,75\) can be written as

$$S(f, u_n, \Delta t_n) = \frac{\Delta t_n}{2} [f(u_n) + f(u_n + \Delta t_n f(u_n))].$$

(17)

The improved Euler method is a numerical method that uses an implicit trapezoidal formulation to improve the accuracy of the Euler method. Specifically, it first takes a one-step Euler method to obtain $\tilde{u}_{n+1} = u_n + \Delta t_n f(u_n)$ and then uses the implicit trapezoidal formula to obtain $u_{n+1} = u_n + \frac{\Delta t_n}{2} [f(u_n) + f(\tilde{u}_{n+1})]$. Even though the improved Euler method requires more computation compared with the Euler method, it has a higher accuracy with a local error of $O(\Delta t^3)$. (3) The $m$th-order Runge–Kutta method can be written as

$$S(f, u_n, \Delta t_n) = \Delta t_n \sum_{i=1}^{m} \lambda_i K_i.$$  

(18)

For the 3rd-order RK method, the number of stages $m = 3$, the coefficients $\lambda_1 = \lambda_3 = \frac{1}{6}$ and $\lambda_2 = \frac{2}{3}$, and the update rule $K_1 = f(u_n), K_2 = f(u_n + \frac{\Delta t_n}{2} K_1)$ and $K_3 = f(u_n - \Delta t_n K_1 + 2\Delta t_n K_2)$. For the 4th-order RK method, $m = 4$, $\lambda_1 = \lambda_4 = \frac{1}{6}$ and $\lambda_2 = \lambda_3 = \frac{1}{4}$. The update rule $K_1 = f(u_n), K_2 = f(u_n + \frac{\Delta t_n}{2} K_1), K_3 = f(u_n + \frac{\Delta t_n}{2} K_1 + \frac{\Delta t_n}{2} K_2)$ and $K_4 = f(u_n + \Delta t_n K_3)$. Runge–Kutta methods, especially the 4th-order Runge–Kutta method, are widely used
Figure 8. Neural network structure in NeurVec. NeurVec consists of two linear transformations layers (yellow) and one nonlinear activation function layer (red).

in engineering and natural sciences. The Euler method and improved Euler method can also be seen as special Runge–Kutta methods. When using the larger order $m$ in Eq. (18), we need to compute iteratively a series of $K_i$, $i = 1, 2, ..., m$, increasing the computation cost for each step. When used to perform large-scale simulation, the higher-order Runge–Kutta methods generally take a longer time to simulate, as shown in Table X.

Implementation details of NeurVec. We use a fully connected neural network to model NeurVec in Eq. (4). The fully connected feed-forward neural network is the composition of $L$ nonlinear functions:

$$
\phi(x; \theta) := W_a \circ h_L \circ h_{L-1} \circ \cdots \circ h_1(x),
$$  

where $h_\ell(x) = \sigma(W_\ell x + b_\ell)$ with $W_\ell \in \mathbb{R}^{N_i \times N_{i-1}}$, $b_\ell \in \mathbb{R}^{N_i}$ for $\ell = 1, ..., L$, $W_a \in \mathbb{R}^{d \times N_L}$, $x \in \mathbb{R}^{d \times N}$, $\sigma$ is a nonlinear activation function, for example, a rectified linear unit (ReLU) $\sigma(x) = \max\{x, 0\}$ or hyperbolic tangent function $\tanh(x)$, $d$ is the dimension of the state, and $N$ is the batch size. Each $h_\ell$ is referred to as a hidden layer, where $N_i$ is the width of the $\ell$th layer. In this formulation, $\theta := \{W_a, W_\ell, b_\ell : 1 \leq \ell \leq L\}$ denotes the set of all parameters in $\phi$, which uniquely determines the underlying neural network. In our implementation (Fig. 8), the feed-forward neural network is of one hidden layer ($L = 1$) with the width $N_1 = 1024$. The activation function used is the rational activation function $^{56}$ defined by

$$
a \mathbf{x}^3 + a_2 \mathbf{x}^2 + a_1 \mathbf{x}^1 + a_0
\overline{b_2 \mathbf{x}^2 + b_1 \mathbf{x}^1 + b_0},
$$  

where $a_i, 0 \leq i \leq 3$ and $b_i, 0 \leq i \leq 2$ are initialized by constants $a_0 = 0.0218$, $a_1 = 0.5000$, $a_2 = 0.5957$, $a_3 = 1.1915$, $b_0 = 1.0000$, $b_1 = 0.0000$, $b_2 = 2.3830$, respectively. The parameters in $W_a$ and $W_\ell$ are initialized from $\mathbb{U}\left[-1/\sqrt{N_0}, 1/\sqrt{N_0}\right]$ and $\mathbb{U}\left[-1/\sqrt{N_1}, 1/\sqrt{N_1}\right]$, respectively. We optimize the $\phi$ for 500 epochs with the Adam optimizer. Moreover, we use the mean square error as the objective function in Eq. (5), and we set the initial learning rate to $1e^{-3}$.

Data and code availability

The synthesized data for the high-dimensional problem and chaotic dynamical systems and source codes for training and testing results are available at the online data warehouse: https://github.com/dedekinds/NeurVec. The source codes are released under MIT license.

References

1. Böttcher, L., Antulov-Fantulin, N. & Asikis, T. AI Pontryagin or how artificial neural networks learn to control dynamical systems. Nat. Commun. 13, 1–9 (2022).
2. Stuart, A. & Humphries, A. R. Dynamical systems and numerical analysis, vol. 2 (Cambridge University Press, 1998).
3. Harlim, J., Jiang, S. W., Liang, S. & Yang, H. Machine learning for prediction with missing dynamics. J. Comput. Phys. 428, 109922 (2021).
4. Kou-Giesbrecht, S. & Menge, D. Nitrogen-fixing trees could exacerbate climate change under elevated nitrogen deposition. Nat. Commun. 10, 1–8 (2019).
5. Benn, D., Fowler, A. C., Hewitt, I. & Sevestre, H. A general theory of glacier surges. J. Glaciol. 65, 701–716 (2019).
6. Owoyele, O. & Pal, P. Chemnode: A neural ordinary differential equations framework for efficient chemical kinetic solvers. *Energy AI* **7**, 100118 (2022).

7. Zhang, H., Linford, J. C., Sandu, A. & Sander, R. Chemical mechanism solvers in air quality models. *Atmosphere* **2**, 510–532 (2011).

8. Delpini, D. *et al.* Evolution of controllability in interbank networks. *Sci. Reports* **3**, 1–5 (2013).

9. Gholami, A. & Sun, X. A. The impact of damping in second-order dynamical systems with applications to power grid stability. *SIAM J. on Appl. Dyn. Syst.* **21**, 405–437 (2022).

10. Schäfer, B., Witthaut, D., Timme, M. & Latora, V. Dynamically induced cascading failures in power grids. *Nat. Commun.* **9**, 1–13 (2018).

11. Saberi, M. *et al.* A simple contagion process describes spreading of traffic jams in urban networks. *Nat. Commun.* **11**, 1–9 (2020).

12. Fan, C., Jiang, X. & Mostafavi, A. A network percolation-based contagion model of flood propagation and recession in urban road networks. *Sci. Reports* **10**, 1–12 (2020).

13. Aulin, L., Liakopoulos, A., van der Graaf, P. H., Rozen, D. E. & van Hasselt, J. Design principles of collateral sensitivity-based dosing strategies. *Nat. Commun.* **12**, 1–14 (2021).

14. Butner, J. D. *et al.* A mathematical model for the quantification of a patient’s sensitivity to checkpoint inhibitors and long-term tumour burden. *Nat. Biomed. Eng.* **5**, 297–308 (2021).

15. Wicha, S. G., Chen, C., Clewe, O. & Simonsson, U. S. A general pharmacodynamic interaction model identifies perpetrators and victims in drug interactions. *Nat. communications* **8**, 1–11 (2017).

16. Butcher, J. C. *Numerical methods for ordinary differential equations* (John Wiley and Sons, 2016).

17. Ames, W. F. *Numerical methods for partial differential equations* (Academic press, 2014).

18. Shampine, L. F. *Numerical solution of ordinary differential equations* (Routledge, 2018).

19. Figueiras, E., Olivieri, D., Paredes, A. & Michinel, H. QMBlender: Particle-based visualization of 3D quantum wave function dynamics. *J. Comput. Sci.* **35**, 44–56 (2019).

20. Xi, R. *et al.* Survey on smoothed particle hydrodynamics and the particle systems. *IEEE Access* **8**, 3087–3105 (2019).

21. Zhang, Z., Zhang, Y., Li, Y. & Liang, X. Cumuliform cloud animation control based on natural images. In *2020 International Conference on Virtual Reality and Visualization (ICVRV)*, 218–224 (IEEE, 2020).

22. Scher, S. & Messori, G. Ensemble methods for neural network-based weather forecasts. *J. Adv. Model. Earth Syst.* **13** (2021).

23. Hu, B., Guo, H., Zhou, P. & Shi, Z.-L. Characteristics of SARS-CoV-2 and COVID-19. *Nat. Rev. Microbiol.* **19**, 141–154 (2021).

24. Bousquet, A., Conrad, W. H., Sadat, S. O., Vardanyan, N. & Hong, Y. Deep learning forecasting using time-varying parameters of the SIRD model for Covid-19. *Sci. Reports* **12**, 1–13 (2022).

25. Beira, M. J. & Sebastião, P. J. A differential equations model-fitting analysis of COVID-19 epidemiological data to explain multi-wave dynamics. *Sci. Reports* **11**, 1–13 (2021).

26. Choi, W. & Shim, E. Optimal strategies for social distancing and testing to control COVID-19. *J. Theor. Biol.* **512**, 110568 (2021).

27. Hsiang, S. *et al.* The effect of large-scale anti-contagion policies on the COVID-19 pandemic. *Nature* **584**, 262–267 (2020).

28. Tregoning, J. S., Flight, K. E., Higham, S. L., Wang, Z. & Pierce, B. F. Progress of the COVID-19 vaccine effort: viruses, vaccines and variants versus efficacy, effectiveness and escape. *Nat. Rev. Immunol.* **21**, 626–636 (2021).

29. Yuan, F., Zhang, L., Xia, X., Huang, Q. & Li, X. A wave-shaped deep neural network for smoke density estimation. *IEEE Transactions on Image Process.* **29**, 2301–2313 (2019).

30. Tumanov, E., Korobchenko, D. & Chentanez, N. Data-driven particle-based liquid simulation with deep learning utilizing sub-pixel convolution. *Proc. ACM on Comput. Graph. Interact. Tech.* **4**, 1–16 (2021).
31. Kolb, A., Latta, L. & Rezk-Salama, C. Hardware-based simulation and collision detection for large particle systems. In Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware, 123–131 (2004).

32. Luo, H. & Wu, Y. Using virtual reality technology to construct computer-aided animation material development. Comput. Des. & Appl. 19, 155–166 (2022).

33. Bellprat, O., Guemas, V., Doblas-Reyes, F. & Donat, M. G. Towards reliable extreme weather and climate event attribution. Nat. Commun. 10, 1–7 (2019).

34. Touma, D., Stevenson, S., Lehner, F. & Coats, S. Human-driven greenhouse gas and aerosol emissions cause distinct regional impacts on extreme fire weather. Nat. Commun. 12, 1–8 (2021).

35. Palmer, T. & Zanna, L. Singular vectors, predictability and ensemble forecasting for weather and climate. J. Phys. A: Math. Theor. 46, 254018 (2013).

36. Wu, H. & Levinson, D. The ensemble approach to forecasting: a review and synthesis. Transp. Res. Part C: Emerg. Technol. 132, 103357 (2021).

37. Popov, A. A., Mou, C., Sandu, A. & Iliescu, T. A multifidelity ensemble kalman filter with reduced order control variates. SIAM J. on Sci. Comput. 43, A1134–A1162 (2021).

38. Brodtkorb, A. R., Hagen, T. R. & Sætra, M. L. Graphics processing unit (GPU) programming strategies and trends in GPU computing. J. Parallel Distributed Comput. 73, 4–13 (2013).

39. Jouppi, N. P. et al. In-datacenter performance analysis of a tensor processing unit. In Proceedings of the 44th international symposium on computer architecture, 1–12 (2017).

40. Tan, T. & Cao, G. FastVA: deep learning video analytics through edge processing and NPU in mobile. In IEEE INFOCOM 2020-IEEE Conference on Computer Communications, 1947–1956 (IEEE, 2020).

41. Liao, Y. & Roberts, D. B. A high-performance and low-power 32-bit multiply-accumulate unit with single-instruction-multiple-data (SIMD) feature. IEEE J. Solid-State Circuits 37, 926–931 (2002).

42. Fehlberg, E. Low-order classical Runge–Kutta formulas with stepsize control and their application to some heat transfer problems, vol. 315 (National Aeronautics and Space Administration, 1969).

43. Chen, R. T., Rubanova, Y., Bettencourt, J. & Duvenaud, D. K. Neural ordinary differential equations. Adv. Neural Inf. Process. Syst. 31 (2018).

44. Liang, S., Huang, Z. & Zhang, H. Stiffness-aware neural network for learning Hamiltonian systems. In International Conference on Learning Representations (2022).

45. Li, Z. et al. Fourier neural operator for parametric partial differential equations. In International Conference on Learning Representations (2021).

46. Lu, L., Jin, P., Pang, G., Zhang, Z. & Karniadakis, G. E. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. Nat. Mach. Intell. 3, 218–229 (2021).

47. Choudhary, A. et al. Physics-enhanced neural networks learn order and chaos. Phys. Rev. E 101, 062207 (2020).

48. Han, C.-D., Glaz, B., Haile, M. & Lai, Y.-C. Adaptable Hamiltonian neural networks. Phys. Rev. Res. 3, 023156 (2021).

49. Greydanus, S., Dzamba, M. & Yosinski, J. Hamiltonian neural networks. Adv. Neural Inf. Process. Syst. 32 (2019).

50. Huang, W. & Leimkuhler, B. The adaptive Verlet method. SIAM J. on Sci. Comput. 18, 239–256 (1997).

51. Wang, X., Han, Y., Xue, W., Yang, G. & Zhang, G. J. Stable climate simulations using a realistic general circulation model with neural network parameterizations for atmospheric moist physics and radiation processes. Geosci. Model. Dev. 15, 3923–3940 (2022).

52. Lu, J., Shen, Z., Yang, H. & Zhang, S. Deep network approximation for smooth functions. SIAM J. on Math. Analysis 53, 5465–5506 (2021).

53. Shen, Z., Yang, H. & Zhang, S. Deep Network with Approximation Error Being Reciprocal of Width to Power of Square Root of Depth. Neural Comput. (2021).

54. Paszke, A. et al. Pytorch: An imperative style, high-performance deep learning library. Adv. neural information processing systems 32 (2019).
55. Abadi, M. et al. Tensorflow: A system for large-scale machine learning. In 12th USENIX symposium on operating systems design and implementation (OSDI 16), 265–283 (2016).

56. Boullé, N., Nakatsukasa, Y. & Townsend, A. Rational neural networks. Adv. Neural Inf. Process. Syst. 33, 14243–14253 (2020).

57. Chen, Z., Zhang, J., Arjovsky, M. & Bottou, L. Symplectic recurrent neural networks. In International Conference on Learning Representations (2020).

58. Feit, M. & Fleck Jr, J. Wave packet dynamics and chaos in the hénon–heiles system. The J. Chem. Phys. 80, 2578–2584 (1984).

59. DiPietro, D., Xiong, S. & Zhu, B. Sparse symplectically integrated neural networks. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. & Lin, H. (eds.) Advances in Neural Information Processing Systems, vol. 33, 6074–6085 (Curran Associates, Inc., 2020).

60. Breitenberger, E. & Mueller, R. D. The elastic pendulum: a nonlinear paradigm. J. Math. Phys. 22, 1196–1210 (1981).

61. Lopes, A. M. & Tenreiro Machado, J. Dynamics of the N-link pendulum: a fractional perspective. Int. J. Control. 90, 1192–1200 (2017).

62. Atkinson, K., Han, W. & Stewart, D. E. Numerical solution of ordinary differential equations (John Wiley and Sons, 2011).

63. Du, S. S., Zhai, X., Poczos, B. & Singh, A. Gradient descent provably optimizes over-parameterized neural networks. In International Conference on Learning Representations (2019).

64. Jacot, A., Gabriel, F. & Hongler, C. Neural tangent kernel: Convergence and generalization in neural networks. Adv. Neural Inf. Process. Syst. 31 (2018).

65. Chizat, L., Oyallon, E. & Bach, F. On lazy training in differentiable programming. Adv. Neural Inf. Process. Syst. 32 (2019).

66. Hu, J., Shen, L. & Sun, G. Squeeze-and-excitation networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 7132–7141 (2018).

67. Huang, Z., Liang, S., Liang, M. & Yang, H. Dianet: Dense-and-implicit attention network. In Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34, 4206–4214 (2020).

68. Liang, S., Huang, Z., Liang, M. & Yang, H. Instance enhancement batch normalization: An adaptive regulator of batch noise. In Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34, 4819–4827 (2020).

69. He, W., Huang, Z., Liang, M., Liang, S. & Yang, H. Blending pruning criteria for convolutional neural networks. In International Conference on Artificial Neural Networks, 3–15 (Springer, 2021).

70. Liu, H., Simonyan, K. & Yang, Y. Darts: Differentiable architecture search. In International Conference on Learning Representations (2018).

71. Huang, Z., Shao, W., Wang, X., Lin, L. & Luo, P. Rethinking the pruning criteria for convolutional neural network. Adv. Neural Inf. Process. Syst. 34, 16305–16318 (2021).

72. Kenton, J. D. M.-W. C. & Toutanova, L. K. Bert: Pre-training of deep bidirectional transformers for language understanding. In Proceedings of NAACL-HLT, 4171–4186 (2019).

73. Wu, Y. et al. Large scale incremental learning. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 374–382 (2019).

74. Süli, E. & Mayers, D. F. An introduction to numerical analysis (Cambridge University Press, 2003).

75. Trench, W. F. Elementary differential equations with boundary value problems (Brooks Cole Thomson Learning, 2013).

76. Arts, L., van den Broek, E. et al. The fast continuous wavelet transformation (fCWT) for real-time, high-quality, noise-resistant time–frequency analysis. Nat. Comput. Sci. 1–12 (2022).
Supplementary

Proof of the Theorem

Proof. We denote $E_n := \hat{u}_{kn} - u_{kn}$. Then by triangle inequality and Proposition 0.1, we have

$$|u_{kp} - u(T)| \leq |u_{kp} - u(T)| + |\hat{u}_{kp} - u_{kp}| = E_p + \frac{M \exp(2TL)}{2L} \Delta t. \quad (21)$$

Next we estimate the error $E_p$. We have

$$\hat{u}_{k(n+1)} - u_{k(n+1)} = \hat{u}_{kn} + f(\hat{u}_{kn})(k\Delta t) + \text{NeurVec}(\hat{u}_{kn}; \theta) - u_{k(n+1)}$$

$$= u_{kn} - u_{kn} + (f(\hat{u}_{kn}) - f(u_{kn}))(k\Delta t) + \text{NeurVec}(\hat{u}_{kn}; \theta) - \text{NeurVec}(u_{kn}; \theta) - (k\Delta t)V_n.$$  

Then using assumption (1),

$$|\hat{u}_{k(n+1)} - u_{k(n+1)}| \leq |\hat{u}_{kn} - u_{kn}| + L|\hat{u}_{kn} - u_{kn}|(k\Delta t) + k\Delta tLNV|\hat{u}_{kn} - u_{kn}| + (k\Delta t)|V_n|$$

$$= (1 + k\Delta tL + k\Delta tLNV)|\hat{u}_{kn} - u_{kn}| + (k\Delta t)|V_n|.$$  

we denote the constant $(1 + k\Delta tL + k\Delta tLNV)$ as $w$. We rewrite the above inequality as $|E_{n+1}| \leq w|E_n| + (k\Delta t)|V_n|.$

Then

$$|E_{n+1}| \leq w|E_n| + (k\Delta t)|V_n|$$

$$\leq w(|E_{n-1}| + (k\Delta t)|V_{n-1}|) + (k\Delta t)|V_n| = w^2|E_{n-1}| + w(k\Delta t)|V_{n-1}| + (k\Delta t)|V_n|$$

$$\leq \ldots$$

$$\leq w^{n+1}|E_0| + (k\Delta t)\sum_{i=0}^{n} w^i|V_{n-i}| = (k\Delta t)\sum_{i=0}^{n} w^i|V_{n-i}|,$$

where $E_0 = 0$ as $E_0 = \hat{u}_0 - u_0 = c_0 - c_0 = 0$. By the Cauchy inequality,

$$|E_p| \leq (k\Delta t)\left(\sum_{i=0}^{p-1} w^{2i}\right)\frac{1}{2} \left(\sum_{i=0}^{p-1} |V_{p-1-i}|^2\right)\frac{1}{2} = (k\Delta t)\left(\frac{w^{2p} - 1}{w^2 - 1}\right)\left(pL\right)^\frac{1}{2}.$$  

Note that $\frac{w^{2p} - 1}{w^2 - 1} \leq \frac{(1 + k\Delta t + k\Delta tLNV)^{2p}}{k\Delta tL + k\Delta tLNV} \leq \frac{\exp(2k\Delta t(L + LNV))}{k\Delta tL + k\Delta tLNV} = \exp(2T(L + LNV))$. We obtain the bound by

$$|E_p| \leq (k\Delta t)\frac{\exp(T(L + LNV))}{\sqrt{k\Delta t(L + LNV)}}\left(pL\right)^\frac{1}{2} = \sqrt{T}\frac{\exp(T(L + LNV))}{\sqrt{L + LNV}}\left(L\right)^\frac{1}{2}.$$  

Combining (21), we end our proof.
Details of the acceleration.

Runtime
We presented the normalized runtime results of different configurations in the main text. Here we provide details about the implementation and the exact runtime values.

We divide the testing dataset of each problem (Table 2) equally into 70 batches. Then, we simulate each batch sequentially on a single GeForce RTX 3080 GPU and record their inference time at each run. In order to mitigate the GPU from overheating, a 10-second pause is executed between every two runs. The mean clock time and its standard derivation (std) are reported in Table 3.

| Problems       | Method              | Step size | Time-mean (sec.) | Time-std |
|----------------|---------------------|-----------|------------------|----------|
| Spring-chain   | Euler               | 2e-1      | 0.099            | 0.002    |
| Spring-chain   | Euler+NeurVec       | 2e-1      | 0.124            | 0.011    |
| Spring-chain   | Improved Euler      | 2e-1      | 0.172            | 0.002    |
| Spring-chain   | Improved Euler+NeurVec | 2e-1    | 0.237            | 0.030    |
| Spring-chain   | RK3                 | 2e-1      | 0.260            | 0.002    |
| Spring-chain   | RK3+NeurVec         | 2e-1      | 0.348            | 0.028    |
| Spring-chain   | RK4                 | 2e-1      | 0.359            | 0.002    |
| Spring-chain   | RK4+NeurVec         | 2e-1      | 0.463            | 0.040    |
| Hénon-Heiles   | RK4                 | 5e-1      | 0.009            | 0.001    |
| Hénon-Heiles   | RK4+NeurVec         | 5e-1      | 0.015            | 0.001    |
| 2-link pendulum| RK4                 | 1e-1      | 0.269            | 0.004    |
| 2-link pendulum| RK4+NeurVec         | 1e-1      | 0.441            | 0.061    |
| Elastic pendulum| RK4              | 1e-1      | 0.056            | 0.001    |
| Elastic pendulum| RK4+NeurVec       | 1e-1      | 0.100            | 0.059    |

Table 3. Comparison of mean runtime and its standard deviation over 70 batches of simulation.
Statistical test
We statistically validate the acceleration performance of NeurVec. We collected the runtime results of solver with fine step size and NeurVec with coarse step size as in Table 3; each of them contains 70 samples. In the t-test, the null hypothesis is that the runtimes of solver with fine step size and NeurVec with coarse step size are identical. We use $P_1$ to denote the $P$-value of the two-sided t-test and $P_2$ to denote the $P$-value of the Welch’s t-test. These two t-tests have the same statistics but for different situations. We use the two-sided t-test if the the pair of the runtimes have the same variance. Otherwise, we use Welch’s t-test. In Table 4, all the $P_1$ and $P_2$ are much smaller than $1e-3$, indicating that the null hypothesis is rejected and the pair of runtime samples are different. In other words, NeurVec can accelerate the simulation with statistical significance.

| Problem      | Method | Setting1 | Setting2 | Statistics | $P_1$ ($P_1 \ll 1e^{-3}$) | $P_2$ ($P_2 \ll 1e^{-3}$) |
|--------------|--------|----------|----------|------------|-------------------------|-------------------------|
| Spring-chain | Euler  | 1e-3     | 2e-1     | 461.97     | 5.59e-222 ✓             | 3.55e-122 ✓             |
| Spring-chain | Euler  | 1e-3     | 2e-1+NeurVec | 461.07   | 7.30e-222 ✓             | 2.44e-122 ✓             |
| Spring-chain | Imp-Euler | 1e-3 | 2e-1     | 125.34     | 4.87e-144 ✓             | 3.88e-83 ✓             |
| Spring-chain | Imp-Euler | 1e-3 | 2e-1+NeurVec | 125.11   | 6.24e-144 ✓             | 4.22e-83 ✓             |
| Spring-chain | RK3    | 1e-3     | 2e-1     | 1252.16    | 1.01e-281 ✓             | 4.73e-152 ✓             |
| Spring-chain | RK3    | 1e-3     | 2e-1+NeurVec | 1246.92  | 1.80e-281 ✓             | 2.41e-153 ✓             |
| Spring-chain | RK4    | 1e-3     | 2e-1     | 291.81     | 1.78e-194 ✓             | 2.08e-108 ✓             |
| Spring-chain | RK4    | 1e-3     | 2e-1+NeurVec | 291.32   | 2.25e-194 ✓             | 2.00e-108 ✓             |
| Hénon-Heiles | RK4    | 1e-3     | 5e-1     | 557.53     | 3.06e-233 ✓             | 7.91e-128 ✓             |
| Hénon-Heiles | RK4    | 1e-3     | 5e-1+NeurVec | 556.97   | 3.51e-233 ✓             | 8.48e-128 ✓             |
| 2-pendulum   | RK4    | 1e-3     | 1e-1     | 888.86     | 3.46e-261 ✓             | 8.85e-142 ✓             |
| 2-pendulum   | RK4    | 1e-3     | 1e-1+NeurVec | 859.68   | 3.46e-259 ✓             | 2.30e-154 ✓             |
| Elastic pendulum | RK4 | 1e-3     | 1e-1     | 669.28     | 3.46e-244 ✓             | 2.63e-133 ✓             |
| Elastic pendulum | RK4 | 1e-3     | 1e-1+NeurVec | 550.22   | 1.88e-232 ✓             | 7.35e-208 ✓             |

Table 4. The statistical test for different pair run time results. $P_1$ is the $P$-value of the two-sided t-test and $P_2$ is the $P$-value of the Welch’s t-test.
Summary of the initialization of different systems.

| Task               | Variable | Dim | Type              | Range         |
|--------------------|----------|-----|------------------|---------------|
| Spring-chain       | p        | 20  | Uniform random   | $[-2.5, 2.5]^{20}$ |
| Spring-chain       | q        | 20  | Uniform random   | $[-2.5, 2.5]^{20}$ |
| Hénon–Heiles       | $q_x$    | 1   | Uniform random   | [-1,1]        |
| Hénon–Heiles       | $q_y$    | 1   | Uniform random   | [-0.5,1]      |
| Hénon–Heiles       | $p_x$    | 1   | Uniform random   | [-1,1]        |
| Hénon–Heiles       | $p_y$    | 1   | Uniform random   | [-1,1]        |
| Elastic pendulum   | $\theta$ | 1   | Uniform random   | [0, $\pi/8$]  |
| Elastic pendulum   | $r$      | 1   | Constant         | 10            |
| Elastic pendulum   | $\dot{\theta}$ | 1 | Constant         | 0             |
| Elastic pendulum   | $\dot{r}$ | 1 | Constant         | 0             |
| Elastic pendulum   | $l_0$    | 1   | Constant         | 10            |
| Elastic pendulum   | $g$      | 1   | Constant         | 9.8           |
| Elastic pendulum   | $k$      | 1   | Constant         | 40            |
| Elastic pendulum   | $m$      | 1   | Constant         | 1             |
| 2-pendulum         | $\theta$ | 2   | Uniform random   | [0, $\pi/8]^2$ |
| 2-pendulum         | $\dot{\theta}$ | 2 | Constant         | 0             |
| 1-pendulum         | $\theta$ | 1   | Uniform random   | [0, $\pi/2$]  |
| 1-pendulum         | $\dot{\theta}$ | 1 | Uniform random   | [0,0.5]       |
| 1&2-pendulum       | $m$      | 1   | Constant         | 1             |
| 1&2-pendulum       | $g$      | 1   | Constant         | 9.8           |

Table 5. The values of $m$ and $k$ in spring-chain systems: $m$ and $k$ are obtained by random and independent sampling, and their values are consistent with those of SRNN.

Table 6. Initial state of different systems. “Uniform random” means that the variables are sampled with uniform distribution of given range. “Constant” means the variable is initialized as a constant. In the Hénon–Heiles system, after the initialization, the data that do not satisfy the energy $\mathcal{H}(q_x, q_y, p_x, p_y) \in [\frac{1}{12}, \frac{1}{6}]$ will be removed.
Simulations on Hénon–Heiles system.

Fig. 9 displays more examples of simulations on Hénon–Heiles system, an implement to Fig. 4e.

Figure 9. More performance comparison on the Hénon–Heiles system. We provide five additional examples of trajectories projected on the coordinates \((q_x, q_y, p_x)\). NeurVec \(\Delta^N t = 5e-1\) produces the most orbits similar to \(\Delta^F t = 1e-3\).
Time series histogram.

Fig. 10 displays the time series histogram of $\theta$ and $\dot{\theta}$ in the elastic pendulum (9), and $q_x$ and $p_x$ in the Hénon–Heiles system (7). The statistical difference of $\theta$ and $\dot{\theta}$ among fine step size, coarse step size and NeurVec with coarse step size is not large. Yet that of $q_x$ and $p_x$ is large.

**Figure 10. Additional experiments for time series histogram.** We visualize the time series histogram of the test set for variables in the elastic pendulum and Hénon–Heiles system. The color represents the number count (the lighter color and the larger frequency). (a), $\theta$ and $\dot{\theta}$ in the elastic pendulum (9). Unlike the results about $r$ and $\dot{r}$ in Fig. 6a, the $\theta$ and $\dot{\theta}$ generated by different step sizes have similar trends, although there are minor differences among them. However, in (b) $q_x$ and $p_x$ in the Hénon–Heiles system (7), there is a consistent observation for the solutions under different step sizes with Fig. 6b.