Simulating Molecular Dynamics with Large Timesteps using Recurrent Neural Networks

JCS Kadupitiya,[1, *] Geoffrey C. Fox,[1, †] and Vikram Jadhao[1, ‡]

[1]*Intelligent Systems Engineering, Indiana University, Bloomington IN 47408, USA*

Molecular dynamics simulations rely on numerical integrators such as Verlet to solve the Newton's equations of motion. Using a sufficiently small timestep to avoid discretization errors, Verlet integrators generate a trajectory of particle positions as solutions to the equations of motions. We introduce an integrator based on recurrent neural networks that is trained on trajectories generated using Verlet integrator and learns to propagate the dynamics of particles with timestep up to 4000 times larger compared to the Verlet timestep. We demonstrate significant net speedup of up to 32000 for few-particle (1 - 16) 3D systems and over a variety of force fields.

Introduction Molecular dynamics (MD) simulations are powerful tools for investigating the microscopic origins of a wide range of material phenomena [1–6]. MD method solves the Newton's equation of motion for a system of many particles and presents the solution in the form of a trajectory of particles (sequence of configurations, e.g., positions) at discrete time values. At the heart of the method is the integrator that propagates the configuration of the system one small timestep at a time. Verlet integrators are a popular choice for this operation [7]. For example, the ordinary Verlet integrator:

$$\vec{x}(t + \Delta) = 2\vec{x}(t) - \vec{x}(t - \Delta) + \Delta^2 \frac{\vec{f}}{m} + O(\Delta^4) \quad (1)$$

updates the current position $\vec{x}(t)$ at time t to the future position $\vec{x}(t + \Delta)$ after timestep Δ using the previous position $\vec{x}(t - \Delta)$ and the force $\vec{f}$ at time t that is assumed to be an analytically known function of $\vec{x}$. Here we have adopted the notation for a single particle of mass m; generalization to many particles is straightforward. With $\vec{x}(t + \Delta)$ as the current position, and $x(t)$ as the previous, the propagation is continued further. The term $O(\Delta t^4)$ denotes the incurred error.

Equation 1 suggests that the Verlet integrator ($\mathcal{V}$) requires a sequence of 2 position coordinates $(\vec{x}_t, \vec{x}_{t-\Delta t})$ in order to evolve the position forward by Δ (we will call this the Verlet timestep). However, the integrator also uses the force term $\vec{f}$ and mass m to process the position update. It is possible to infer $\vec{f}$ using the information encoded in the stream of position data such that the time evolution can be done with just a sequence of positions as input. We illustrate this with a 1-dimensional example of a particle experiencing simple harmonic motion governed by the force $f = -kx$. It can be shown that the particle position can be accurately evolved to $t + \Delta t$ via the function $\mathcal{V}(x_t, x_{t-\Delta t}, x_{t-2\Delta t}) = x_{t-\Delta}^{-1} \left(x_t^2 - x_{t-\Delta}^2 + x_t x_{t-2\Delta} \right)$ that uses a sequence of 3 positions (the information about f, m, and Δ is hidden in the input sequence). This idea can be extended to a general system where one could simulate the dynamics (time evolution) via the operator function $\mathcal{V}$: $\vec{x}(t + \Delta) = \mathcal{V}(\vec{x}_t, \vec{x}_{t-\Delta}, \ldots \vec{x}_{t-n_V \Delta})$ that takes a se-
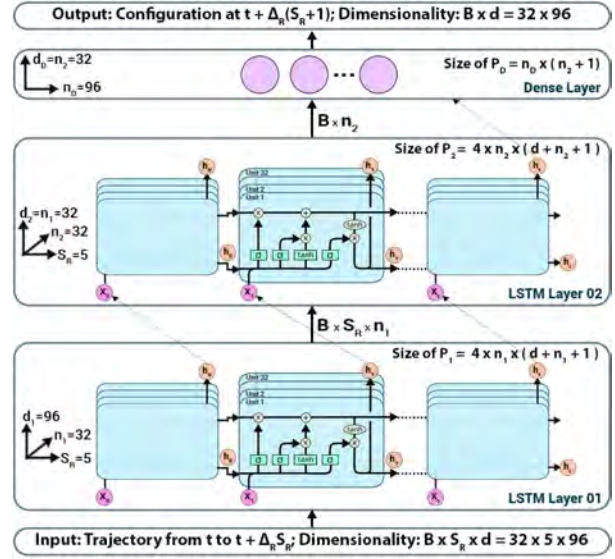

FIG. 1. $\mathcal{R}$ integrator based on RNN propagates the single timestep (Δ_R) evolution of the N particle system characterized by input features of size d. Input system and LSTM parameters are shown for $N = 16$. Symbols are defined in the main text.

quence of n_V positions and propagates the particle Δ forward in time. However, it becomes challenging to derive $\mathcal{V}$ in a simple functional form like above for general systems with many particles, as such one defaults to Equation 1 for implementing the time evolution. We show that deep learning can be used to derive operators like $\mathcal{V}$ for different force fields, and we demonstrate that these operators can simulate the dynamics using a timestep that is orders of magnitude larger than Δ.

The use of deep learning in sequence processing and time series prediction problems has been well studied by industry for different applications including voice recognition and translation, pattern recognition in stock market data, and ride-hailing [8]. Recurrent neural networks (RNN) are established DL tools in these applications. RNNs process input sequence data and maintain a vector $\vec{h}_t$ known as the "hidden state" for each recurrent cell to model the temporal behavior of sequences through directed cyclic connections between its cells. $\vec{h}_t$ is updated
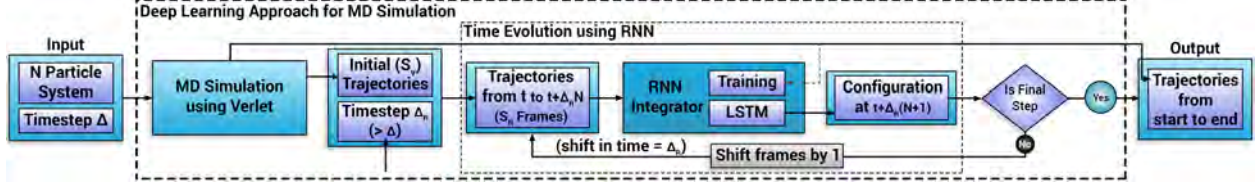
FIG. 2. Overview of the deep learning approach based on recurrent neural networks for molecular dynamics simulations.

by applying a function f to the previous hidden state ($\vec{h}_{t-1}$) and the current input ($\vec{x}_t$). The cells are arranged in a fashion where they will fire when the right sequence is fed. A common choice for f is the Long Short Term Memory networks (LSTM). An LSTM unit contains a cell which is the memory of the unit, and three gates (input gate, output gate, and forget gate) which regulate the flow of information into and out of the cells. This gated architecture allows the LSTM to remember longer dependencies of the sequences fed into the network and deal with the exploding and vanishing gradient problems encountered while training RNNs (see SI).

We introduce an operator $\mathscr{R}$ based on RNN derived using LSTMs (Figure 1) that employs a sequence S_R of current and past positions up to time t to predict the future position at time $t+\Delta_R$: $\vec{x}(t + \Delta_R) = \mathscr{R}\left[\vec{x}_t, \vec{x}_{t-\Delta_R}, \ldots, \vec{x}_{t-S_R\Delta_R}\right]$ where Δ_R is the timestep associated with the RNN integrator. $\mathscr{R}$ can be written as $\mathscr{R}\{x\} = \mathscr{D}[\mathscr{L}_2[\mathscr{L}_1[\{x\}]]]$, where $\mathscr{D}$, $\mathscr{L}_1$, $\mathscr{L}_2$ are operators associated with the dense layer, first LSTM layer, and second LSTM layer of the RNN respectively, and $\{x\}$ is a shorthand for the sequence $\vec{x}_t, \vec{x}_{t-\Delta_R}, \ldots, \vec{x}_{t-S_R\Delta_R}$. These layers are stacked up on each other as explained in Figure 1 such that the output of one (e.g., $\mathscr{L}_1$) becomes the input for another ($\mathscr{L}_2$). Each of these layer operators contains a set of parameters in the form of weights, biases, and activation functions. For example, the first hidden layer $\mathscr{L}_1$ is characterized with weights W, U, and bias b. It takes an input vector $\{x\}$ and outputs hidden state vectors $\{h\} = \mathscr{L}_1[\{x\}, W_{1f}, U_{1f}, b_{1f}, \ldots, W_{1c}, U_1, b_{1c}]$. These hidden states are fed as input to the $\mathscr{L}_2$ operator characterized with its own set of weights and biases. A similar connection is made between $\mathscr{L}_2$ and $\mathscr{D}$. Post training, these layers acquire well-determined values for all the parameters such that an integrator operator $\mathscr{R}$ emerges akin to the $\mathcal{V}$ operator in the illustrative example above, with the more explicit expression:

$$\mathscr{R}[\{x\}] = \mathscr{D}[\mathscr{L}_2[\mathscr{L}_1[\{x\}, \{P_1\}], \{P_2\}], \{P_D\}], \quad (2)$$

where $\{P_1\}, \{P_2\}, \{P_D\}$ are trainable parameters associated with LSTM layer 1, LSTM layer 2, and the dense layer respectively. $\mathscr{R}$ in Equation 2 characterized with up to $100,000$ parameters can be considered as a reformulation of the ordinary Verlet integrator $\mathcal{V}$ (Equation 1).

This computational complexity accounts for its ability to handle much larger timesteps in driving MD simulations as we demonstrate below. Note that while, for the ease of exposition, we showed how RNNs can describe the ordinary Verlet integrator, the derivation can be readily adapted to describe the velocity Verlet integrator (see SI). In what follows, $\mathcal{V}$ denotes the velocity Verlet integrator.

Related Work The key related work focuses on DL to learn differential equations and replicate numerical integrators [9–16]. Generally, these integrators have been derived to operate with the time discretization associated with the baseline integrator timestep dt. Their applications have been demonstrated on relatively simple 1D systems governed by ordinary differential equations. Few studies have probed the DL applications to learn partial differential equations [11]. Research more closely related to ours includes recent approaches to improve the accuracy of integrators at a higher dt [10, 17]. Shen et. al [17] have developed an artificial neural network (ANN) based Euler method for numerical integration with high dt by approximating the local truncation error with deep neural networks to obtain accurate solutions with dt up to 200 times larger than the baseline Euler approach. As applications, they have shown the approach to work on 1D systems and relatively simple 2-body problems. For the 1D problems, a 100 fold increase in the timestep (from $2\times$ to $200\times$) led to an increase in the prediction error by over 3 orders of magnitude. In this paper, we show that RNN-based integrators can limit the rise in the error with increasing timestep ($> 1000\times$ the baseline) to within an order of magnitude, even for the complex 3D systems.

Instead of explicit time evolution of the system with an operator that emulates the numerical time integration, other related work has focused on using DL to learn the Hamiltonian of simple physical systems and respect conservation laws in an unsupervised manner [15, 18–25]. Here, neural networks observe the positions and momenta (input) and produce the Hamiltonian as output, in some cases leveraging the information that the partial derivatives of the output with respect to inputs are the time derivatives of the inputs [18]. Hamiltonian neural networks [18], Lagrangian neural networks [19], and symplectic RNNs [15] are examples of these DL approaches that have been shown to learn the physics of
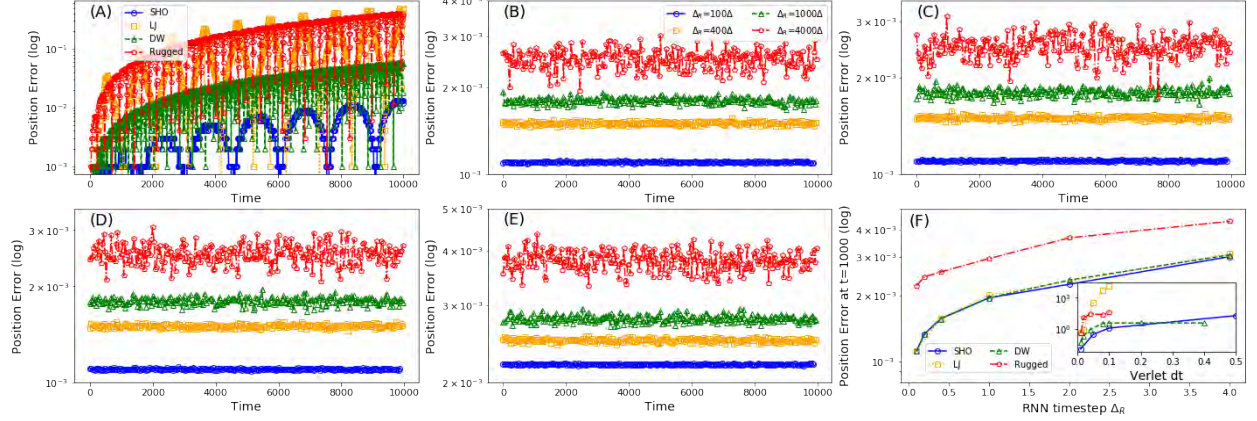
FIG. 3. Errors (log scale) in position updates (trajectory) for 1D simulations. Results are shown for 4 potentials: SHO with $m = 10$, $k = 1$, LJ with $m = 1$, $x_0 = 2$, DW with $m = 1$, $x_0 = -2$ and rugged with $m = 1$, $x_0 = -6$. Results are compared to respective ground truth results obtained using $\mathscr{V}$ with $\Delta = 0.001$. (A) Errors using $\mathscr{V}$ with timestep $dt = 10\Delta$. (B), (C), (D), and (E) show trajectory errors using $\mathscr{R}$ for SHO, LJ, DW, and rugged potentials respectively for $\Delta_R = 100\Delta$ (circles), 400Δ (squares), 1000Δ (triangles), and 4000Δ (pentagons). (F) Error incurred in position updates using $\mathscr{R}$ at $t = 1000$ vs. timestep Δ_R for the 4 potentials; inset shows corresponding error using $\mathscr{V}$ vs. Verlet timestep dt.

simple systems such as spring-mass problems in 1D.

In materials science and engineering research, recent years have seen a surge in the use of machine learning (ML) to enhance the performance of simulations and analyze the data they generate [26–40]. However, to the best of our knowledge, DL-based advancements at the level of re-formulating the integrator driving the MD simulation have not been made. Generally, ML applications in MD simulations of materials have centered on accelerating the sampling of systems with rugged free-energy landscapes [36], generating new configuration updates in latent-space (low-dimensional) variables in ab initio MD simulations of nuclei-electron systems [37], auto-tuning timestep in MD simulations of ions near nanomembranes [38], and classifying assembly landscapes employing MD-generated particle trajectories [39]. Furthermore, DL has been used to derive networks that produce "surrogates" of large-scale simulations, including MD simulations [40–44].

We build on these recent works with the goal towards enabling simulations of complex 3D systems of particles interacting with commonly employed potentials describing soft-matter systems (e.g., Lennard-Jones potential), and develop a DL approach that learns the Hamiltonian of the system and the underlying differential equation from the high-dimensional configuration space (trajectories) to perform the time evolution with large timesteps. We show that RNN with generic LSTM models, with minimal architectural changes, can learn both the dynamical equations and the interaction potentials driving the dynamics of particles. Most importantly, we demonstrate that our integrator performs accurate time evolu-

tion of 1D systems and complex 3D systems with realistic boundary conditions and over a variety of force fields using large timesteps (up to $4000\times$ the baseline). To the best of our knowledge, in terms of number of particles, timesteps for trajectory evolution, and the complexity of the systems probed, our approach fares better than existing ones in metrics such as trajectory errors, energy conservation tracking, and net speedup.

Methods Figure 2 describes the DL approach using $\mathscr{R}$ to evolve the dynamics of an N particle system with timestep Δ_R. $\mathscr{R}$ is trained using particle trajectories generated via $\mathscr{V}$ with small $\Delta = 0.001 < \Delta_R$ such that the Verlet result can be considered as the ground truth. Input system attributes and Δ are fed to $\mathscr{V}$ to simulate the dynamics and obtain $S_V = \Delta_R(S_R - 1)/\Delta$ number of configurations. Out of S_V, S_R number of configurations (frames) separated by Δ_R are distilled to feed the $\mathscr{R}$ operator (S_R is the sequence length). $\mathscr{R}$ predicts the time evolution of the system after Δ_R. Until the end of the simulation, the input sequence to $\mathscr{R}$ is left shifted to discard the oldest time frame, and the latest frame predicted by $\mathscr{R}$ is appended to the right of the sequence. The adjusted input sequence is fed to $\mathscr{R}$ to evolve the system Δ_R further in time.

The training and testing datasets are generated and processed for $N \in (1, 16)$ by changing the parameters characterizing the potential energies and the physical attributes describing the particles. Four different potentials are considered: 1) simple harmonic oscillator (SHO), 2) Lennard-Jones (LJ), 3) double well (DW), and 4) rugged (see SI). In these initial studies, we select datasets generated by sweeping parameters that mainly shift the ini-
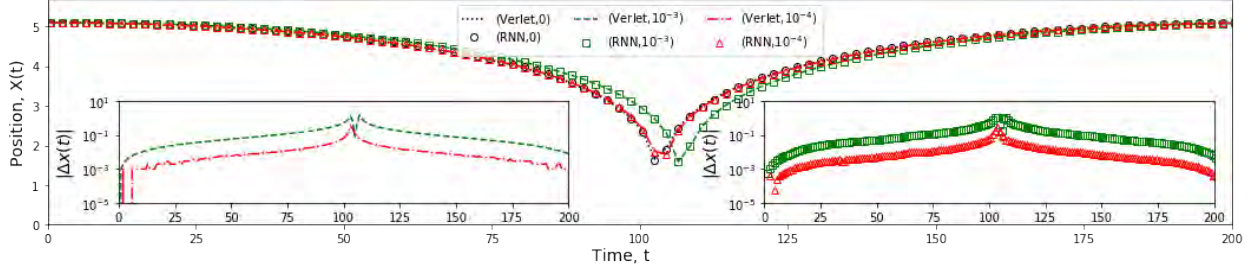
FIG. 4. Lyapunov instability tests in 1D simulations of a particle in an LJ potential with $m = 1, x_0 = 5.1$ using $\mathcal{V}$ with $\Delta = 0.001$ and $\mathcal{R}$ with $\Delta_R = 0.1 = 100\Delta$. Legend indicates (integrator, δp) where δp represents the small momentum shift introduced at the start of trajectory. Lines and markers are the results of simulations driven by $\mathcal{V}$ and $\mathcal{R}$ integrators respectively. Outset shows the trajectory for $\delta p = 0, 10^{-3}, 10^{-4}$. Left and right insets show the absolute difference between trajectories with $\delta p \neq 0$ and $\delta p = 0$ ($\Delta x(t) = |x(t, 0) - x(t, \delta p)|$) as produced by $\mathcal{V}$ and $\mathcal{R}$ respectively.

tial configuration (e.g., x_0 in LJ) or scale the particle attributes (e.g., mass m in LJ), and in some cases change the shape of the potential energy (e.g., k in SHO). $\mathcal{R}$ integrator with LSTM layer 1, LSTM layer 2, and final dense layer (Figure 1) is implemented in TensorFlow for regression of the particle trajectories using n_1, n_2, and n_D number of hidden units respectively (specific values depend on the system). $\mathcal{R}$ takes a $B \times S_R \times d$ dimensional vector as input where B and d refer to batch and feature size respectively. All the parameters $\{P\}$ (weights, biases) describing the layers are trained with an error backpropagation algorithm, implemented via a stochastic gradient descent. Data preparation and preprocessing, feature extraction, and LSTM implementation details are provided in the SI.

Experiments and Empirical Results We now compare the empirical results (e.g., trajectories, energies) obtained from simulations using $\mathcal{R}$ against the baseline velocity Verlet integrator $\mathcal{V}$. Our first experiments are on testing $\mathcal{R}$ to predict the time evolution of single particle systems in 1D. Training datasets of trajectories simulated using $\mathcal{V}$ with $\Delta = 0.001$ up to $t = 200$ are used. We find that the errors in positions (trajectory errors) predicted by $\mathcal{R}$ do not increase over time for t up to 10000, and are $O(10^{-3})$ for all systems studied with $\Delta_R \in (100\Delta, 4000\Delta)$ (Figure 3 B, C, D, E). The errors rise with increase in Δ_R and the complexity of the potential (e.g., higher for rugged than LJ) but remain within an order of magnitude (Figure 3 F). The average error values across different potentials for $\Delta_R = [100\Delta, 400\Delta, 1000\Delta, 4000\Delta]$ are $\approx 0.001, 0.002, 0.003, 0.004$ respectively. On the other hand, the trajectory errors incurred using $\mathcal{V}$ with timestep 10Δ show exponential increase with t and with rising timestep for all potentials considered (Figure 3 A and F). Similarly, velocity v and position x predictions produced by $\mathcal{R}$ exhibit small trajectory errors, and the phase diagram (v vs. x) and the total energy track the

ground truth associated with the respective potential energy (e.g., LJ or SHO) for t up to 10000 and Δ_R up to 4000Δ (SI Figure 8). $\mathcal{R}$ learns the energy conservation feature associated with the dynamics of these systems obeying Newton's equations of motion.

Next set of experiments involve assessing the stability of the solutions (trajectories) predicted by the integrator $\mathcal{R}$. This is typically discussed in terms of Lyapunov instability (LI) which states that close-by trajectories diverge exponentially [45, 46]. LI is present in standard Verlet integrators for many potentials. Trajectories of a particle of $m = 1$ in 1D LJ potential generated using $\mathcal{V}$ are used for these experiments. A small perturbation in the momentum δp is introduced at the start to investigate its effects on long-time evolution of the trajectory. We find that the RNN surrogate inherits the characteristic LI of $\mathcal{V}$ (Figure 4). As δp increased from 10^{-4} to 10^{-3}, the trajectories predicted by $\mathcal{R}$ ($\mathcal{V}$) exhibit similar average divergences from 0.0153 (0.0152) to 0.149 (0.148).

Our final set of experiments probe the extension of the proposed DL approach to $N > 1$ particles. $N = 3, 8, 16$ particle systems with LJ interaction potential are used. $\mathcal{R}$ is trained to learn from trajectories generated using $\mathcal{V}$ ($\Delta = 0.001$) in a cubic box with periodic boundary conditions (PBC) or in a spherical hard-wall confinement with reflective boundaries. We find that $\mathcal{R}$ successfully evolved the positions and velocities of particles up to time $t = 1$ million for all the N-particle systems studied with Δ_R up to 4000Δ. In the interest of brevity, we only discuss the results of experiments on the $N = 16$ system in PBC. The training dataset size for this study involved dynamics up to $t = 2000$. The random initial configuration for testing $\mathcal{R}$ was selected outside the training dataset. We find that the trajectory error (computed as an average over 16 particles) associated with the positions predicted by $\mathcal{R}$ for $\Delta_R = 100\Delta$ to 4000Δ is $O(10^{-3})$ for all times (SI Figure 9), exhibiting a behavior similar to the 1D results (Figure 3). The total energy $E_t \equiv E(t)$

and the energy deviation $\delta E(t) = (E_t - E_0)/E_0$ of the system, extracted from the predicted position and velocity data, show that $\mathcal{R}$ conserves the total energy of the system ($\delta E(t) \lesssim 10^{-3}$) for up to $t = 10^6$, closely tracking the ground truth results for all Δ_R (from 100Δ to 4000Δ). In strike contrast, the baseline $\mathcal{V}$ integrator suffers from growing accumulated error of $> 10^{-1}$ when simulated with timestep of 40Δ and exhibits a rapid divergence in the energy for $t > 10^5$ (with associated average trajectory error of 10^9 at $t \approx 10^5$; SI Figure 9).
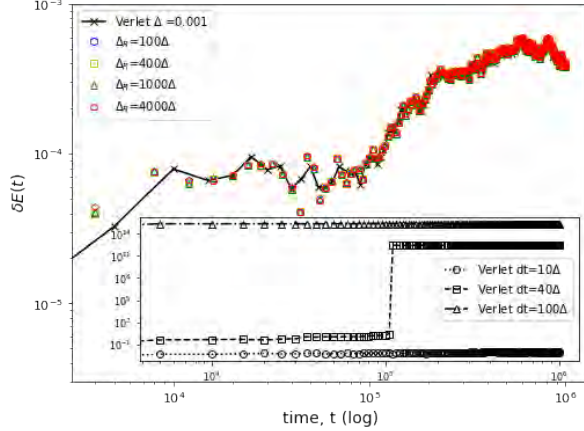

FIG. 5. Energy conservation using $\mathcal{R}$ in a $N = 16$ particle 3D simulation with LJ interactions ($\epsilon = 2$ and $m = 1$) in PBC with initial positions randomly selected and initial velocities set to 0. Black lines are the ground truth results obtained using $\mathcal{V}$ with $\Delta = 0.001$. Outset shows the energy deviation δE (log scale) vs. time t (log scale) predicted by $\mathcal{R}$ for $\Delta_R =$ 100Δ, 400Δ, 1000Δ, 4000Δ. Inset shows the same for time evolution using $\mathcal{V}$ with timestep 10Δ, 40Δ, and 100Δ. Note the order of magnitude difference between the outset and inset y-axis values.

Speedup The DL approach uses Verlet integrator to kickstart the MD simulation and $\mathcal{R}$ to evolve the dynamics forward in time. The calculation of net speedup S_p should incorporate this detail. Without accounting for the time spent on creating training datasets (total < 24 hours for the experiments shown), S_p can be defined as:

$$S_p = t_V S_T / \left[S_V t_V + (S_T - S_V) t_R \Delta / \Delta_R \right] \quad (3)$$

where S_T is the total number of steps needed if the time evolution is performed only using $\mathcal{V}$, t_V is the time for one forward step propagation using $\mathcal{V}$, and t_R is the time for one forward step propagation using $\mathcal{R}$ (other symbols are defined above). S_p is 1 if $S_R = 0$ (no time evolution using $\mathcal{R}$). As most MD simulations are expected to run for long times, in the limit $S_T \gg S_V$, we obtain $S_p \approx t_V \Delta_R / (t_R \Delta)$. For example, for the system of 16 LJ particles with $t_v \approx 0.04392$ seconds, $\Delta = 0.001$, $t_R \approx 0.0026$ seconds, and $\Delta_R = 4.0$, we

find $S_p > 10^4$. Table I shows S_p obtained in different experiments for different Δ_R. For time evolution up to $t = 10^6$ requiring $S_T = 10^9$ steps, we find S_p up to $45\times$ for 1D systems, and from $230\times$ to $32000\times$ for the many particle simulations with LJ interaction potential. $S_p < 1$ data is generally for experiments with 1D systems for $\Delta_R = 100\Delta$, 200Δ and when $t_V < t_R$ (that is, for the 1D cases where the computational cost for time evolution using $\mathcal{V}$ is smaller than that for $\mathcal{R}$).

TABLE I. Net Speedup S_p of the DL approach

Expt. / Δ_R	0.1	0.2	0.4	1.0	2.0	4.0
SHO	0.5	1.3	3.2	8.6	20.0	45.0
LJ	0.9	1.5	3.9	12.8	22.5	42.3
Double-well	0.6	1.2	2.8	8.7	17.3	38.0
Rugged	0.4	0.8	2.1	4.7	9.7	20.6
LJ, 8	600	1000	1500	5500	8300	12000
LJ, 16	3000	4900	7100	20000	28000	32000

Conclusion We introduced an integrator based on recurrent neural networks which can learn the dynamics of many particle systems described by the Newton's equations of motion using trajectory data generated via the traditional Verlet integrators. To evaluate our integrator, we showed that it exhibits excellent energy conservation and produces accurate predictions of the trajectories of systems with up to 16 particles using up to $4000\times$ larger timestep than the Verlet integrators. We also showed that our integrator inherits the Lyapunov instabilities of the Verlet integrator. Finally, we demonstrated significant net speedup over a variety of interaction potentials and total number of particles. Future work will aim to generalize this approach to make it applicable to systems with large number of particles in part by the use of alternative deep learning approaches including non-recurrent networks such as transformers [47].

SUPPORTING INFORMATION

Numerical Integration

In MD simulations, one needs to follow the motion of N particles interacting with potential energy U. Using U, the forces on particles can be derived via $\vec{F} = -\vec{\nabla}U$, and the motion of particles gets determined via Newton's equation $\vec{F} = -m\vec{a}$ ($\vec{\nabla}$ is the gradient operator). There are many numerical integration methods that can integrate these equations. Among these, the velocity Verlet integrator is a very popular choice and we have used this integrator to generate the ground truth for the experiments described in this paper. The velocity Verlet integrator $\mathcal{V}$ is given by the following equations:

$$\vec{x}(t + \Delta) = \vec{x}(t) + \Delta\vec{v}(t) + \frac{\Delta^2}{2m}\vec{f}(t), \qquad (4)$$

$$\vec{v}(t + \Delta) = \vec{v}(t) + \frac{\Delta}{2m}\left(\vec{f}(t) + \vec{f}(t + \Delta)\right). \qquad (5)$$

$\mathcal{V}$ updates the current position $\vec{x}(t)$ at time t to the future position $\vec{x}(t + \Delta)$ after timestep Δ using the current velocity $\vec{v}(t)$ and the force $\vec{f}$ at time t. Here we have adopted the notation for a single particle of mass m; generalization to many particles is straightforward. Similarly, the integrator updates the current velocity $\vec{v}(t)$ at time t to the future velocity $\vec{v}(t + \Delta)$ after timestep Δ using the force $\vec{f}$ at time t and the force $\vec{f}(t + \Delta)$ at time $t + \Delta$. Note that after the position update in Equation 4, the forces will need to be re-evaluated to update the velocities in Equation 5. With $\vec{x}(t + \Delta)$ as the current position and $\vec{v}(t + \Delta)$ as the current velocity, the propagation is continued further.

Equations 4 and 5 suggest that the velocity Verlet integrator requires the current position ($\vec{x}_t$) and velocity coordinates ($\vec{v}_t$) in order to evolve the dynamics forward by Δ. Further, the integrator also uses the force term $\vec{f}$ and mass m to process the update. It is possible to infer $\vec{f}$ using the information encoded in the stream of position data and velocity data such that the time evolution can be done with just a sequence of positions and velocities as input. Similar to ordinary verlet, we illustrate this with a 1-dimensional example of a particle experiencing simple harmonic motion governed by the force $f = -k\vec{x}$. It can be shown that the particle position can be accurately evolved to $t + \Delta t$ via the function $x_{t-\Delta}^{-1}\left(x_t^2 - \Delta(x_{t-\Delta}v_t + x_{t-\Delta}v_t)\right)$ that uses a sequence of 2 positions, 2 velocities and Δ (the information about f and m is hidden in the input sequence). Similarly, It can be shown that the particle velocity can be accurately evolved to $t + \Delta t$ via the function $(x_{t-\Delta} + x_t)^{-1}\left((x_{t-\Delta} + x_t)v_t + (v_t - v_{t-\Delta})(x_t + x_{t+\Delta})\right)$

that uses a sequence of 3 positions and 2 velocities. This idea can be extended to a general system where one could simulate the dynamics (time evolution) via the operator function $\mathcal{V}$: $\vec{x}(t + \Delta), \vec{v}(t + \Delta) = \mathcal{V}\left(\vec{x}_t, \vec{x}_{t-\Delta}, \ldots \vec{x}_{t-n_V\Delta}, \vec{v}_t, \vec{v}_{t-\Delta}, \ldots \vec{v}_{t-k_V\Delta}, \Delta\right)$ that takes a sequence of n_V positions, k_V velocities, and Δ to propagate the particles Δ forward in time.

LSTM Networks

There are several architectures of LSTM units. A common architecture is composed of a cell (the memory part of the LSTM unit) and three "regulators", usually called gates, that regulate the flow of information inside the LSTM unit. An input gate (i_t) controls how much new information is added from the present input (x_t) and past hidden state (h_{t-1}) to our present cell state (c_t). A forget gate (f_t) decides what is removed or retained and carried forward to the current cell state (c_t) from the previous cell state (c_{t-1}). An output gate (o_t) decides what to output as the current hidden state (h_t) from the current cell state (c_t). The math formulation of LSTM is given as:

$$
\begin{aligned}
f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \\
\tilde{c}_t &= \sigma_h(W_c x_t + U_c h_{t-1} + b_c) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \tilde{c}_t \\
h_t &= o_t \circ \sigma_h(c_t). \qquad (6)
\end{aligned}
$$

Here, $x_t \in \mathbf{R^d}$ is the input vector to the LSTM unit, $f_t \in \mathbf{R^h}$ is the forget gate's activation vector, $i_t \in \mathbf{R^h}$ is the input gate's activation vector, $o_t \in \mathbf{R^h}$ is the output gate's activation vector, $h_t \in \mathbf{R^h}$ is the hidden state vector also known as the output vector of the LSTM unit, $c_t \in \mathbf{R^h}$ is the cell state vector, and $\circ$ is the Hadamard product operator. $W \in \mathbf{R^{h \times d}}$, $U \in \mathbf{R^{h \times h}}$ and $b \in \mathbf{R^h}$ are the weight matrices and bias vector parameters which need to be learned during training. σ_g and σ_h represent sigmoid function and hyperbolic tangent functions respectively. The superscripts d and h refer to the number of input features and the number of hidden units respectively.

Data Generation, Preparation, and Preprocessing

Prior domain experience and backward elimination using the adjusted R squared is used for selecting the significant input parameters for creating the training data set. This process determines which inputs are important to change the desired output. The process begins with all

possible inputs and eliminates them one by one, monitoring the adjusted R squared value to determine which are important. Dropping of an input that produces a sharp spike in the R squared value is indicative of the high importance of the input parameter. This process is used to select the inputs parameters for the datasets described under this section. Unlike traditional deep neural networks where you would map the physical inputs to outputs generally distinct from inputs, the inputs and outputs for the RNN approach (integrator $\mathscr{R}$) are both trajectory data as shown in Figure 6. Traditional MD simulation ran with the velocity Verlet algorithm will have multi feature time series data (e.g., position, velocity vectors) separated by $\Delta = 0.001$ in time. To train the integrator $\mathscr{R}$, we prepare the inputs and target data as explained in Figure 6. Since integrator $\mathscr{R}$ will operate with different timestep
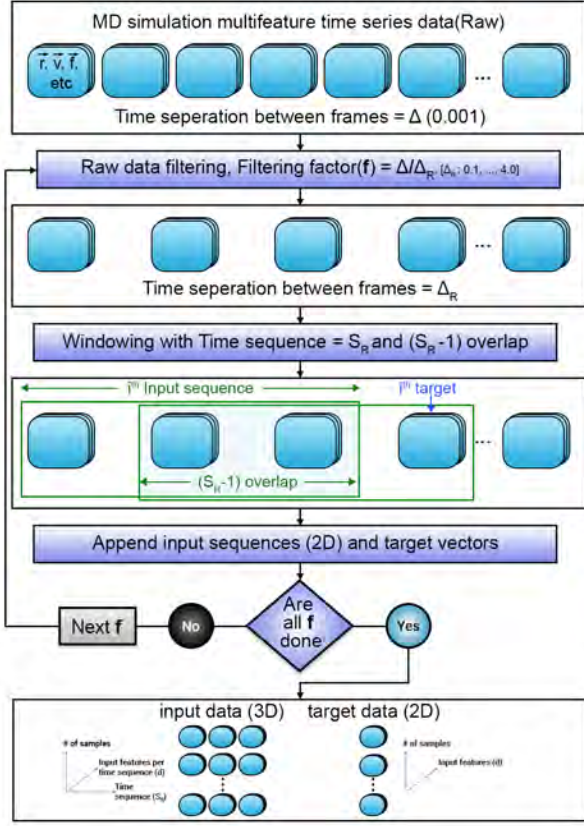

FIG. 6. Data processing for training and testing of integrator $\mathscr{R}$

Δ_R (e.g., 100Δ, 1000Δ), we filter the time series data with a filtering factor defined as Δ_R/Δ to make the data separated with Δ_R. Next, we do windowing through the time series data with the window length of S_R (the RNN model sequence length), and frame overlap of $S_R - 1$ to generate a sequence of length S_R as input and the future configuration as the output (target). We do this repeatedly until the time series is ended and then move

to the next choice for Δ_R and a new associated filtering factor. We do the same process for all simulation data and append all the input sequences and target vectors as shown in Figure 6. In the end, all the input sequences are expected to form a 3D matrix (number of samples $\times$ feature size (d) $\times$ S_R) and the associated output vectors form a 2D matrix (number of samples $\times$ d). A min-max normalization filter is applied to normalize the input data and randomly shuffle them (along the axis of the number of samples) to generate the data for training and testing $\mathscr{R}$. The entire data set is separated into training and testing sets using a ratio of 0.8:0.2.
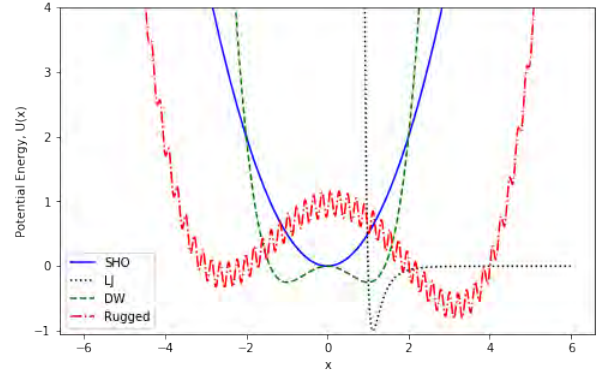

FIG. 7. Potential energies associated with the 1D experiments. Solid, dotted, dashed, and dash-dotted lines represent SHO, LJ, double well, and rugged potential respectively.

We now describe the datasets associated with systems characterized by different interaction potential energies and physical attributes used in the experiments for training and testing $\mathscr{R}$. The potential energy functions are shown in Figure 7. For the 1D systems, the timestep (for velocity Verlet integrator) and the total time were chosen to be $\Delta = 0.001$ and $t_f = 100$. For the 3D many particle systems, $\Delta = 0.001$ and $t_f = 2000$.

Simple harmonic oscillator (SHO) For this system, the potential energy is given by

$$U = \frac{1}{2}kx^2, \tag{7}$$

and the associated force is

$$F = -kx. \tag{8}$$

The dataset is generated by varying three input parameters: mass of the particle $m \in [1, 10]$, spring constant $k \in [1, 10]$, and initial position of the particle $x_0 \in [-10, -1]$. The parameter sweep generated a data set of 500 simulations, each having 50,000 position and velocity values.

1D Lennard-Jones (LJ) system   For this system, the potential energy is given by

$$U(x) = 4\epsilon \left(\left(\frac{\sigma}{x} \right)^{12} - \left(\frac{\sigma}{x} \right)^{6} \right), \qquad (9)$$

and the associated force is

$$F = \frac{48\epsilon}{x} \left(\left(\frac{\sigma}{x} \right)^{12} - 0.5 \left(\frac{\sigma}{x} \right)^{6} \right). \qquad (10)$$

The dataset is generated by varying two input parameters: mass of the particle $m \in [1, 10]$ and the initial position of the particle $x_0 \in [1.0, 3.0]$. The parameter sweep generated a data set of 500 simulations, each having 50,000 position and velocity values.

Particle in a double well (DW)   For this system, the potential energy is given by

$$U = \frac{1}{4}x^4 - \frac{1}{2}x^2, \qquad (11)$$

and the associated force is

$$F = x - x^3. \qquad (12)$$

The dataset is generated by varying two input parameters: mass of the particle $m \in [1, 10]$ and the initial position of the particle $x_0 \in [-3.1, 3.1]$. The parameter sweep generated a data set of 500 simulations, each having 50,000 position and velocity values.

Particle in a rugged potential   For this system, the potential energy is given by

$$U(x) = \frac{x^4 - x^3 - 16x^2 + 4x + 48}{50} + \frac{\sin\left(30(x+5)\right)}{5}, \qquad (13)$$

and the associated force is

$$F = \frac{-4x^3 + 3x^2 + 32x - 4}{50} - 6\cos\left(30(x+5)\right). \qquad (14)$$

The dataset is generated by varying two input parameters: mass of the particle $m \in [1, 10]$ and the initial position of the particle $x_0 \in [-6.1, 6.1]$. The parameter sweep generated a data set of 640 simulations, each having 64,000 position and velocity values.

Many particles interacting with LJ potential   For this 3D system, the interaction potential energy between any two particles is given by the LJ potential:

$$U(r) = 4\epsilon \left(\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^{6} \right), \qquad (15)$$

and the associated force is

$$\vec{F} = \frac{48\epsilon}{r^2} \left(\left(\frac{\sigma}{r} \right)^{12} - 0.5 \left(\frac{\sigma}{r} \right)^{6} \right) \vec{r}. \qquad (16)$$

We prepared two different types of simulation boxes to generate the datasets: cubic box with periodic boundary conditions (PBC) and spherical box with reflective boundary. In each box, we performed simulations with $N = 3, 8$ and 16 particles. Each of these simulations were created as a separated dataset that yielded 6 different datasets to train and test $\mathscr{R}$. The dataset is generated by varying two input parameters: mass of the particle $m \in [1, 10]$ and the initial position of the particles (x_0, y_0, z_0) with each Cartesian coordinate chosen between -3.0 and 3.0. Initial velocities were chosen to be zero. The parameter sweep generated a dataset of 5000 simulations for each of the aforementioned cases (or 30,000 simulations in total).

Feature Extraction and Regression

In the interest of brevity, we only present the details of feature extraction and regression associated with the integrator $\mathscr{R}$ implemented for the most complex case of 16 particles interacting with LJ potential in 3D with PBC. An RNN with two LSTM layers and a dense layer (Figure 1) is implemented in TensorFlow for regression of the prediction for the trajectories of 16 particles (position and velocity vectors). Outputs of the LSTM layers are wrapped with the tanh function. No wrapping functions are used in the final dense layer of the model as integrator $\mathscr{R}$ was trained for regression. The weights and biases of all the layers are trained with an error backpropagation algorithm, implemented via a stochastic gradient descent procedure. The size of the hidden layers was chosen depending on the model complexity and data dimensions. By performing a grid search, hyper-parameters such as the number of units per each of the two layers (n_1, n_2) and final dense layer (n_D), batch size (B), and the number of epochs are optimized to 32, 32, 96, 256, and 2500 respectively. Here we have not done a thorough hyper-parameter optimization but have identified reasonable values for key choices – the architecture and the size of the network, and the length of time series. $\mathscr{R}$ takes a $B \times S_{\mathrm{R}} \times d$ dimensional vector as input, where B and d refer to batch and feature size respectively. For the 3D system considered above, $d = 96$.

Adam optimizer is used to optimize the error backpropagation. The learning rate of Adam optimizer and the dropout rate in the dropout layer is set to 0.0005 and 0.15 respectively to prevent overfitting. Both learning and dropout rates were selected using a trial-and-error process. The weights in the hidden layers and in the output layer are initialized for better convergence using a Xavier normal distribution at the beginning. Xavier normal distribution is characterized with 0 mean and $\sigma = 1/(\sqrt{d_i + d_o})$ variance, where d_i and d_o are input
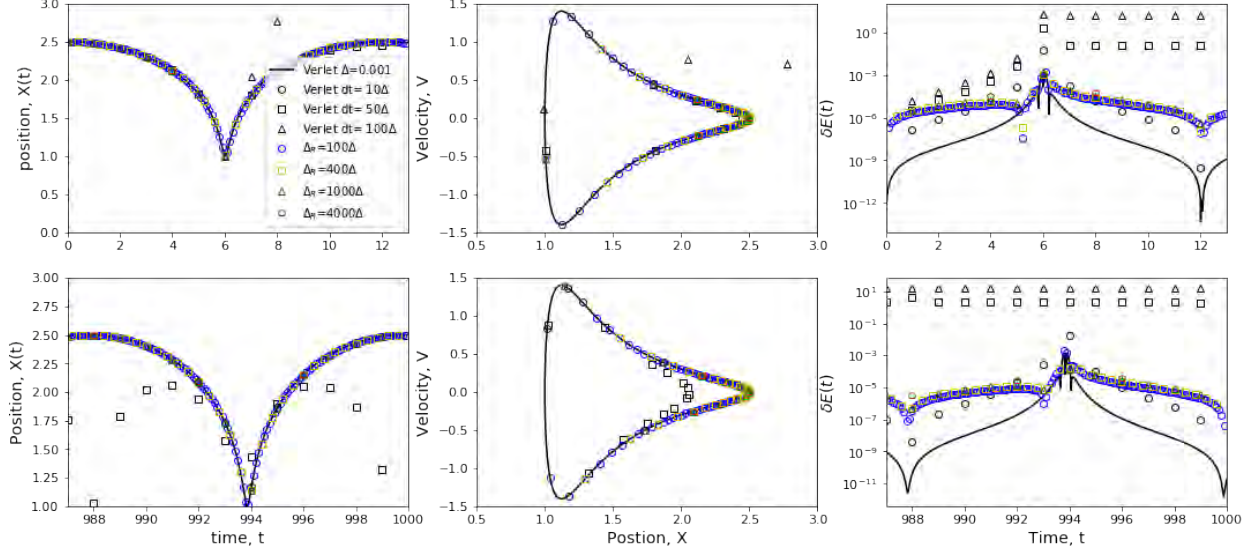
FIG. 8. Dynamics of a particle in a 1D LJ potential ($m = 1$, $x_0 = 2.5$) captured by tracking the position x vs. time t (left column), phase space plot of velocity v vs. x (center column), and the deviation in the total energy (right column). Results are shown for the dynamics produced by $\mathscr{V}$ and $\mathscr{R}$. Top row is the dynamics from $t = 0$ to $t = 13$, and bottom row is the dynamics from $t = 987$ to $t = 1000$. Black lines are the ground truth results obtained using $\mathscr{V}$ with $\Delta = 0.001$. Black color circles, squares and triangles represent the results obtained using $\mathscr{V}$ with timestep 10Δ, 50Δ, and 100Δ. Blue circles, orange squares, green triangles and red pentagons represent the results obtained using $\mathscr{R}$ with $\Delta_R = 100\Delta$, 400Δ, 1000Δ, and 4000Δ respectively.

and output sizes of the hidden layers, respectively [48]. The L2 error (mean square loss) between target and predicted trajectories is used for error calculation. LSTM implementation, training, and testing are programmed using scikit-learn, Keras, and TensorFlow ML libraries [49–51]. Scikit-learn is used for grid search and scaling, Keras is used to save and load models, and TensorFlow is used to create and train the integrator $\mathscr{R}$.

Additional Experiments and Empirical Results

Phase diagram for LJ   This experiment focuses on testing $\mathscr{R}$ to predict the time evolution of the 1D LJ system (with particle of mass $m = 1$ and initial position $x_0 = 2.5$). Training dataset comprising of trajectories (positions and velocities) up to $t = 200$ simulated using $\mathscr{V}$ with $\Delta = 0.001$ are used to train $\mathscr{R}$. Figure 8 shows the 1st period (top row) from $t = 1$ to $t = 13$ and the 75th period (bottom row) from $t = 987$ to $t = 1000$ of the simulation. The 75th period is towards the end of the simulation. For either time periods characterizing the oscillations, the positions predicted by $\mathscr{R}$ remain close to the numerically exact result with $\Delta_R \in [100\Delta, 4000\Delta]$. On the other hand, the trajectory simulated using $\mathscr{V}$ shows errors rising with t for timesteps 10Δ, 50Δ, 100Δ (all $\leq \Delta_R$). Similarly, velocities v produced by $\mathscr{R}$ exhibit very small deviations from the ground truth result as evi-

denced by the phase space plots (v vs. x) in Figure 8. On the other hand, the phase space plot generated using trajectories evolved with $\mathscr{V}$ deviated from the ground truth. Finally, the total energy produced by $\mathscr{R}$ tracks the ground truth result for t up to 1000 and $\Delta_R \in [100\Delta, 4000\Delta]$, in stark contrast with results using $\mathscr{V}$ as the integrator.

Many particle experiments   Figure 9 shows the total error associated with the positions of many particles as a function of time for evolution driven by $\mathscr{R}$ and $\mathscr{V}$. The errors are computed using:

$$\delta r = \frac{1}{N} \sum_{i=1}^{N} |\vec{r}_{i,\text{GT}} - \vec{r}_{i,\mathscr{R}}| \qquad (17)$$

where the formula is shown for $\mathscr{R}$. The simulated system is $N = 16$ particles interacting in 3D with LJ potential under PBC. The ground truth (GT) results for the trajectories are generated using $\mathscr{V}$ with $\Delta = 0.001$. Note that the training set consisted of dynamics up to $t = 2000$ and we intentionally kept the random initial configuration tested here outside the training and testing datasets. We find that the trajectory error δr associated with the positions predicted by $\mathscr{R}$ for $\Delta_R = 100\Delta$ to 4000Δ is $O(10^{-3})$ up to $t = 10^6$. On the other hand, δr for the system evolved using $\mathscr{V}$ with timestep 10Δ increases to $O(10^{-1})$ at around $t \approx 1000$, and δr for 40Δ, 100Δ rises to very large values $> 10^9$, rendering the time evolution of the system meaningless.
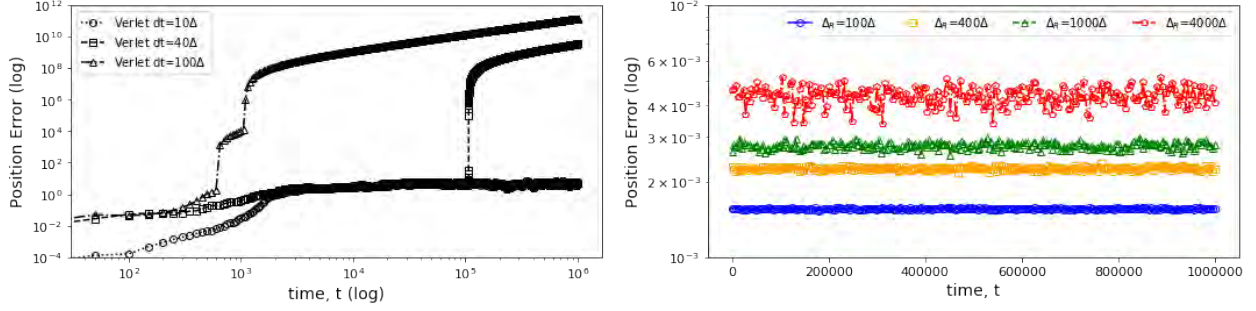
FIG. 9. Average error δr (log scale) in position updates (trajectory) vs. time t incurred in a simulation of a $N = 16$ particle system with LJ interaction potentials ($\epsilon = 2$ and $m = 1$) in 3D. Initial positions are randomly selected and velocities are initialized to zero. All results are compared with the ground truth result obtained using $\mathscr{V}$ with $\Delta = 0.001$. Left plot shows δr for the time evolution up to $t = 1$ million (log scale) for $\mathscr{V}$ using different $dt = 10\Delta, 40\Delta, 100\Delta$ values. Right plot shows δr for the same time evolution using $\mathscr{R}$ with different Δ_R=100Δ, 400Δ, 1000Δ, 4000Δ values.

* kadu@iu.edu
† gcf@iu.edu
‡ vjadhao@iu.edu

[1] Daan Frenkel and Berend Smit, Understanding Molecular Simulation, 2nd ed. (Academic Press, 2001).

[2] Sharon C Glotzer, "Assembly engineering: Materials design for the 21st century (2013 pv danckwerts lecture)," Chemical Engineering Science **121**, 3–9 (2015).

[3] Nicholas E Brunk and Vikram Jadhao, "Computational studies of shape control of charged deformable nanocontainers," Journal of Materials Chemistry B **7**, 6370 (2019).

[4] Oliver Beckstein, Kaihsu Tai, and Mark SP Sansom, "Not ions alone: barriers to ion permeation in nanopores and channels," Journal of the American Chemical Society **126**, 14694–14695 (2004).

[5] R Qiao and Narayana R Aluru, "Charge inversion and flow reversal in a nanochannel electro-osmotic flow," Physical review letters **92**, 198301 (2004).

[6] Jodi A Hadden, Juan R Perilla, Christopher John Schlicksup, Balasubramanian Venkatakrishnan, Adam Zlotnick, and Klaus Schulten, "All-atom molecular dynamics of the hbv capsid reveals insights into biological function and cryo-em resolution limits," Elife **7**, e32478 (2018).

[7] Loup Verlet, "Computer" experiments" on classical fluids. i. thermodynamical properties of lennard-jones molecules," Physical review **159**, 98 (1967).

[8] X Huang, G C Fox, S Serebryakov, A Mohan, P Morkisz, and D Dutta, "Benchmarking deep learning for time series: Challenges and directions," in 2019 IEEE International Conference on Big Data (Big Data) (ieeexplore.ieee.org, 2019) pp. 5679–5682.

[9] Maziar Raissi and George Em Karniadakis, "Hidden physics models: Machine learning of nonlinear partial differential equations," Journal of Computational Physics **357**, 125–141 (2018).

[10] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis, "Multistep neural networks for data-driven discovery of nonlinear dynamical systems," arXiv preprint arXiv:1801.01236 (2018).

[11] Zichao Long, Yiping Lu, Xianzhong Ma, and Bin Dong, "Pde-net: Learning pdes from data," arXiv preprint arXiv:1710.09668 (2017).

[12] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud, "Neural ordinary differential equations," in Advances in neural information processing systems (2018) pp. 6571–6583.

[13] Katsuhiro Endo, Katsufumi Tomobe, and Kenji Yasuoka, "Multi-step time series generator for molecular dynamics," in Thirty-Second AAAI Conference on Artificial Intelligence (2018).

[14] Philip G Breen, Christopher N Foley, Tjarda Boekholt, and Simon Portegies Zwart, "Newton vs the machine: solving the chaotic three-body problem using deep neural networks," arXiv preprint arXiv:1910.07291 (2019).

[15] Zhengdao Chen, Jianyu Zhang, Martin Arjovsky, and Léon Bottou, "Symplectic recurrent neural networks," arXiv preprint arXiv:1909.13334 (2019).

[16] Filip Tronarp, Simo Sarkka, and Philipp Hennig, "Bayesian ode solvers: The maximum a posteriori estimate," arXiv preprint arXiv:2004.00623 (2020).

[17] Xing Shen, Xiaoliang Cheng, and Kewei Liang, "Deep euler method: solving odes by approximating the local truncation error of the euler method," arXiv preprint arXiv:2003.09573 (2020).

[18] Sam Greydanus, Misko Dzamba, and Jason Yosinski, "Hamiltonian neural networks," arXiv preprint arXiv:1906.01563 (2019).

[19] Miles Cranmer, Sam Greydanus, Stephan Hoyer, Peter Battaglia, David Spergel, and Shirley Ho, "Lagrangian neural networks," arXiv preprint arXiv:2003.04630 (2020).

[20] Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al., "Interaction networks for learning about objects, relations and physics," in Advances in neural information processing systems (2016) pp. 4502–4510.

[21] Michael B Chang, Tomer Ullman, Antonio Torralba, and Joshua B Tenenbaum, "A compositional object-based ap-

proach to learning physical dynamics," arXiv preprint arXiv:1612.00341 (2016).

[22] Matthias Rupp, Alexandre Tkatchenko, Klaus-Robert Müller, and O Anatole Von Lilienfeld, "Fast and accurate modeling of molecular atomization energies with machine learning," Physical review letters **108**, 058301 (2012).

[23] Stefan Chmiela, Alexandre Tkatchenko, Huziel E Sauceda, Igor Poltavsky, Kristof T Schütt, and Klaus-Robert Müller, "Machine learning of accurate energy-conserving molecular force fields," Science advances **3**, e1603015 (2017).

[24] Alvaro Sanchez-Gonzalez, Victor Bapst, Kyle Cranmer, and Peter Battaglia, "Hamiltonian graph networks with ode integrators," arXiv preprint arXiv:1909.12790 (2019).

[25] Yeonjong Shin, Jerome Darbon, and George Em Karniadakis, "On the convergence and generalization of physics informed neural networks," arXiv preprint arXiv:2004.01806 (2020).

[26] Andrew L Ferguson, "Machine learning and data science in soft materials engineering," Journal of Physics: Condensed Matter **30**, 043002 (2017).

[27] Pankaj Mehta, Marin Bukov, Ching-Hao Wang, Alexandre GR Day, Clint Richardson, Charles K Fisher, and David J Schwab, "A high-bias, low-variance introduction to machine learning for physicists," Physics reports (2019).

[28] Keith T Butler, Daniel W Davies, Hugh Cartwright, Olexandr Isayev, and Aron Walsh, "Machine learning for molecular and materials science," Nature **559**, 547 (2018).

[29] Geoffrey Fox, James A. Glazier, JCS Kadupitiya, Vikram Jadhao, Minje Kim, Judy Qiu, James P. Sluka, Endre Somogyi, Madhav Marathe, Abhijin Adiga, Jiangzhuo Chen, Oliver Beckstein, and Shantenu Jha, "Learning everywhere: Pervasive machine learning for effective High-Performance computation," in HPDC Workshop at IPDPS 2019 (2019).

[30] Geoffrey Fox, Shantenu Jha, "Learning everywhere: A taxonomy for the integration of machine learning and simulations," in IEEE eScience 2019 Conference (2019) arXiv:1909.13340.

[31] Samuel S. Schoenholz, "Combining machine learning and physics to understand glassy systems," Journal of Physics: Conference Series **1036**, 012021 (2018).

[32] Kelvin Ch'ng, Juan Carrasquilla, Roger G. Melko, and Ehsan Khatami, "Machine learning phases of strongly correlated fermions," Phys. Rev. X **7**, 031038 (2017).

[33] Matthew Spellings and Sharon C Glotzer, "Machine learning for crystal identification and discovery," AIChE Journal **64**, 2198–2206 (2018).

[34] Jiang Wang, Mohit A Gayatri, and Andrew L Ferguson, "Mesoscale simulation and machine learning of asphaltene aggregation phase behavior and molecular assembly landscapes," The Journal of Physical Chemistry B **121**, 4923–4944 (2017).

[35] Junwei Liu, Yang Qi, Zi Yang Meng, and Liang Fu, "Self-learning monte carlo method," Phys. Rev. B **95**, 041101 (2017).

[36] Ashley Z Guo, Emre Sevgen, Hythem Sidky, Jonathan K Whitmer, Jeffrey A Hubbell, and Juan J de Pablo, "Adaptive enhanced sampling by force-biasing using neural net-

works," The Journal of chemical physics **148**, 134108 (2018).

[37] Venkatesh Botu and Rampi Ramprasad, "Adaptive machine learning framework to accelerate ab initio molecular dynamics," International Journal of Quantum Chemistry **115**, 1074–1083 (2015).

[38] JCS Kadupitiya, Geoffrey C Fox, and Vikram Jadhao, "Machine learning for parameter auto-tuning in molecular dynamics simulations: Efficient dynamics of ions near polarizable nanoparticles," The International Journal of High Performance Computing Applications (2020), 10.1177/1094342019899457.

[39] Andrew W Long, Jie Zhang, Steve Granick, and Andrew L Ferguson, "Machine learning assembly landscapes from particle tracking data," Soft Matter **11**, 8141–8153 (2015).

[40] Alireza Moradzadeh and Narayana R Aluru, "Molecular dynamics properties without the full trajectory: A denoising autoencoder network for properties of simple liquids," The journal of physical chemistry letters **10**, 7568–7576 (2019).

[41] Florian Hse, Ignacio Fdez. Galvn, Aln Aspuru-Guzik, Roland Lindh, and Morgane Vacher, "How machine learning can assist the interpretation of ab initio molecular dynamics simulations and conceptual understanding of chemistry," Chem. Sci. **10**, 2298–2307 (2019).

[42] JCS Kadupitiya, Geoffrey C Fox, and Vikram Jadhao, "Machine learning for performance enhancement of molecular dynamics simulations," in International Conference on Computational Science (2019) pp. 116–130.

[43] JCS Kadupitiya, Fanbo Sun, Geoffrey Fox, and Vikram Jadhao, "Machine learning surrogates for molecular dynamics simulations of soft materials," Journal of Computational Science , 101107 (2020).

[44] MF Kasim, D Watson-Parris, L Deaconu, S Oliver, P Hatfield, DH Froula, G Gregori, M Jarvis, S Khatiwala, J Korenaga, et al., "Up to two billion times acceleration of scientific simulations with deep neural architecture search," arXiv preprint arXiv:2001.08055 (2020).

[45] HA Posch and Wm G Hoover, "Lyapunov instability of dense lennard-jones fluids," Physical Review A **38**, 473 (1988).

[46] GD Venneri and William G Hoover, "Simple exact test for well-known molecular dynamics algorithms," Journal of computational physics (Print) **73**, 468–475 (1987).

[47] Alex Graves, "Sequence transduction with recurrent neural networks," arXiv preprint arXiv:1211.3711 (2012), arXiv:1211.3711 [cs.NE].

[48] Xavier Glorot and Yoshua Bengio, "Understanding the difficulty of training deep feedforward neural networks," (2010) pp. 249–256.

[49] François Chollet et al., "Keras," (2015).

[50] Lars Buitinck et al., "Api design for machine learning software: experiences from the scikit-learn project," arXiv:1309.0238 (2013).

[51] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al., "Tensorflow: a system for large-scale machine learning." in OSDI, Vol. 16 (2016) pp. 265–283.