Deep Learning Based Integrators for Solving Newton’s Equations with Large Timesteps

JCS Kadupitiya, Geoffrey C. Fox, and Vikram Jadhao

Intelligent Systems Engineering, Indiana University, Bloomington IN 47408, USA

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

The time evolution of a diverse array of systems is governed by Newton’s equations of motion [1]. These equations are the basis of powerful computational methods such as classical molecular dynamics (MD) that are used to understand the microscopic origins of a wide range of material and biological phenomena [2–5]. In the MD method, Newton’s equations are integrated for a system of many particles using numerical integrators such as Verlet to produce trajectories of particles. The time evolution is generally performed one small timestep at a time for sufficiently long times in order to accurately sample enough representative configurations and extract useful structural and dynamical information. Consider the ordinary Verlet integrator

\[ \vec{x}(t + \Delta) = 2\vec{x}(t) - \vec{x}(t - \Delta) + \Delta^2 \frac{\vec{f}}{m} + O(\Delta^4) \]  

(1)

that updates the current position \( \vec{x}(t) \) of a particle of mass \( m \) at time \( t \) to position \( \vec{x}(t + \Delta) \) after timestep \( \Delta \) using the previous position \( \vec{x}(t - \Delta) \) and the force \( \vec{f} \) at time \( t \). With \( \vec{x}(t + \Delta) \) as the current position, and \( x(t) \) as the previous, the propagation is continued further. While the term \( O(\Delta^4) \) denotes the incurred local error, the global (accumulated) error is \( O(\Delta^2) \) and Equation 1 expresses a 2nd order integrator [2] [6]. The discretization errors are minimized by choosing a sufficiently small \( \Delta \) which often makes the simulations computationally expensive.

The ordinary Verlet integrator requires a sequence of 2 position coordinates \( (\vec{x}_1, \vec{x}_{t-\Delta}) \) in order to update the position using other quantities such as \( \vec{f} \) and \( m \). However, it is possible to infer these other quantities using the information encoded in a longer stream of position data such that the time evolution can be done with only the history of positions as input. We illustrate this with a 1D 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 \( t + \Delta \) via the function \( V(x_t, x_{t-\Delta}, x_{t-2\Delta}) = x_{t-\Delta}^{-1} (x_t^2 - x_{t-\Delta}^2 + x_{t-2\Delta}) \) that uses a sequence of 3 positions. \( V \) also incurs a global error of \( O(\Delta^2) \).

This idea generalizes for more accurate integrators and systems of many particles such that the time evolution can be performed via the operator \( V: \vec{x}(t + \Delta) = V(\vec{x}_t, \vec{x}_{t-\Delta}, \ldots, \vec{x}_{t-n_V\Delta}) \) that takes a sequence of \( n_V \) positions and evolves the system forward by \( \Delta \). For example, one can derive a 4th order Adams-Bashforth-Moulton operator that uses a sequence of 4 positions to evolve the dynamics with a global error of \( O(\Delta^4) \). The longer history of input positions enables such higher-order integrators to accurately perform time evolution with larger \( \Delta \). However, this advantage generally comes at the expense of higher computing costs per timestep that offset the speedup gained via the use of a larger \( \Delta \).

Here, we adopt a data-driven approach and show that accurate integrators based on a sequence of past configurations can be derived using deep learning. The deep neural network based operators are trained using the ground truth results obtained with the 2nd order Verlet integrator. They possess a complex mathematical structure described with ~ 100,000 parameters compared to the relatively simple functional forms of the traditional higher-order integrators. We demonstrate that the network complexity enables the integrators to perform the time evolution of systems of many particles for a wide range of force fields using a timestep up to \( 4000\times \) the baseline \( \Delta \) associated with the 2nd order Verlet integrator. The relatively low time for inferring the positions as predictions of the deep learning model ensures that the overhead costs are low and we demonstrate that speedups up to 32000 can be obtained using larger timesteps.

**RELATED WORK**

Recent years have seen a dramatic rise in the use of machine learning (ML) including deep learning (DL) to enhance the performance of simulations of materials [7–22]. In MD simulations, ML has been used to accelerate the sampling of systems with rugged free-energy landscapes [14], generate new configuration updates in reduced-dimensional space [15], auto-tune simulation timestep [16], classify particle assembly landscapes [17], and derive “surrogates” of MD simulations [18–22]. However, the use of DL for advancements in the
We build on this research with the goal to produce integrators for solving Newton’s equations of motion and drive MD simulations. For the ease of exposition, we describe the DL-based integrator evolving positions of particles (akin to the ordinary Verlet integrator). The process can be adapted to describe an integrator that evolves both positions and velocities (SI). Each component of the position vector of a particle is identified as a feature, with the size of the input and output features being the same. The total feature size for $N$ particles in $D$ physical dimensions is $d = N \times D$. We introduce an operator $R$ based on RNN derived using LSTMs (Figure 1) that employs a sequence $\{x\} = \vec{x}, \vec{x}_{t-\Delta t}, \ldots, \vec{x}_{t-S_R\Delta t}$ of length $S_R$ of current and past positions up to time $t$ to predict the future position at time $t+\Delta_R$: $\vec{x}(t+\Delta_R) = R[\{x\}]$. $R$ can be written as $R[\{x\}] = D [ L_2 [ L_1 [ \{x\} ] ] ]$, where $D, L_1, L_2$ are operators associated with the dense layer, first LSTM layer, and second LSTM layer of the RNN respectively. The layers are stacked up on each other such that the output of one (e.g., $L_1$) becomes the input for another ($L_2$) as shown in Figure 1. Each LSTM layer
METHODS

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) double well (DW), 3) Lennard-Jones (LJ), and 4) rugged (SI). In these initial studies, we select datasets generated by sweeping parameters that mainly shift the initial 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). Details on the data preparation and preprocessing are in the SI.

Figure 2 describes the DL approach using \( \mathcal{R} \) to evolve the dynamics of an \( N \) particle system with timestep \( \Delta_R \). \( \mathcal{R} \) is trained using particle trajectories generated via \( \mathcal{V} \) with small \( \Delta = 0.001 \); these Verlet results provide the ground truth. Input system attributes and \( \Delta \) are fed to \( \mathcal{V} \) 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 \( \Delta_R \) are distilled to feed the \( \mathcal{R} \) operator (\( S_R \) is the sequence length). \( \mathcal{R} \) predicts the time evolution of the system after \( \Delta_R \). Until the end of the simulation, the input sequence to \( \mathcal{R} \) is left shifted to discard the oldest time frame, and the latest frame predicted by \( \mathcal{R} \) is appended to the right of the sequence. The adjusted input sequence is fed to \( \mathcal{R} \) to evolve the system \( \Delta_R \) further in time. The types of training and testing datasets generated and the LSTM architecture details are provided in the next section.

\( n_1, n_2, \) and \( n_D \) are chosen depending on the problem complexity and data dimensions. We discuss these choices and other details of the feature extraction and regression process for the most complex case of 16 particles interacting with LJ potential in 3D with PBC. For this system the feature size \( d = 96 \). By performing a grid search, hyper-parameters such as the number of units for each of the two LSTM layers \( (n_1, n_2) \), number of units in the final dense layer \( (n_D) \), batch size \( (B) \), and
FIG. 3. Errors (log scale) in position updates for 1D simulations. Results are shown for 4 potentials: SHO with $m = 10$, $k = 1$, DW with $m = 1$, $x_0 = -2$, LJ with $m = 1$, $x_0 = 2$, and rugged with $m = 1$, $x_0 = -6$. Trajectories obtained using $\mathcal{P}$ with $\Delta = 0.001$ are taken as the ground truth. (A) Errors using $\mathcal{P}$ with timestep $dt = 10\Delta$. (B), (C), (D), and (E) show errors using $\mathcal{P}$ for SHO, DW, LJ, and rugged potentials respectively for $\Delta_R = 100\Delta$ (circles), $400\Delta$ (squares), $1000\Delta$ (triangles), and $4000\Delta$ (pentagons). (F) Error incurred in position updates using $\mathcal{P}$ at $t = 1000$ vs. timestep $\Delta_R$ for the 4 potentials; inset shows corresponding error using $\mathcal{P}'$ vs. Verlet timestep $dt$.

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. The learning rate of Adam optimizer is set to 0.0005 and the dropout layer is set to 0.15 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 [45].

EXPERIMENTS AND RESULTS

We now compare the results obtained from simulations using $\mathcal{P}$ against the baseline velocity Verlet integrator $\mathcal{P}'$ (unless otherwise noted, $\mathcal{P}'$ will denote the velocity Verlet integrator). For 1D systems, results are shown for four different potentials: 1) simple harmonic oscillator (SHO), 2) Lennard-Jones (LJ), 3) double well (DW), and 4) rugged (see SI Text for details). For the 3D many particle system, experiments are performed on systems of particles interacting with LJ potential. In each case we adopt units such that the values of the system parameters and predicted quantities are around 1. For example, in the case of 3D many particle system, we use reduced LJ units common in MD simulations.

1D Simulations Our first experiments are on testing $\mathcal{P}$ to predict the time evolution of single particle systems in 1D. Training datasets of trajectories simulated using $\mathcal{P}'$ with $\Delta = 0.001$ up to $t = 200$ are used. We find that unlike the traditional Verlet integrator with large timesteps, the errors in positions (trajectory errors) predicted by $\mathcal{P}$ do not increase over time for $t up to 10000$, and are $O(10^{-3})$ for all $\Delta_R \in (100\Delta, 4000\Delta)$ (Figure 3B, C, D, E). These bounded errors can be attributed to the unique functional form of the $\mathcal{P}$ integrator produced via the connected LSTM layers, $L_1$ and $L_2$, and the dense layer $D$ with each layer parameterized using fixed (time-independent) weights. The outputs of $L_1$ and $L_2$ are wrapped with a tanh function and are bounded regardless of the range of values associated with the input positions. The matrix multiplication of the bounded outputs of $L_2$ and the fixed weights of the linear layer $D$ with no wrapping function produces the updated positions.

The errors rise with increase in $\Delta_R$ and the complexity of the potential (e.g., higher for rugged than LJ) but remain within an order of magnitude (Figure 3F). The average error across different potentials for $\Delta_R \in [100\Delta, 4000\Delta]$ are $0.001, 0.002, 0.003, 0.004$ respectively. On the other hand, the trajectory errors incurred using $\mathcal{P}'$ with timestep $10\Delta$ show exponential increase with $t$ and with rising timestep for all potentials considered (Figure 3A and Figure 3F). Similarly, velocity $v$ and position $x$ predictions produced by $\mathcal{P}$ exhibit small trajectory errors, and the phase diagram ($v$ vs. $x$) and the total energy track the ground truth results for $t up to 10000$ and $\Delta_R$ up to $4000\Delta$ (SI Figure 8). $\mathcal{P}$ learns the energy conservation feature associated with the dynamics governed by Newton’s equations of motion.

Lyapunov Instability Next set of experiments involve assessing the stability of the solutions (trajectories) pre-
dicted by the integrator $\mathcal{R}$. This is typically discussed in terms of Lyapunov instability which states that close-by trajectories diverge exponentially [46, 47]. Lyapunov instability is present in standard Verlet integrators for many potentials. Trajectories of a particle of $m = 1$ in 1D LJ potential generated using $\mathcal{Y}$ are used for these experiments. A small perturbation in the momentum $\delta p$ is introduced at the start of trajectory. Lines and markers are the results of simulations driven by $\mathcal{Y}$ and $\mathcal{R}$ integrators respectively. Outset shows the trajectory for $\delta p = 0$, $10^{-4}$, $10^{-3}$. Insets show the absolute difference between positions generated with $\delta p \neq 0$ and $\delta p = 0$ ($\Delta x(t) \equiv |x(t, 0) - x(t, \delta p)|$) as produced by $\mathcal{Y}$ (left inset) and $\mathcal{R}$ (right inset).

Many Particle Simulations in 3D Our final set of experiments probe the extension of the proposed DL approach to systems of $N = 3, 8, 16$ particles interacting with LJ potential. $\mathcal{R}$ is trained using trajectories generated by $\mathcal{Y}$ ($\Delta = 0.001$) in a cubic box with periodic boundary conditions (PBC) or in a spherical hard-wall confinement with reflective boundaries. In the former case, $\mathcal{R}$ observed the positions of the particles governed by Newton’s equations of motion and the re-mapping dictated by the use of PBC. We find that $\mathcal{R}$ successfully evolved the positions and velocities of particles up to time $t = 10^6$ for all the $N$-particle systems studied with $\Delta x$ up to 4000$\Delta$. In the interest of brevity, we discuss the results of experiments on the $N = 16$ system in PBC. Results with the spherical hard-wall confinement exhibited similar position errors ($\delta r \sim O(10^{-3})$) (SI Figure 10). The training dataset for this study involved time evolution up to $t = 2000$ and the characteristic LJ potential energy $\epsilon = 1$. Results are shown for time evolution using $\mathcal{R}$ for $\epsilon = 2$ (not present in the entire dataset); slightly better results were obtained for $\epsilon = 1$. 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 $\Delta x = 100\Delta$ to 4000$\Delta$ 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 show that $\mathcal{R}$ conserves the total energy of the system ($\delta E(t) \lesssim 10^{-3}$) for up to $t = 10^6$ for all $\Delta x$ (from 100$\Delta$ to 4000$\Delta$). In strike contrast, the baseline $\mathcal{Y}$ integrator suffers from growing accumulated error of $> 10^{-1}$ when simulated with timestep of 40$\Delta$ and exhibits a rapid divergence in the energy for $t > 10^3$ (with associated average trajectory error of $10^9$ at $t \approx 10^5$; SI Figure 9).

Speedup The DL approach uses Verlet integrator to kickstart the MD simulation and $\mathcal{R}$ to evolve the dynam-
ics forward in time. This detail should be incorporated in calculating the net speedup $S_p$:

$$S_p = t_V S_T / [S_V t_V + (S_T - S_V) t_R \Delta / \Delta_R]$$  \hspace{1cm} (3)$$

where $S_T$ is the total number of steps needed if the time evolution is performed only using $\mathcal{Y}$, $t_V$ is the time for one forward step propagation using $\mathcal{Y}$, and $t_R$ is the time for one forward step propagation using $\mathcal{R}$. We have not accounted for the time spent on creating training datasets ($< 24$ hours for the experiments shown). $S_p$ 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 $S_p \approx t_V \Delta / (t_R \Delta)$. 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 $S_p \approx 10^4$. Table I shows $S_p$ for different experiments at different $\Delta_R$. For time evolution up to $t = 10^6$ positions, and used the resulting integrator to evolve the dynamics of the 1D system of a particle in an LJ potential. We extracted the position error $\delta r$ incurred in the time evolution up to $t = 1000$ using the ground truth results obtained with the 2nd order baseline ordinary Verlet integrator $\mathcal{Y}$ with $\Delta = 0.001$.

We find that for smaller sequence length $S_R = 3$ or 4, $\mathcal{R}$ is only able to accurately propagate the dynamics with $\delta r \sim O(\Delta^3)$ for $\Delta_R \lesssim 10\Delta$ (SI Figure [1]). For $S_R = 3$, the error associated with $\mathcal{R}$ rises steeply for $\Delta_R > 10\Delta$ and spans over 4 orders of magnitude, similar to the results for the 2nd order Verlet integrator. For $S_R = 4$, the accuracy improves and error scaling is similar to that produced by a traditional 4th order integrator such as Runge-Kutta or Adams-Bashforth-Moulton method (with global error $\sim O(\Delta^4)$). $\mathcal{R}$ integrator trained with sequence length $S_R = 5$ shows a much weaker rise in error limited to within an order of magnitude as $\Delta_R$ rises up to 4000$\Delta$. Thus, the $\mathcal{R}$ operator with increasing sequence length $S_R$ learns a progressively more accurate integrator employing the trajectory data generated by the 2nd order Verlet ($\mathcal{Y}$) integrator.

**Comparison with Traditional Large-Timestep Integrators**  The main source of speedup shown in Table I is the use of large timestep $\Delta_R \gg \Delta$ in performing particle dynamics with $\mathcal{R}$. Extending the approximate expression for speedup to a general higher-order / multi-step traditional integrator $I$, we find $S_p^I \approx (t_I / t_R) \times (\Delta_R / \Delta_1) = S_p \times (t_I / t_V) \times (\Delta / \Delta_1)$, where $t_I$ and $\Delta_1$ are the time spent and timestep used in the single-step forward propagation using $I$. The use of $I$ as the baseline (reference) integrator can reduce the reported net speedup numbers tabulated above as $\Delta / \Delta_1 \approx 1$. However, the increase of timestep with the use of $I$ generally accompanies a rise in the computational overhead costs as the time for single step propagation typically rises, i.e. $t_I > t_V$. For example, in the case of a 3D system of 16 particles interacting with LJ potential, a 4th order integrator (e.g., Runge-Kutta, Adams-Bashforth-Moulton) is expected to yield $\Delta / \Delta_1 \approx 0.1$ but also take $2t_V$ to $5t_V$ longer along with a much higher memory utilization ($4 \times$ to $8 \times$). Moreover, $t_I / t_V$ is expected to rise with the number of particles $N$ following the $O(N^2)$ scaling in several applications due to the higher number of force computations and memory utilization. Thus, with increasing $N$, the reduction in $\Delta / \Delta_1$ is expected to be offset by the increase in $t_I / t_V$ to yield $S_p^I$ values similar to $S_p$. The $\mathcal{R}$ integrator uniquely bypasses many of these computational challenges at the design stage and enjoys a relatively small forward step propagation (inference) time $t_R$ that is largely independent of the timestep $\Delta_R$ and exhibits $O(N)$ scaling. In the case of the above example, $t_R = 1.48 \times 10^{-5}$ seconds (compared to $t_V \approx 0.04$ seconds) is roughly the same for over 3 orders of magnitude change in $\Delta_R$ and is expected.

**DISCUSSION**

**Role of Sequence Length**  The DL-based integrator $\mathcal{R}$ developed here learns both the interaction potentials and the dynamics of the particles based on its experience with the solutions (trajectories) of Newton’s equations of motion via the training data generated using the 2nd order Verlet integrator. The $\mathcal{R}$ integrator employs a sequence of 5 past configurations (e.g., positions of particles) to predict the future configuration using a timestep up to 4000$\Delta$ with high accuracy (where $\Delta$ is the baseline timestep associated with the 2nd order Verlet integrator). We hypothesize that the long input sequence length endows $\mathcal{R}$ with properties that enable it to perform the time evolution with large timesteps. To test this hypothesis, we trained $\mathcal{R}$ with an input sequence length of 3 and 4 requiring $S_T = 10^9$ steps, we find $S_p$ up to $45\times$ for 1D systems, and from $230 \times 32000 \times$ for the many particle simulations with LJ forces. $S_p < 1$ data is generally for 1D systems with $\Delta_R = 100\Delta$, $200\Delta$, and when $t_V < t_R$ (e.g., for the 1D cases where the computational cost for time evolution using $\mathcal{Y}$ is smaller than that for $\mathcal{R}$).

**TABLE I. Net speedup $S_p$ using the $\mathcal{R}$ integrator**

<table>
<thead>
<tr>
<th>Expt. / $\Delta_R$</th>
<th>100</th>
<th>200</th>
<th>400</th>
<th>1000</th>
<th>2000</th>
<th>4000</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHO</td>
<td>2.0</td>
<td>1.3</td>
<td>3.2</td>
<td>8.6</td>
<td>20.0</td>
<td>45.0</td>
</tr>
<tr>
<td>Double-well</td>
<td>0.5</td>
<td>0.6</td>
<td>1.2</td>
<td>2.8</td>
<td>8.7</td>
<td>17.3</td>
</tr>
<tr>
<td>LJ</td>
<td>0.4</td>
<td>0.9</td>
<td>1.5</td>
<td>3.9</td>
<td>12.8</td>
<td>22.5</td>
</tr>
<tr>
<td>Rugged</td>
<td>0.4</td>
<td>0.8</td>
<td>2.1</td>
<td>4.7</td>
<td>9.7</td>
<td>20.6</td>
</tr>
<tr>
<td>LJ, 8</td>
<td>600</td>
<td>1000</td>
<td>1500</td>
<td>5500</td>
<td>8300</td>
<td>12000</td>
</tr>
<tr>
<td>LJ, 16</td>
<td>3000</td>
<td>4900</td>
<td>7100</td>
<td>20000</td>
<td>28000</td>
<td>32000</td>
</tr>
</tbody>
</table>

The main source of speedup shown in Table I is the use of large timestep $\Delta_R \gg \Delta$ in performing particle dynamics with $\mathcal{R}$. Extending the approximate expression for speedup to a general higher-order / multi-step traditional integrator $I$, we find $S_p^I \approx (t_I / t_R) \times (\Delta_R / \Delta_1) = S_p \times (t_I / t_V) \times (\Delta / \Delta_1)$, where $t_I$ and $\Delta_1$ are the time spent and timestep used in the single-step forward propagation using $I$. The use of $I$ as the baseline (reference) integrator can reduce the reported net speedup numbers tabulated above as $\Delta / \Delta_1 \approx 1$. However, the increase of timestep with the use of $I$ generally accompanies a rise in the computational overhead costs as the time for single step propagation typically rises, i.e. $t_I > t_V$. For example, in the case of a 3D system of 16 particles interacting with LJ potential, a 4th order integrator (e.g., Runge-Kutta, Adams-Bashforth-Moulton) is expected to yield $\Delta / \Delta_1 \approx 0.1$ but also take $2t_V$ to $5t_V$ longer along with a much higher memory utilization ($4 \times$ to $8 \times$). Moreover, $t_I / t_V$ is expected to rise with the number of particles $N$ following the $O(N^2)$ scaling in several applications due to the higher number of force computations and memory utilization. Thus, with increasing $N$, the reduction in $\Delta / \Delta_1$ is expected to be offset by the increase in $t_I / t_V$ to yield $S_p^I$ values similar to $S_p$. The $\mathcal{R}$ integrator uniquely bypasses many of these computational challenges at the design stage and enjoys a relatively small forward step propagation (inference) time $t_R$ that is largely independent of the timestep $\Delta_R$ and exhibits $O(N)$ scaling. In the case of the above example, $t_R = 1.48 \times 10^{-5}$ seconds (compared to $t_V \approx 0.04$ seconds) is roughly the same for over 3 orders of magnitude change in $\Delta_R$ and is expected.
to scale linearly with $N$.

**Challenges and Future Work** While the use of the DL approach to design integrators has many unique benefits as evidenced by the initial results, to make the approach more generalizable and suitable for practical applications, further research is needed. One key challenge is predicting the dynamics of systems that exhibit rare-event characteristics such that times to yield the low-energy configurations are longer due to the kinetic bottlenecks that can trap the particles in metastable pathways. In these cases, training the RNN model to “see” a variety of distinct phase space explorations may prove too time-consuming. Another challenge is the scaling of the approach to a larger number of particles $N$. Future work will seek to address these challenges. We will investigate the design of DL-based models for predicting the trajectories of particles in systems where self-organization into aggregates occurs at longer times depending on the combination of particle and potential attributes. We plan to address the issue of scaling with $N$ in part by the use of hierarchical RNNs [48] and alternative DL approaches [49] including transformers [50]. Also, we plan to explore the “recurrent” versions of physics-informed neural network architectures [33, 34] where in addition to the use of sequence-to-sequence mapping, the infusion of conservation laws in the design of loss functions of the LSTM units can help scale the approach to larger $N$. Future work will also explore the training of $R$ using trajectories produced via simulations in different ensembles (e.g., NVT) in order to test its accuracy in predicting configuration updates where thermal effects (often stochastic in nature) are incorporated. The implications of the DL-based integrators for other MD methods such as discontinuous MD, which enable large gains in speedup with small loss of accuracy, will also be explored [51].

**CONCLUSION**

We introduced an integrator based on recurrent neural networks which can learn the dynamics of many particle systems described by Newton’s equations of motion using trajectory data generated via the traditional Verlet integrators. The data-driven approach provides a new way to design integrators that are accurate for large timesteps by controlling only a few parameters (e.g., the sequence length). The deep learning model enables an accurate and rapid determination of the large set of 100,000 trainable parameters characterizing the mathematically complex functional form of the neural network based integrator. This complexity makes the integrator robust enough to be applicable over a wide range of potentials and generalizable to systems of many particles. 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. The deep learning based integrator introduced here and the associated results illustrate an important approach to learn the time evolution operators which can be applied across a variety of fields including fluid dynamics [31], robotics [52], and fusion-energy science [53].

**Acknowledgments**

This work is partially supported by the National Science Foundation (NSF) through awards 1720625 and DMR-1753182. G.C.F was partially supported by NSF CIF21 DIBBS 1443054 and CINES 1835598 awards. Simulations were performed using the Big Red III supercomputing system. V.J. thanks M. O. Robbins for useful discussions.

**SUPPORTING INFORMATION**

Reformulation of Velocity Verlet Integrator using Deep Learning

Many numerical integration methods can perform the dynamics of particles determined by Newton’s equation $\ddot{x} = m\ddot{a}$. Among these, the velocity Verlet integrator $\mathcal{V}$ is a popular choice that is also used in our work to generate the ground truth for many experiments. $\mathcal{V}$ is expressed as:

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

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

Here we have adopted the notation for a single particle of mass $m$; generalization to many particles is straightforward. $\mathcal{V}$ updates the current position $\bar{x}(t)$ at time $t$ to the future position $\bar{x}(t + \Delta)$ after timestep $\Delta$ using the current velocity $\bar{v}$ and the force $\bar{f}$ at time $t$. The current velocity $\bar{v}(t)$ at time $t$ is updated to velocity $\bar{v}(t + \Delta)$ after timestep $\Delta$ using the force $\bar{f}(t)$ at time $t$ and the force $\bar{f}(t + \Delta)$ at time $t + \Delta$. Note that after the position update in Equation [4] the force needs to be re-evaluated to update the velocity in Equation [5]. With $\bar{x}(t + \Delta)$ as the current position and $\bar{v}(t + \Delta)$ as the current velocity, the propagation is continued further.
We describe a data-driven approach to design an integrator that takes a sequence of past positions and velocities as input and generates the future position and velocity of the particle. Each component of the position and the velocity vectors of a particle is identified as a feature. Inputs and outputs have the same feature size. The total feature size for $N$ particles in $D$ dimensions is $d = N \times D \times 2$. For example, for $N = 16$ particles interacting in 3D, the size of both the input and output features is $d = 96$.

We introduce an operator $\mathcal{R}$ based on RNN derived using LSTMs (Figure 1) that employs a sequence of current and past positions and velocities (treated as input features) up to time $t$ to predict the future position and velocity at time $t + \Delta R$: $(\vec{x}_{t + \Delta R}, \vec{v}_{t + \Delta R}) = \mathcal{R}[\vec{x}_t, \vec{x}_{t - \Delta R}, \ldots, \vec{x}_{t - S_R \Delta_R}, \vec{v}_t, \vec{v}_{t - \Delta R}, \ldots, \vec{v}_{t - S_R \Delta_R}]$ where $\Delta_R$ is the timestep associated with the operator $\mathcal{R}$ and $S_R$ is the length of the sequence. $\mathcal{R}$ can be written as $\mathcal{R}\{x, v\} = \mathcal{D}[\mathcal{L}_2[\mathcal{L}_1\{\{x, v\}\}]]$, where $\mathcal{D}$, $\mathcal{L}_1$, $\mathcal{L}_2$ are operators associated with the dense layer, first LSTM layer, and second LSTM layer of the RNN respectively, and $\{x, v\}$ is a shorthand for the sequence $\vec{x}_t, \vec{x}_{t - \Delta R}, \ldots, \vec{x}_{t - S_R \Delta_R}, \vec{v}_t, \vec{v}_{t - \Delta R}, \ldots, \vec{v}_{t - S_R \Delta_R}$. Similar to the derivation shown in the main text, these layers are stacked up on each other such that the output of one (e.g., $\mathcal{L}_1$) becomes the input for another ($\mathcal{L}_2$). Post training, these layers acquire well-determined values for all the parameters and the integrator $\mathcal{R}$ emerges with the more explicit expression:

$$\mathcal{R}\{x, v\} = \mathcal{D}[\mathcal{L}_2[\mathcal{L}_1\{\{x, v\}, \{P_1\}, \{P_2\}, \{P_D\}\}]],$$

(6)

where $\{P_1\}, \{P_2\}, \{P_D\}$ are trained parameters associated with LSTM layer 1, LSTM layer 2, and the dense layer respectively. While training, $\{x, v\}$ is fed as a vector of size $B \times S_R \times d$, where $B$ is a training parameter denoting the batch size. When used as an integrator (testing phase), $B = 1$. $\mathcal{R}$ is trained on trajectories produced by the using traditional velocity Verlet integrator $V$ to evolve the dynamics dictated by Newton’s equations of motion. Characterized with up to 100,000 parameters, $\mathcal{R}$ can be considered as a reformulation of the velocity Verlet integrator $V$ (Equation 3 and 5).

**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 LSTM formulation can be expressed as:

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

(7)

Here, $x_t \in \mathbb{R}^d$ is the input vector to the LSTM unit, $f_t \in \mathbb{R}^h$ is the forget gate’s activation vector, $i_t \in \mathbb{R}^h$ is the input gate’s activation vector, $o_t \in \mathbb{R}^h$ is the output gate’s activation vector, $h_t \in \mathbb{R}^h$ is the hidden state vector also known as the output vector of the LSTM unit, $c_t \in \mathbb{R}^h$ is the cell state vector, and $\sigma$ is the Hadamard product operator. $W \in \mathbb{R}^{h \times d}$ and $U \in \mathbb{R}^{h \times h}$ are the weight matrices and $b \in \mathbb{R}^h$ are the bias vector parameters which need to be learned during training. $\sigma_g$ and $\sigma_h$ 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**

In most cases, prior domain experience and backward elimination using the adjusted R squared method is employed to determine the important input parameters that significantly change the desired output. In some cases, such as particles in Lennard-Jones (LJ) potential, only a subset of the important input parameters are varied to create the dataset to train the RNN-based $\mathcal{R}$ integrator. Unlike traditional deep neural networks where the physical inputs are mapped to outputs generally distinct from inputs, the inputs and outputs for the RNN approach are both trajectory data as shown in Figure 6. MD simulation driven using the velocity Verlet algorithm generally produces time series data separated by $\Delta = 0.001$ in time. The data is characterized by multiple features such as position and velocity vectors. To train the integrator $\mathcal{R}$, we prepare the inputs and target data as explained in Figure 6. Since integrator $\mathcal{R}$ will operate with different timestep $\Delta_R$ (e.g., 100$\Delta$, 1000$\Delta$), we filter the time series data with a filtering factor defined as $\Delta_R/\Delta$ to make the data separated with $\Delta_R$. Next, we do windowing through the time series data with the window length of $S_R$ (the RNN model sequence length), and frame overlap length 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 $\Delta R$ 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.

**FIG. 6.** Data processing for training and testing of integrator $R$ with sequence length $S_R$ as a key input parameter.

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 $\times d$). The entire dataset is randomly shuffled (along the axis of the number of samples) and separated into training and testing sets using a ratio of 0.8:0.2. We now describe the datasets associated with systems characterized by different potential energies and particle attributes used in the experiments for training and testing $R$. The potential energy functions are shown in Figure 7. For the 1D systems, the timestep and the total time associated with generating the ground truth using the velocity Verlet integrator were $\Delta = 0.001$ 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

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

and the associated force is

$$f = -kx.$$  

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 dataset 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

$$U = \frac{1}{4}x^4 - \frac{1}{2}x^2,$$

and the associated force is

$$f = x - x^3.$$  

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 dataset 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

$$U(x) = 4\epsilon \left( \left( \frac{1}{x} \right)^{12} - \frac{1}{2} \left( \frac{1}{x} \right)^6 \right),$$

and the associated force is

$$f = \frac{48\epsilon}{x} \left( \left( \frac{1}{x} \right)^{12} - 0.5 \left( \frac{1}{x} \right)^6 \right).$$  

**FIG. 7.** Potential energies associated with the 1D experiments. Dash-dotted, dotted, dashed, and solid lines represent SHO, DW, LJ, and rugged potential respectively.
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 dataset 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

\[
U(x) = \frac{x^4 - x^3 - 16x^2 + 4x + 48}{50} + \frac{\sin(30(x + 5))}{5},
\]

and the associated force is

\[
f = \frac{-4x^3 + 3x^2 + 32x - 4}{50} - 6\cos(30(x + 5)).
\]

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 dataset 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:

\[
U(r) = 4\epsilon \left( \left( \frac{1}{r} \right)^{12} - \left( \frac{1}{r} \right)^{6} \right) + 0.0163\epsilon \quad \text{for } r \leq 2.5,
\]

and the associated force is

\[
\vec{f} = \frac{48\epsilon}{r^2} \left( \left( \frac{1}{r} \right)^{12} - 0.5 \left( \frac{1}{r} \right)^{6} \right) \vec{r} \quad \text{for } r \leq 2.5.
\]

For \( r > 2.5 \), both \( U \) and \( \vec{f} \) are 0. Note that the unit of distance is chosen to be the particle diameter \( \sigma \) and the unit of energy is chosen such that the LJ potential well depth \( \epsilon \) has values around 1. 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 was created as a separate dataset that yielded 6 different datasets to train and test \( \mathcal{R} \). 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 \(3.0\). The well depth was held constant to \( \epsilon = 1 \) in creating the training dataset. 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).

**Additional Experiments and Empirical Results**

**Phase diagram for LJ** This experiment focused on testing \( \mathcal{R} \) to predict the time evolution of the 1D LJ system. Training dataset comprised of trajectories (positions and velocities) generated using \( \mathcal{Y} \) with \( \Delta = 0.001 \) for up to \( t = 200 \). 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. The result is shown for a particle of mass \( m = 1 \), initial position \( x_0 = 2.5 \), and \( \epsilon = 1 \). For either time periods characterizing the oscillations, the positions predicted by \( \mathcal{R} \) using a wide range of timestep \( \Delta_R \in [100\Delta, 4000\Delta] \) remain close to the numerically exact result. On the other hand, the trajectory simulated using \( \mathcal{Y} \) shows errors rising with \( t \) for timesteps \( \gtrsim 10\Delta \). Similarly, velocities \( v \) produced by \( \mathcal{R} \) exhibit very small deviations from the ground truth result for all \( \Delta_R \) investigated as evidenced by the phase space plots (\( v \) vs. \( x \)) in Figure 8. On the other hand, the phase space plot generated using trajectories evolved with \( \mathcal{Y} \) deviated from the ground truth for timesteps \( \gtrsim 10\Delta \). Finally, the total energy produced by \( \mathcal{R} \) tracks the ground truth result for \( t \) up to 1000 and \( \Delta_R \in [100\Delta, 4000\Delta] \), in stark contrast with results using \( \mathcal{Y} \) 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 dynamics driven by \( \mathcal{R} \) and \( \mathcal{Y} \). The errors are computed using:

\[
\delta r = \frac{1}{N} \sum_{i=1}^{N} |\vec{r}_{i,GT} - \vec{r}_{i,\mathcal{R}}|
\]

where the formula is shown for \( \mathcal{R} \). Here \( N \) is the number of particles, \( \vec{r}_{i,GT} \) is the ground truth result for the position of the \( i \)th particle at time \( t \), \( \vec{r}_{i,\mathcal{R}} \) is the position of the \( i \)th particle predicted by \( \mathcal{R} \) at the same time \( t \). The ground truth (GT) results for the trajectories are generated using \( \mathcal{Y} \) 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. The simulated system is \( N = 16 \) particles interacting in 3D with LJ potentials under PBC. We find that the trajectory error \( \delta r \) associated with the positions predicted by \( \mathcal{R} \) for \( \Delta_R = 100\Delta \) to 4000\Delta is \( O(10^{-3}) \) up to \( t = 10^6 \). On the other hand, \( \delta r \) for the system evolved using \( \mathcal{Y} \) with timestep 10\Delta increases to \( O(10^{-1}) \) at around \( t \approx 1000 \). For timestep values of 40\Delta, 100\Delta, \( \mathcal{Y} \) does worse as \( \delta r \) rises to very large values \( > 10^9 \), rendering the time evolution of the system meaningless.

**Testing \( \mathcal{R} \) with different sequence lengths** In this experiment, we tested the dependence of the accuracy of time evolution performed by \( \mathcal{R} \) integrator on the sequence length of input configurations used to train the operator. We trained \( \mathcal{R} \) with different input sequence length \( S_R \) experimenting with \( S_R = 3, 4 \) and 5. The training was performed using the same dataset employed
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 \(\mathcal{V}\) and \(\mathcal{A}\). 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 \(\mathcal{V}\) with \(\Delta = 0.001\). Black color circles, squares and triangles represent the results obtained using \(\mathcal{V}\) with timestep \(10\Delta\), \(50\Delta\), and \(100\Delta\). Blue circles, orange squares, green triangles and red pentagons represent the results obtained using \(\mathcal{A}\) with \(\Delta_R = 100\Delta\), \(400\Delta\), \(1000\Delta\), and \(4000\Delta\) respectively.

FIG. 9. Average error \(\delta_r\) (log scale) in evolved positions vs. time \(t\) incurred in a 3D simulation of \(N = 16\) particles interacting with LJ potentials \((\epsilon = 2\) and \(m = 1)\) in periodic boundary conditions (PBC). Initial positions are randomly selected and velocities are initialized to zero. All results are compared with the ground truth result obtained using \(\mathcal{V}\) with \(\Delta = 0.001\). Left plot shows \(\delta_r\) for the time evolution up to \(t = 1\) million (log scale) for \(\mathcal{V}\) using different \(dt = 10\Delta, 40\Delta, 100\Delta\) values. Right plot shows \(\delta_r\) for the same time evolution using \(\mathcal{A}\) with \(\Delta_R = 100\Delta, 400\Delta, 1000\Delta,\) and \(4000\Delta\).

in the original 1D LJ system experiment. The resulting integrator was used to evolve the dynamics of the 1D system of a particle in an LJ potential. Figure 11 shows the position error \(\delta_r\) incurred in the time evolution up to \(t = 1000\) vs. \(\Delta\) using different integrators. \(\delta_r\) is evaluated using the ground truth results obtained with the 2nd order baseline Verlet integrator \(\mathcal{V}\) with \(\Delta = 0.001\).

We find that for smaller sequence length \(S_R = 3\) or \(4\), \(\mathcal{A}\) fails to accurately perform the time evolution for \(\Delta_R \gtrsim 10\Delta\) incurring errors \(\delta_r \gtrsim O(\Delta^{-2})\) (SI Figure 11). For \(S_R = 3\), the error associated with \(\mathcal{A}\) rises steeply for \(\Delta_R > 10\Delta\) and spans over 4 orders of magnitude, similar to the results for the 2nd order Verlet integrator (or its equivalent \(\mathcal{V}\) operator that takes a sequence of 3 positions as input). For \(S_R = 4\), the accuracy improves and error scaling is similar to that produced by a traditional 4th order integrator such as Runge-Kutta or Adams-Bashforth-Moulton method (with global error \(\sim O(\Delta^4)\)). \(\mathcal{A}\) integrator trained with sequence length \(S_R = 5\) shows a much weaker rise in error limited to within an order of magnitude as \(\Delta_R\) rises up to \(4000\Delta\).

\* kadu@iu.edu
\† gcf@iu.edu
FIG. 10. Average error $\delta r$ (log scale) in evolved positions vs. time $t$ incurred in a 3D simulation of $N = 16$ particles interacting with LJ potentials ($\epsilon = 2$ and $m = 1$) in spherical hard-wall confinement. Initial positions are randomly selected and velocities are initialized to zero. All results are compared with the ground truth result obtained using $Y$ with $\Delta = 0.001$. Left plot shows $\delta r$ for the time evolution up to $t = 1$ million (log scale) for $Y$ using different $dt = 10\Delta, 40\Delta, 100\Delta$ values. Right plot shows $\delta r$ for the same time evolution using $R$ with $\Delta R = 100\Delta, 400\Delta, 1000\Delta$, and $4000\Delta$.

FIG. 11. Error in position at time $t = 1000$ for different timestep $dt \geq 10^{-3}$ in 1D simulations of a particle of mass $m = 1$ in an LJ potential with initial position $x_0 = 2.0$. The 2nd order baseline Verlet integrator $Y$ with $dt = 10^{-3}$ is used to generate the ground truth results. Open symbols correspond to traditional numerical integrators and closed symbols are results from the $\bar{R}$ integrator. Open circles and squares represent the results of using the traditional Verlet integrator ($Y$) and its equivalent operator (Verlet-OP or $\bar{V}$) with $dt \geq 10^{-2}$. Open triangles and pentagons represent the errors obtained with 4th order Runge-Kutta (RK4) and 4th order Adams-Bashforth-Moulton (ABM4) respectively. Closed squares, triangles, and diamonds represent errors incurred using $\bar{R}$ with sequence length $S_R = 3, 4$ and 5 respectively.

‡ vjadhoa@iu.edu


