Physics-Driven Modelling of the Schrödinger Equation

Bentley Carr

ABSTRACT

Quantum mechanics is governed by the Schrödinger equation which has few analytical solutions and can only be solved generally with slow error-prone numerical methods. We detail two machine-learning models which are able to predict solutions to the 1D Schrödinger equation for a particle confined to a box under the influence of an arbitrary potential. We present the data-driven model which is trained from numerical solutions and the physics-driven model which discovers solutions on its own during training with only knowledge of the governing PDE and boundary conditions. We find that both methods are able to reproduce solutions to the Schrödinger equation to a good approximation. The physics-driven model demonstrates better performance than the data-driven model for simple input vectors but the data-driven model is better at generalising to more complicated initial state and potentials. In particular the physics-driven model struggles learning fast dynamics of the Schrödinger equation. We compare the physics-driven model to the numerical Runge–Kutta method and we demonstrate that our model is able to predict solutions 1125 times faster than the numerical method while demonstrating similar performance.

I. INTRODUCTION

Quantum mechanics is a fundamental theory which provides an almost-complete description of nature [1]. Almost all fields from condensed matter to optics rely on this description to accurately predict the behaviour of their systems [2, 3]. The state of a quantum system is governed by a state vector, sometimes called the wavefunction, whose time evolution is governed by the Schrödinger equation [4], given in 1D for a particle of unit mass in natural units ($\hbar = 1$) as

$$-\frac{1}{2}\frac{\partial^2\psi(x,t)}{\partial^2 x} + V(x)\psi(x,t) = i\frac{\partial\psi(x,t)}{\partial t}. \qquad (1)$$

Equation 1 provides unitary evolution of the state vector meaning that the wavefunction's normalisation is constant in time [4]. The dynamics of a quantum system is governed by the potential term $V(x)$, however the Schrödinger equation is only analytically solvable for a small subset of potentials [4]. As a result, numerical methods, such as the finite difference method, are required to solve the Schrödinger equation for a general potential. These are typically slow [5, 6] and do not scale well to higher-dimensional or multiparticle systems [7].

Deep learning is a tool to develop function approximators to data by training parametric models [8, 9]. We train these parameters by minimising a loss function through gradient descent; we differentiate our loss function with respect to our parameters and we can update our parameters to move towards a smaller value of the loss function in small steps [10]. Multilayer feedforward neural networks with only one hidden layer have been shown to be capable of learning any arbitrary continuous function [11].

Different approaches to machine learning include supervised learning and unsupervised learning [8]. Supervised learning uses a labelled dataset to train a model to predict the output for new data. The dataset consists of N pairs of the form $(\boldsymbol{X}_i, \boldsymbol{Y}_i)$ where $\boldsymbol{X}_i$ is the input data and $\boldsymbol{Y}_i$ is the target data. Our model can be written in terms of some parameters $\boldsymbol{\theta}$ as $\mathcal{F}(\boldsymbol{X}, \boldsymbol{\theta})$. Supervised learning aims to find the parameters $\boldsymbol{\theta}$ which best reproduce the training data, $\mathcal{F}(\boldsymbol{X}_i, \boldsymbol{\theta}) \approx \boldsymbol{Y}_i$. This is done by using gradient descent to minimise the mean squared error (MSE) loss function [10],

$$\mathcal{L}_{\text{MSE}}(\boldsymbol{Y}, \mathcal{F}(\boldsymbol{X}, \theta)) = \frac{1}{N}\sum_{i=1}^{N} \|\boldsymbol{Y}_i - \mathcal{F}(\boldsymbol{X}_i, \theta)\|^2. \qquad (2)$$

During unsupervised learning we train a model on unlabelled training data. We feed the model a set of input data $\boldsymbol{X}$ and the model is able to learn patterns in the data without explicit knowledge of the true solution [8].

Physics-informed neural networks (PINNs) are a special class of function approximators used to embed physical laws into the learning process and are used from parameter estimation [12] to discovering solutions to partial differential equations (PDEs) [13]. PINNs are able to enforce a PDE of the form $g(u, \partial_i u, \cdots) = 0$ with solution $u(t, x)$ by adding a term to the loss function of the form,

$$\mathcal{L}_{\text{PDE}}(\boldsymbol{X}) = \|g(\hat{u}, \partial_i \hat{u}, \cdots)\|^2, \qquad (3)$$

where $\hat{u}$ are the predicted values of $u(t, x)$ given an input vector $\boldsymbol{X}$. The trained model can be used to predict solutions of the system of equations for new inputs, without the need to re-train the model.

We aim to model the 1-dimensional Schrödinger equation for a particle confined to a box under the influence of an external potential. We present two different models which we will call the *data-driven model* based on supervised techniques and a novel *physics-driven model* inspired by PINNs. We characterise our models by comparing them to finite difference methods and determining whether they can reproduce analytical solutions to the Schrödinger equation. We find that we can use our knowledge from physics to increase the performance of machine learning models and we can use machine learning models to provide alternative, faster methods to simulate physical systems.

Section II outlines our method. In particular we outline our model's architecture and detail our training method for the data-driven and physics-driven models. In Section III we present the results from training the data-driven and physics-driven models and compare the abilities of each to generalise to a larger set of input vectors. We then give a detailed comparison of our physics-driven model to the numerical Runge–Kutta model. Key results are summarised in Section IV alongside the prospective applications of the machine learning techniques demonstrated. Appendix A details our computational setup and Appendix B gives a summary of all the experiments we perform. Appendix C gives further details from training our models. We present an alternate procedure for training the physics-driven model in D. In Appendix E we outline the dynamics of the sloped potential which we use to provide further characterisation of our models in Appendix F.

II. METHOD

A. MODEL ARCHITECTURE

The 1D Schrödinger equation for a particle of unit mass in natural units is given by Equation 1. For a particle confined in the box of unit width, the wavefunction is subject to boundary conditions $\psi(0,t) = \psi(1,t) = 0$. Given an initial condition $\psi(x,0) = \psi_0(x)$, the time evolution of the wavefunction $\psi(x,t)$ is governed by the PDE of Equation 1 with Dirichlet boundary conditions

$$\psi(0,t) = \psi(1,t) = 0, \quad \psi(x,0) = \psi_0(x). \tag{4}$$

We aim to solve this PDE on the region $x \in [0,1]$ and $t > 0$.

We model our solution using a multilayer perceptron (MLP) which we use to approximate the function $f(\boldsymbol{X})$, which gives the solution the Schrödinger equation at a time x and t given an initial state ψ_0 and a potential V. Our input vector $\boldsymbol{X}$ is of length 302 and consists of the position to evaluate the wavefunction x, the time to evolve the state by t, the real and imaginary parts of the initial state $\vec{\psi}_0$, and the potential $\vec{V}$. We generate the vectors $\vec{\psi}_0$ and $\vec{V}$ by sampling the initial state and the potential on a grid of size 100. Our solution $f(\boldsymbol{X})$ gives us the value of the wavefunction,

$$f\left(x, t, \mathrm{Re}\left[\vec{\psi}_0\right], \mathrm{Im}\left[\vec{\psi}_0\right], \vec{V}\right) = \begin{pmatrix} \mathrm{Re}\left[\psi(x,t)\right] \\ \mathrm{Im}\left[\psi(x,t)\right] \end{pmatrix}. \tag{5}$$

We choose to output the value of the wavefunction at a single position. This forces our output wavefunction to be continuous in space and allows us to calculate derivatives using automatic differentiation [14].

We give a pictorial representation of our MLP in Figure 1. Our MLP consists of our input layer, 2 hidden layers each with 500 nodes, and the output layer. The value of the nodes in each layer is determined by the expression [10]

$$x_i^{(n+1)} = \mathrm{softplus}(W_{ij}^{(n)} x_j^{(n)} + b_i^{(n)}). \tag{6}$$

The activation function (softplus) results in non-linear behaviour as the input is passed through each layer of the MLP. We use a softplus activation function as it behaves similarly to the rectified linear unit (ReLU) activation function, but provides an output which is everywhere continuous and differentiable which we expect from our output wavefunctions [4]. A smooth output also prevents divergent second derivatives when using physics-driven training.

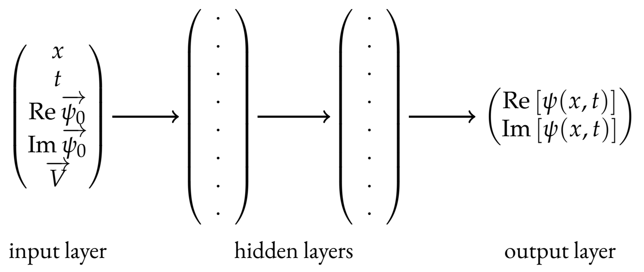

FIG. 1. Pictorial representation of our network architecture. Our input layer is of size 302 and consists of the position x, model time t, the real and imaginary parts sampled on a size 100 grid $\vec{\psi}_0$ and the potential sampled on a size 100 grid $\vec{V}$. The input vector is passed forward through two hidden layers of size 500 before reaching the output layer. The output layer is a 2-component vector which gives the real and imaginary component of the solution at the position x and model time t.

B. DATA-DRIVEN TRAINING

During training, our data-driven model uses pairs of the form $(\boldsymbol{X}_i, \boldsymbol{Y}_i)$ with the $\boldsymbol{X}_i$'s generated from an initial wavefunction and potential, $\vec{\psi}_0$ and $\vec{V}$. The input vector is a 302-component vector and is given by $\boldsymbol{X}_i = \left(x, t, \mathrm{Re}\,\vec{\psi}_0, \mathrm{Im}\,\vec{\psi}_0, \vec{V}\right)$. The target vector $\boldsymbol{Y}_i$ is a 2-component vector which gives the real and imaginary components of the solution the Schrödinger equation with initial state $\psi_0(x)$ in a potential $V(x)$ evaluated at a position x and time t. The target vector $\boldsymbol{Y}_i$ is written explicitly as $\boldsymbol{Y}_i = (\mathrm{Re}\,\psi(x,t), \mathrm{Im}\,\psi(x,t))$.

The initial state $\vec{\psi}_0$ is generated from Fourier composition. This is because the Fourier basis is the natural basis to use given that the wavefunction must be zero at the boundaries of the box. We sample real coefficients $\{a_k\}$ from a uniform distribution and then the initial state $\psi_0(x)$ is given by,

$$\psi_0 = \mathcal{N} \sum_{k=0}^{n_0} (a_k + i b_k) \sin(k\pi x), \tag{7}$$

where our wavefunction is normalised such that the normalisation constant $\mathcal{N}$ is given by

$$\mathcal{N} = \sqrt{\frac{2}{\sum_{k=0}^{n_0} \left(a_k^2 + b_k^2\right)}}. \tag{8}$$

The initial state $\psi_0(x)$ is sampled on a uniformly-spaced grid of size 100 to obtain the vector $\vec{\psi}_0$ which we add to our input vector $\boldsymbol{X}_i$.

The potential $V(x)$ is sampled from a Taylor expansion about the centre of the box. We sample real coefficients $\{c_k\}$ from a normal distribution and the potential is given by,

$$V(x) = K \sum_{k=0}^{N_V} c_k \left(x - \tfrac{1}{2}\right)^k, \tag{9}$$

where we change the value of K to vary the strength of the potential. The potential $V(x)$ is sampled on a uniformly-spaced grid of size 100 to obtain the vector $\vec{V}$ which we add to our input vector $\boldsymbol{X}_i$. We refer to the initial state-potential pair as the *initial system*.

For each initial system we generate, we find an approximate solution to the Schrödinger equation through numerical integration. This is done by sampling the wavefunction on a uniform grid of size $1/\Delta x$. We denote the wavefunction and potential at a grid position i as ψ_i and V_i respectively. We approximate the second derivatives in Equation 1 using the central approximation [15] to give,

$$\begin{aligned}\frac{\partial \operatorname{Re}\psi_i}{\partial t} &= -\frac{1}{2}\frac{\operatorname{Im}\psi_{i+1} - 2\operatorname{Im}\psi_i + \operatorname{Im}\psi_{i-1}}{\Delta x^2} + V_i \operatorname{Im}\psi_i,\\ \frac{\partial \operatorname{Im}\psi_i}{\partial t} &= \frac{1}{2}\frac{\operatorname{Re}\psi_{i+1} - 2\operatorname{Re}\psi_i + \operatorname{Re}\psi_{i-1}}{\Delta x^2} - V_i \operatorname{Re}\psi_i.\end{aligned} \tag{10}$$

We enforce boundary conditions by setting $\psi_i = 0$ outside the box when we calculate the second space derivatives at the edge of the box. We have reduced our PDE to the couple ODEs of Equation 1 which we solve by Runge–Kutta numerical integration [16].

For each initial system we find the numerical solution $\psi(x,t)$. Each numerical solution is sampled on a grid at N_x positions and N_t times. This process is depicted in Figure 2. Therefore a single initial system gives rise to N_xN_t training data pairs. The simulation grid size $1/\Delta x$ need not be the same size as the sampling size N_x. Therefore we can perform the numerical integration on a large grid size to get more accurate numerical simulations and downsample the solution to the training grid size so that we are not exposing our model to input vectors which are too similar.

It becomes a difficult balancing act of choosing the optimal values of the sampling sizes N_x and N_t and the number of initial systems N_0. The length of the training data is given by the product $N_0N_xN_t$. We fix the length of our training data to be 1×10^6 due to memory constraints. If the number of initial systems N_0 is too small then the model will not be exposed to enough different initial systems to be able to generalise to new initial systems. However if the sampling size N_xN_t is too small then the model will only be exposed to the input vectors at a fixed discrete set of positions and times $\{x_i\}$ and $\{t_i\}$. We would expect in this case the model would be able to give good predictions at this fixed discrete set of points but may struggle at positions and times in the continuum of values between these discrete values.

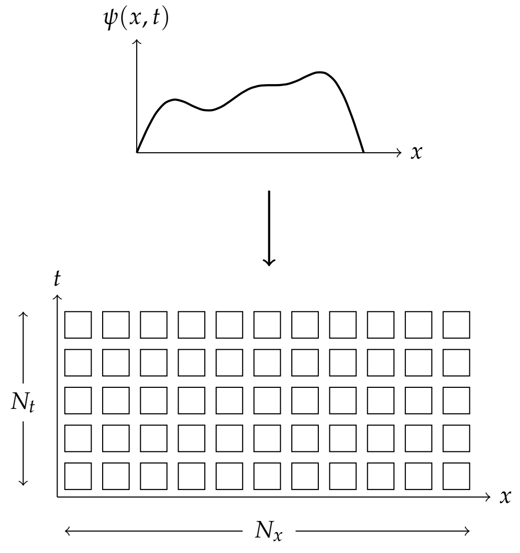

FIG. 2. Depiction of sampling numerical solution to the Schrödinger equation for a single initial system used to generate the dataset for data-driven training. The solution $\psi(x,t)$ from numerical integration is sampled on a grid at various positions and times. We sample at N_x positions and N_t times, so a single numerical solution gives rise to N_xN_t training pairs.

We present two improvements which mitigate the above effects and characterise their impact on the model. We refer to the first improvement as *random sampling*. For each numerical solution, we select the sampling position and times from a continuous uniform distribution. This means that in the limit as the training data length becomes arbitrarily large, the set of positions and times the model is exposed to during training becomes a continuous interval, even for arbitrarily small values of N_x and N_t.

We make a second further improvement by exploiting the global phase invariance of quantum mechanics. Consider an initial system with initial state $\psi_0(x)$ under the influence of some potential and solution at a position x and time t be $\psi(x,t)$. Then the initial and final state are both covariant under global phase transformations [4]

$$\forall\, \theta \in [0,2\pi),\ \psi_0(x) \mapsto e^{i\theta}\psi_0(x) \Rightarrow \psi(x,t) \mapsto e^{i\theta}\psi(x,t). \tag{11}$$

At each epoch during training we sample a set of $\{\theta_i\}$ from a uniform distribution from 0 to 2π. The initial state and solution in each training data pair undergoes phase rotations by different angles. This increases the set of the input vectors the model is exposed to during training. We refer to this improvement as performing *random phase transformations*.

We train our model by minimising the MSE loss given by Equation 2. This is achieved through gradient descent for which we use the Adam optimiser [17]. We lower the learning rate by an order of magnitude when the training loss fluctuations drop below a threshold [18]. We give more details about the parameters used for training in Appendix C.

C. PHYSICS-DRIVEN TRAINING

Our data driven training method relies entirely on numerical solutions of the Schrödinger equation. Such numerical solutions are slow to calculate and suffer from numerical errors due to being sampled on a finite-sized grid. We present the physics-driven training method where we enforce our knowledge of physics to provide an alternative method to train our model. We add five new terms to our loss function and we weight each term by some value which we call the *weightings* of the loss function component. We add a term which we call the *differential loss* which enforces Equation 1, and two terms called the *boundary condition loss* and the *initial condition loss* which enforce the Dirichlet boundary conditions of Equation 4. We add two further terms; the *normalisation loss* enforces the wavefunction's normalisation and the *energy loss* enforces conservation of energy. The differential loss, boundary condition loss and initial condition loss are required to define the unique solution to the Schrödinger equation; the normalisation loss and energy loss are extra conditions enforcing properties of the Schrödinger equation used to guide training.

Many terms in the loss function require evaluation of derivatives of the model's output of the form $\partial f(\boldsymbol{X})/\partial X^{\alpha}$. These are evaluated using the finite difference symmetric approximation [15] using a spacing of 1×10^{-5} for first order derivatives and 1×10^{-2} for second order derivatives. We find that these give similar results as using automatic differentiation while halving training time and using 5.5 times less memory.

The differential loss is given by the form of Equation 3 inspired by PINNs. We define $\hat{\psi}(x,t)$ such that the output of the model $f(\boldsymbol{X})$ is given by $\left(\operatorname{Re}\hat{\psi}(x,t), \operatorname{Im}\hat{\psi}(x,t)\right)$. The differential loss enforcing Equation 1 written out explicitly in terms of real and imaginary components is

$$\mathcal{L}_{\text{dt}}(\boldsymbol{X}) = \left[\partial_t \operatorname{Re}\hat{\psi} + \frac{1}{2}\partial_x^2 \operatorname{Im}\hat{\psi} - V(\boldsymbol{x}) \operatorname{Im}\hat{\psi}(\boldsymbol{x},t)\right]^2 + \left[\partial_t \operatorname{Im}\hat{\psi} - \frac{1}{2}\partial_x^2 \operatorname{Re}\hat{\psi} + V(\boldsymbol{x}) \operatorname{Re}\hat{\psi}(\boldsymbol{x},t)\right]^2. \quad (12)$$

The boundary condition and initial condition losses enforce the Dirichlet boundary conditions of Equation 4. We can write these respectively in terms of real and imaginary components as

$$\mathcal{L}_{\text{bc}}(\boldsymbol{X}) = \left[\operatorname{Re}\hat{\psi}(0,t)\right]^2 + \left[\operatorname{Re}\hat{\psi}(1,t)\right]^2 + \left[\operatorname{Im}\hat{\psi}(0,t)\right]^2 + \left[\operatorname{Im}\hat{\psi}(1,t)\right]^2, \quad (13)$$

$$\mathcal{L}_{\text{ic}}(\boldsymbol{X}) = \left(\operatorname{Re}\hat{\psi}(x,0) - \operatorname{Re}\psi_0(x)\right)^2 + \left(\operatorname{Im}\hat{\psi}(x,0) - \operatorname{Im}\psi_0(x)\right). \quad (14)$$

We calculate the initial state $\psi_0(x)$ at an arbitrary value of x by using linear interpolation of the sampled initial state $\vec{\psi}_0$. To calculate these two terms we replace the value of x or t in the input vector $\boldsymbol{X}$ with the value of x or t at the boundary, so different input vectors with the same initial state and potential gives rise to the same values for the two losses.

Equation 1 enforces unitary evolution of the wavefunction meaning that the wavefunction normalisation is conserved with time. The normalisation loss enforces wavefunction normalisation and is given explicitly in terms of real and imaginary parts by,

$$\mathcal{L}_{\text{norm}} = \left(\left[\int \mathrm{d}x' \left(\operatorname{Re}\psi(x',t)\right)^2 + \left(\operatorname{Im}\psi(x',t)\right)^2\right] - 1\right)^2. \quad (15)$$

We evaluate the integrals using Gaussian quadrature, a very accurate method of 1-dimensional integration given by a weighted sum of function values, calculated from the zeros of the Legendre polynomials [15]. We use Gaussian quadrature evaluated at seven points which is guaranteed to give an exact integral for polynomials of up to degree 13.

Furthermore, Ehrenfest's theorem predicts that the energy expectation value is conserved with time [4]. The energy loss enforces conservation of energy expectation value. We define it as

$$\mathcal{L}_{\text{energy}} = \left(1 - \frac{\langle E\rangle_t}{\langle E\rangle_0}\right)^2. \quad (16)$$

The energy expectation value at a time t is given by [4]

$$\langle E\rangle_t = \int \mathrm{d}x' \, \tfrac{1}{2}\left[\partial_x \operatorname{Re}\hat{\psi}(x',t)\right]^2 + \tfrac{1}{2}\left[\partial_x \operatorname{Im}\hat{\psi}(x',t)\right]^2 + V(x')\left(\left[\operatorname{Re}\hat{\psi}(x',t)\right]^2 + \left[\operatorname{Im}\hat{\psi}(x',t)\right]^2\right), \quad (17)$$

where we have integrated by parts from the usual expectation value to get an integrand which only depends on first order spatial derivatives so is more numerically stable. This integral is calculated using Gaussian quadrature [15] evaluated at seven points. The energy expectation value of the initial state is,

$$\langle E\rangle_0 = \int \tfrac{1}{2}\left[\partial_x \operatorname{Re}\psi_0(x')\right]^2 + \tfrac{1}{2}\left[\partial_x \operatorname{Im}\psi_0(x')\right]^2 + V(x)\left(\left[\operatorname{Re}\psi_0(x)\right]^2 + \left[\operatorname{Im}\psi_0(x)\right]^2\right). \quad (18)$$

The input vector contains the sampled initial wavefunction $\vec{\psi}_0$ sampled at 100 points. We calculate the first order spatial derivative sampled at 99 points using the the symmetric approximation [15] evaluated between two grid points. Since we easily obtain the wavefunction and its derivative sampled at a large number of points we can evaluate the integral by using the trapezoidal rule [15].

If we remove the data-driven MSE loss from the total loss function, we do not require a target vector $\boldsymbol{Y}_i$ since our total loss function only depends on the input vectors $\boldsymbol{X}_i$. We will mostly work in the case where the data-driven MSE loss is removed. In generating our dataset $\{\boldsymbol{X}_i\}$ we generate our sampled initial state $\vec{\psi}_0$ and sampled potential $\vec{V}$ in the same method as described in Section II B. However, since we do not have to generate a target vector $\boldsymbol{Y}_i$ we need not sample the solution at different x and t. Rather, during training we produce an input vector $\boldsymbol{X}$ by taking an already-generated initial state and potential and we sample x and t uniformly on the intervals $[0, 1]$ and $[0, t_{\max}]$ respectively. This means at every epoch the model will be exposed to different input vectors.

We make further improvements to expose the models to a larger set of input vectors during training. As discussed in Section II B we can perform random phase transformations at each epoch. We can also apply this to the physics-driven model as rotating the phase of the initial wavefunction by a randomly-sampled angle will yield another normalised initial state. However we can go further than this. We do not need to calculate the corresponding target vector for each input vector; we only need to provide the model with valid input vectors. We can produce another set of input vectors at each epoch by creating new potentials from linear combinations of existing potentials and pairing up each potential with a different input state. This allows us to expose the model to infinite set of input vectors at each epoch and provides a faster alternative to continuously generating new input vectors and potentials.

Each element of our loss function is scaled by some weighting. The set of possible combination of weights is the hyperparameter space. We train our hyperparameters by minimising the validation loss. The validation data is generated in the same way as the data-driven training dataset described in Section II B, but fewer numerical integrations are required and the numerical simulations are performed on a larger grid of size 300.

The validation loss consists solely of the MSE loss of the validation dataset. The validation loss is not seen by the model during training and is solely used to train the hyperparameters.

During training, the neural network discovers solutions that satisfy the Schrödinger equation along with the Dirichlet boundary conditions. As stated in [13]: "this discovery is done without using any explicit information about the true solution; only the PDE and the boundary conditions that must be satisfied are provided". We therefore classify the physics-driven training method as an unsupervised learning method.

D. BENCHMARKING

Finally, we benchmark our models against numerical finite difference models. We define the *model time* as the value of t at which we calculate an approximation to the solution $\psi(x, t)$. We compare the performance of the models by computing the MSE error between the output of the model and an analytical solution against model time for a particle in a box. We compare the speed of our model against the numerical model by calculating the average time it takes to evaluate the solution as a function of model time. We choose to run our models on the CPU so that they are run on the same device as the numerical models, but our models can be run even faster on the GPU. Further details about the computational setup used for benchmarking is given in Appendix A. The results of the numerical model are calculated using the methods described in Section II B.

We train our models on the time interval $t \in [0, t_{\max}]$. We can predict the solution to the Schrödinger equation at a point x on this time interval by passing a single input vector through the neural network. We can obtain the wavefunction sampled at a series of points $\{x_i\}$ by passing a set of input vectors through the neural network once. However we cannot simply predict the wavefunction at a later time $t > t_{\max}$ by a single pass through the neural network. Instead we predict the solution at a time $t = \tau < t_{\max}$ to get a solution $\hat{\psi}(x, \tau)$. We then create an input vector where the initial state is this solution $\hat{\psi}(x, \tau)$ and we pass this through the neural network again to yield to predict the solution $\hat{\psi}(x, 2\tau)$ at $t = 2\tau$. By iteratively calculating how the state evolves after a time $\tau < t_{\max}$ we can make predictions for the solution at any arbitrarily large time. We use this method when benchmarking our models against the numerical method within the domain $t > \tau$.

III. RESULTS AND DISCUSSION

We outline how the data-driven and physics-driven models are trained and show that both methods are able to reproduce solutions to the Schrödinger equation to a good approximation. We discuss the capabilities of each model to generalise to a larger set of input vectors. Alternate methods of training are then provided before we give a detailed comparison between the physics-driven model and the numerical Runge–Kutta method.

We train a data-driven model from training data generated from five Fourier modes and potential with RMS strength of 1.5 generated from polynomials of up to degree three. This dataset is of total length 1×10^6 made up of 1×10^4 initial systems, each randomly sampled at 10 positions and 10 times. The target vectors are generated from numerical simulation on a grid of size 100. We train our model with a learning rate of 1×10^{-3} where the learning rate is reduced when the training loss plateaus. We apply random phase transformations at each epoch. We find that the model trains within 160 epochs. Further details about training this model is given in Appendix C.

We train a physics-driven model similarly from training data generated from five Fourier modes and potential with RMS strength of 1.9 generated from polynomials of up to degree three. However, since we do not need slow numerical integration to generate our training data, our training data is of total length 1×10^6 and consists of 1×10^6 initial systems.

We perform a hyperparameter search to find the optimal weightings of each loss function element. We find that best performance is achieved when the normalisation loss and energy loss are removed. We attribute this to being due to the Gaussian quadrature only sampling the wavefunction at a few discrete set of positions, so only the value of the wavefunction at these discrete points contribute to the gradient. We find that the optimal weightings occur when the boundary condition and initial conditions losses have unit weighting, and when the differential loss has a weighting of

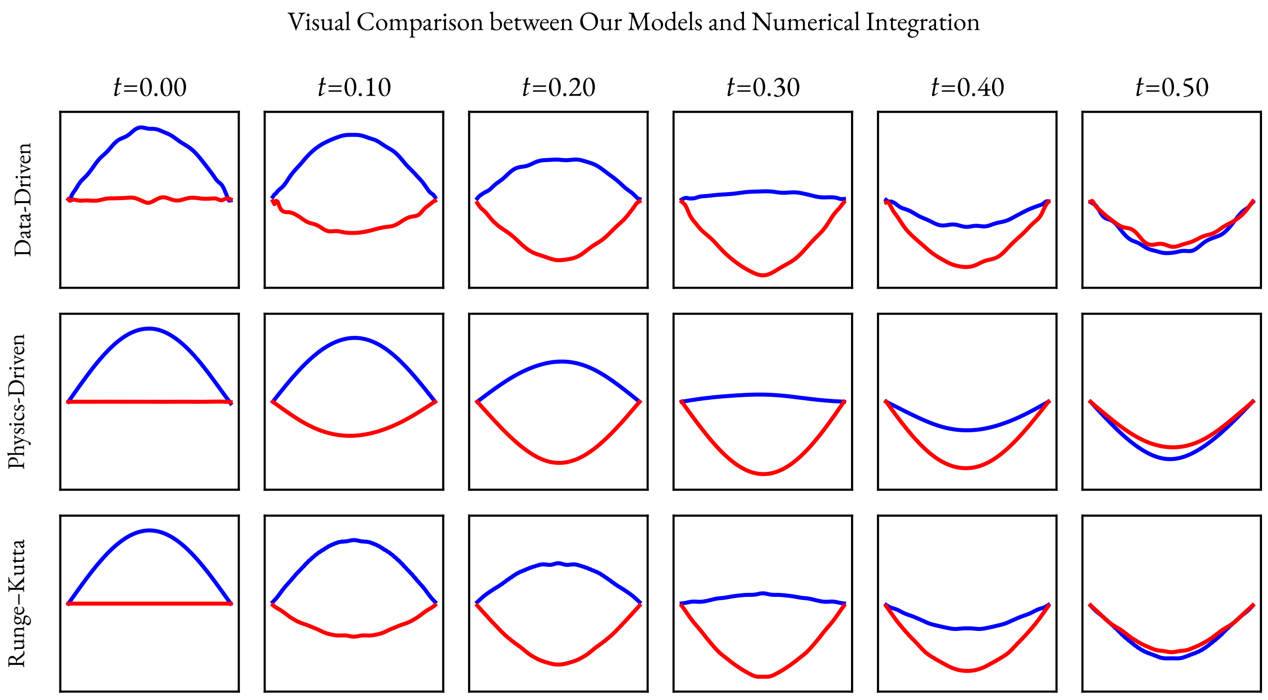

FIG. 3. Solutions to the Schrödinger equation for the ground state of the zero potential system. The real component of the solution is presented in blue and the imaginary component in red. The solutions are evaluated at six times between 0.00 and 0.50. The data-driven model is shown on top, with the physics-driven model below and the Runge–Kutta method on the bottom. All methods are able to reproduce the basic features of the solution. Both the numerical method and the data-driven models display short-wavelength fluctuations, with the data-driven model displaying greater fluctuations. The physics-driven model does not display such fluctuations.

1×10^{-3}. An improper set of hyperparameters result in a decay of the wavefunction's normalisation over time. If the differential loss is weighted too heavily, the boundary conditions are not well satisfied and we find the wavefunction 'leaks out the box', resulting in a drop in the normalisation. If the boundary condition loss is weighted too heavily, then the model does not fully obey the Schrödinger equation; the wavefunction's evolution is not unitary so the normalisation decays over time.

We initially train our physics-driven model using the optimal weightings over a constant time interval $t \in [0, t_{\max}]$. We consider the case where the potential is set to zero everywhere. We find that the physics-driven model has trouble learning the fast-dynamics of the wavefunction; exposing the model to higher Fourier modes during training resulted in a large drop in normalisation over time, even when predicting the time evolution for lower Fourier modes. Significant performance improvements occur when we reduce the time interval cutoff $t_{\max}$ during training. We attribute this to the fact that larger times are further away from the initial condition $t = 0$. During training the model has no knowledge of the information at an arbitrary time t, only on the boundary. To work out the solution at a time t', the model has to build up knowledge of the solution at all times $t < t'$. Increasing the complexity of initial states results in faster dynamics which the model is not able to learn as easily. Reducing the cutoff time $t_{\max}$ reduces this effect as the model does not have to learn as much of the dynamics to evaluate the solution at an arbitrary time on the training interval.

We find that during physics-driven training each epoch takes approximately 2.2 times longer than during data-driven training.

The data-driven and physics-driven models described above are compared qualitatively to the numerical model in Figure 3. The time evolution of an initial state of the zero potential ground state is calculated using each method and evaluated at times in the interval $[0, 0.5]$. We see that all methods reproduce the basic properties of the stationary state; the state evolves with a constant amplitude and rotating phase. We see that the numerical model is prone to short-wavelength errors which we attribute to errors due to the numerical simulation being performed on a finite grid. The data-driven model is also subject to similar errors which we attribute to being trained using the error-prone numerical solutions. However we see that the physics-driven model is not affected by such short wavelength fluctuations and produces the smoothest results.

The following models are trained from initial states sampled from five Fourier modes and potentials made from a linear combination of polynomials up to a finite degree. We find that our models are both unable to extrapolate to new input vectors made from higher order Fourier modes and polynomials than those exposed to during training. In order to produce a model which we are able to generalise to any arbitrary input vector we need to characterise the capabilities of each model at generalising to a larger

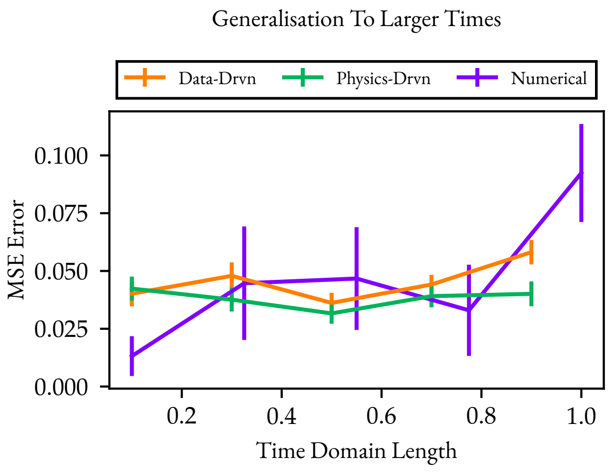

FIG. 4. MSE error between models' output and analytical solution for first excited state of zero potential system against the length of the time domain used during training. The data-driven model errors are given in orange while the physics-driven model errors are given in green.

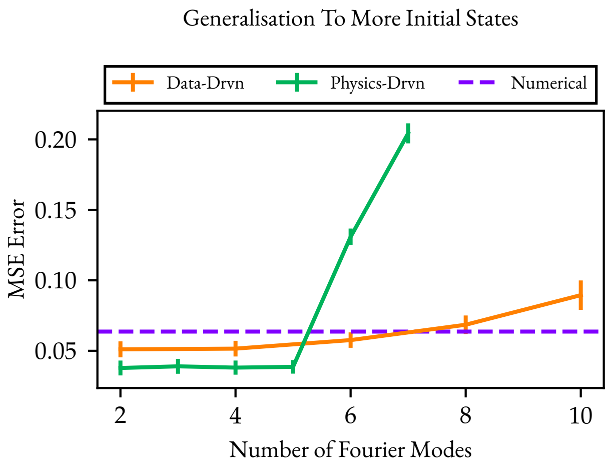

FIG. 5. MSE error between models' output and analytical solution for first excited state of zero potential system against the number of initial state Fourier modes used during training. The data-driven model errors are given in orange while the physics-driven model errors are given in green.

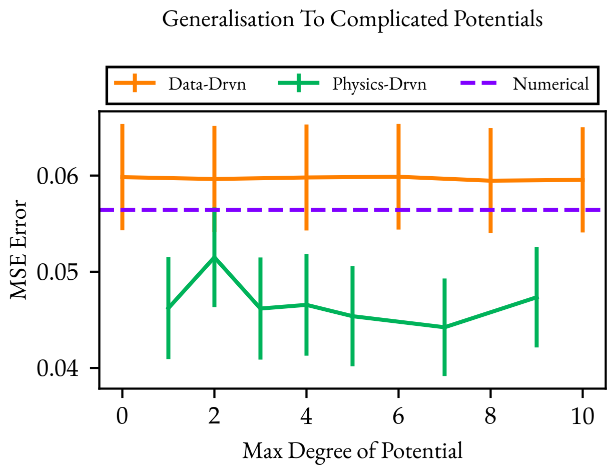

FIG. 6. MSE error between models' output and analytical solution for first excited state of zero potential system against the maximum polynomial degree used to generate the potentials in the weak potential regime. The data-driven model errors are given in orange while the physics-driven model errors are given in green.

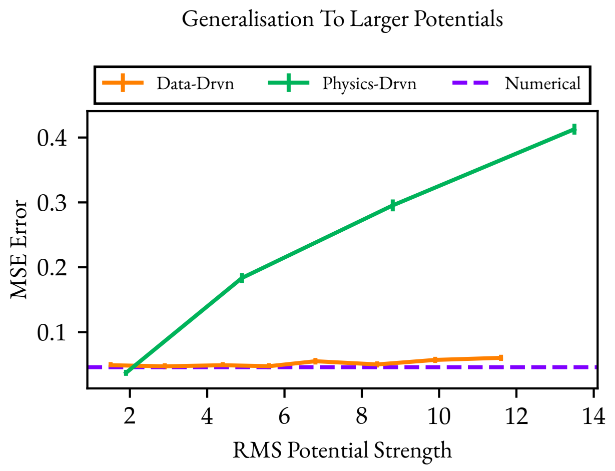

FIG. 7. MSE error between models' output and analytical solution for first excited state of zero potential system against the RMS potential strength. The data-driven model errors are given in orange while the physics-driven model errors are given in green.

set of input vectors. We compare our models' abilities to generalise in four different areas. We assess how our models perform when the time domain is increased, when the maximum number of Fourier modes of the initial states is increased (the number of Fourier modes), when the maximum degree of polynomials used to generate the potentials is increased (the potential complexity), and when the potentials are scaled by some constant factor (the potential strength). The models were trained with the same parameters as described above. For each model we assess its performance by computing the MSE error between its output and an analytical solution based off a superposition of the lowest two energy states of a box with zero potential.

We first compare the abilities of each model to generalise to a larger time domain. We train a set of models where the initial states are composed of five Fourier modes and we do not include a potential and their corresponding performance are given in Figure 4. It is observed that the physics-driven and data-driven models both perform just as well as the numerical model. Performance slowly decreases with a longer time domain.

The abilities of each model to generalise to an input vector consisting of more Fourier modes are explored. We train a set of models on the time domain $t \in [0, 1]$ and their performances are shown in Figure 5. The physics-driven model displays better performance for a low number of Fourier modes. We observe that both models struggle to generalise to an initial state consisting of more Fourier modes, however the physics-driven model struggles significantly more than the data-driven model. For a small number of Fourier modes, both models produce better performance than the numerical model.

We find that the physics-driven model has trouble learning the fast-dynamics of the wavefunction. We attribute this to the fact that larger times are further away from the initial condition $t = 0$. During training the model has no knowledge of the information at an arbitrary time t, only on the boundary. To work out the solution at a time t', the model has to build up knowledge of the solution at all times $t < t'$. Increasing the energy of initial states results in faster dynamics which the model is not able to learn as easily. Reducing the cutoff time $t_{\max}$ combats this effect as the model does not have to learn as much of the dynamics to evaluate the solution at an arbitrary time on the training interval.

We explore the abilities of our models to generalise to input vectors with more complicated potentials. We explore the weak potential regime ($V \lesssim \langle E \rangle$) in which our coefficients in Equation 9 are sampled from a normal distribution with variance 2.0. We plot the performance of each model against the maximum degree polynomial in Figure 6. We find that when trained in the weak potential regime, both the data-driven and physics-driven models are still able to give good solutions to the zero-potential system even when the input vector space is increased. We cannot conclude that our models are able to give good solutions in the case of a non-zero potential, but training them on a larger input vector space does not affect the performance of our solutions for the zero-potential system.

In Figure 7 we present the models ability to generalise to be trained on a set of stronger potentials. We train our models with initial states from three Fourier modes and potentials from polynomials of up to degree three, but we vary the scaling factor. We find that for weak potentials both the physics-driven and data-driven model are able to perform better than the numerical model. However the physics-driven model struggles to generalise to an set of input systems with larger potentials.

From Figures 4-7, we conclude that the data-driven model is better at learning fast dynamics whereas the physics-driven model leads to better performance when trained well.

The data-driven model is straightforward to train whereas training the physics-driven model requires hyperparameter tuning and there are challenges associated with learning fast dynamics. However when properly trained the physics-driven model is able to display smoother solutions and better performance due to not being exposed to error-prone numerical solutions. We make a number of attempts at combining the two methods to produce better results. Training with the data-driven MSE loss combined physics-driven loss function resulted in worse performance than each individual method; hyperparameter searches found that the best trials minimised either the MSE loss weighting, or all of the physics-driven loss element weightings. We tried training the models piecewise; we train a model using data-driven training, followed by physics-driven training intended to fine-tune the solutions. However we find that changing the training method results in the training loss quickly increasing to order unity, effectively undoing any progress made by the data-driven training. We also try an approach based off population based training [19] where the hyperparameters are varied during training. All the methods mentioned above had little effect on the model's performance.

We present an alternate procedure for training the physics-driven model which relies on initially training over a small time interval and then gradually increasing the time domain used during training. We find that this training method increases the model's performance and is able to lead to the model preserving normalisation of states with fast dynamics over a larger time interval. We detail this training method and results in Appendix D.

Finally, we show explicitly that the physics-driven model is able to give the correct features in the case of a sloped potential of gradient 10. We then give a detailed comparison the performance and speed of our model compared to the numerical model.

We consider a sloped potential $V(x) = 10x - 5 - \frac{\pi^2}{2}$. We adjust the slope so that it is zero at the centre of the box, and then we lower the box by an amount so that the particle in a box ground state energy has zero expectation value. This sloped potential does not have an analytical solution, however Ehrenfest's theorem tells us that the expectation value of the observables should obey classical equations of motion. Specifically the position expectation value obeys [4]

$$\frac{\mathrm{d}^2}{\mathrm{d}t^2} \langle x \rangle = - \left\langle \frac{\partial V}{\partial x} \right\rangle . \tag{19}$$

Therefore in the presence of a sloped potential, the quantum mechanical particle will slide down the potential and hit the wall at $x = 0$ before sliding back up the potential to its original position. We discuss the subtleties of these qualitative dynamics in Appendix E. Figure 8 presents the solution to the sloped potential as predicted by the numerical method and the physics-driven model trained with three initial state Fourier modes and a set of linear potentials with RMS strengths of 4.4. We see that the physics-driven model is able to reproduce the behaviour of the wavefunction falling down the slope before bouncing back up to be very good degree.

We give a detailed comparison of the physics-driven model used above to the numerical Runge–Kutta method on a grid size 100. We compare the two models' performance of predicting solutions to the zero potential system by comparison to the true analytical solution. We also compare the models at predicting solutions to the sloped potential in Appendix F, however since the sloped potential does not have analytical solutions we compare our models with a numerical solution evaluated at a larger grid size of 300.

We evaluate the model performance by calculating the MSE error between the model output and the analytical solution. The performance of each model is presented against the model time in Figure 9. Both methods produce greater error when evaluating states

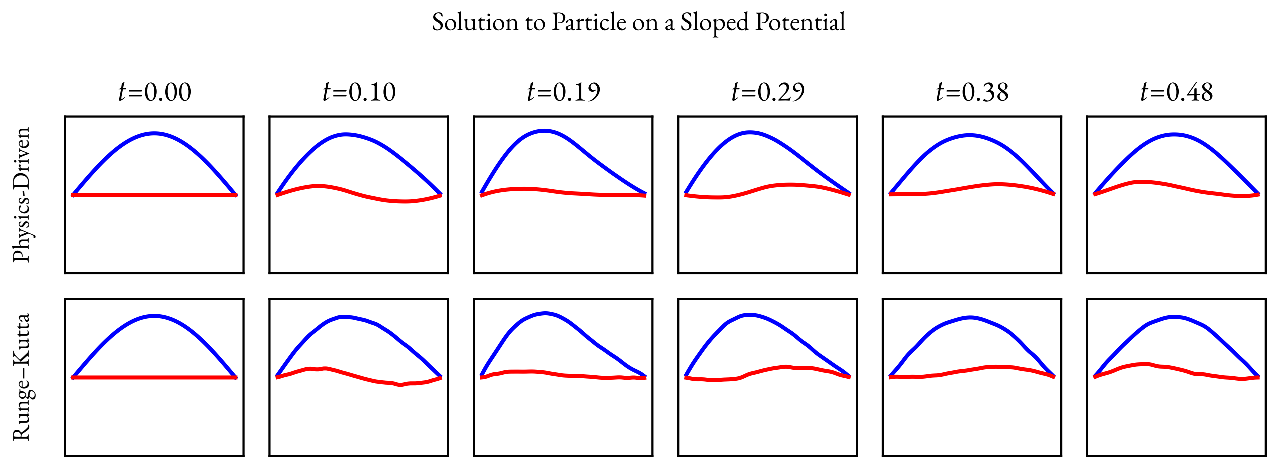

FIG. 8. Solutions to the Schrödinger equation for the sloped potential. The real component of the solution is presented in blue and the imaginary component in red. The solutions are evaluated at six times between 0.00 and 0.48. The physics-driven model is presented on top and the Runge–Kutta method is presented on the bottom. Both methods are able to reproduce the key features of the particle sliding down the potential, however the numerical model displays short-wavelength fluctuations whereas the physics-driven model does not.

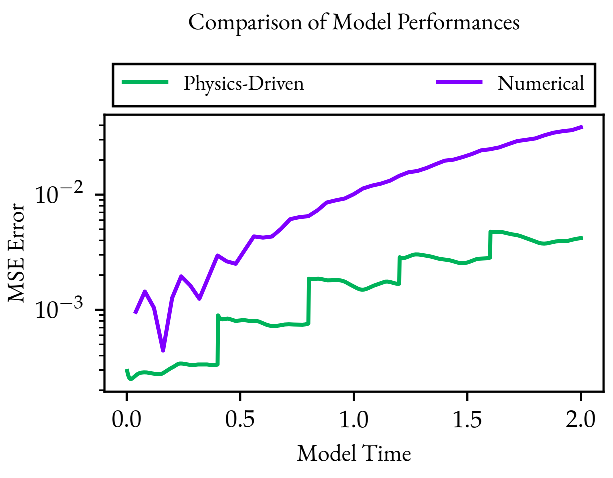

FIG. 9. MSE error between each model's output against the analytical solution for the first excited state of the zero potential system against model time for the physics-driven model and the numerical Runge–Kutta model. The errors of the physics-driven model are given in green while the errors of the numerical model are given in purple.

at a later time. We find that both models perform similarly well, with the physics-driven model performing slightly better than the numerical model. We note that the error in the physics-driven model increases in steps with each extra iteration the model has to perform, but stays roughly constant within a fixed number of iterations.

We quantify the speed of the two models by measuring the evaluation time of each model. We detail the machines used in Appendix A. The evaluation time of each model is presented against the model time in Figure 10a and we present the relative evaluation time in Figure 10b. We find that the physics-driven model is up to 1125 times faster than the numerical Runge–Kutta model at a grid size of 100. We find the evaluation time of the numerical model increases linearly with model time and the evaluation time of the physics-driven model increases in steps with each extra iteration the model has to perform, but remains constant within a fixed number of iterations.

We repeat the detailed comparison of performance of the physics-driven model against the numerical model for the sloped potential in Appendix F. We find similar results; we are able to achieve similar performance to the numerical model and we find that the physics-driven model is up to 1260 times faster than the numerical model.

IV. CONCLUSION AND OUTLOOK

We present two machine learning models which are able to predict solutions to the Schrödinger equation for a single particle in a box with an arbitrary potential. The data-driven model is trained by being exposed to numerical solutions to the Schrödinger equation. The physics-driven model is trained by enforcing the Schrödinger equation and the Dirichlet boundary conditions in the loss function and is able to be trained without being explicitly exposed to any solutions. We find that both methods are able to reproduce solutions to the Schrödinger equation to a good approximation. The physics-driven model is able to provide solutions which more closely match analytical solutions for a limited set of input vectors, however it is difficult to train. In particular it has trouble learning fast dynamics of the wavefunction. We compare our physics-driven model to the numerical model and find our model is 1125 times faster while being able to predict solutions with similar performance to the numerical model. However our models are limited in their capacity to generalise to more complicated initial states and larger potentials.

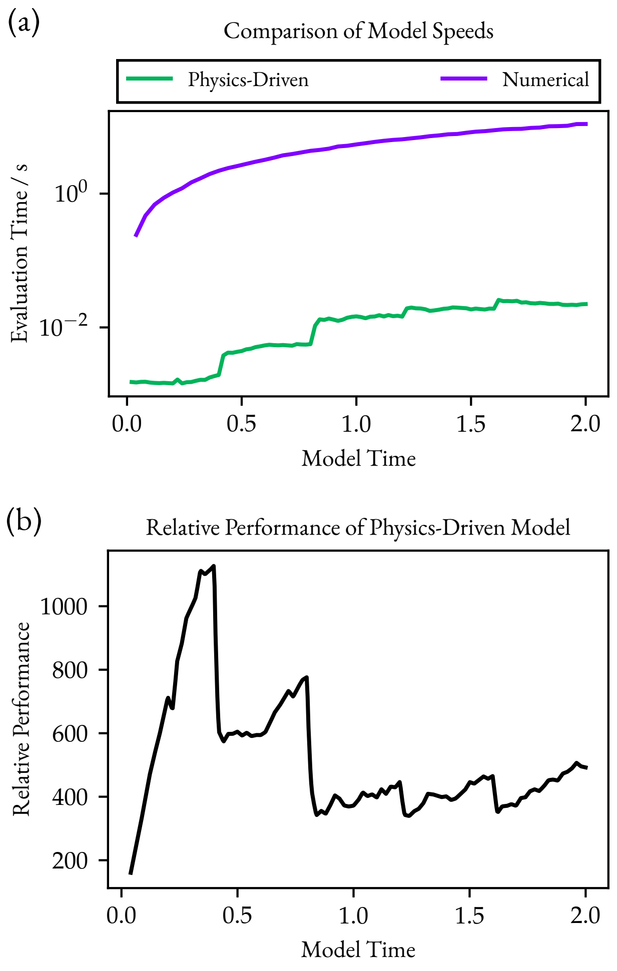

FIG. 10. Evaluation time of the physics-driven model and the numerical Runge–Kutta model against model time for the first excited state of the zero potential system. (a) Absolute evaluation time of the physics-driven model are given in green and the absolute evaluation time of the numerical model is given in purple. (b) The relative evaluation time between the two models is given. At a model time of 0.40, the physics-driven model is 1125 times faster than the numerical model.

In this project we would have liked to explore other ways to aid training of the physics-driven model to explore its full potential, however we were limited by our access to the department's resources. This limited our ability to find the optimal set of loss function weighting to train the physics-driven model, so we were unable to properly train the physics-driven model until relatively recently. Details concerning the limited access to resources and the machines used are given in Appendix A.

We have been able to show that neural networks are capable of learning the time evolution of a quantum state, however there is also interest in learning the energy eigenstate spectrum of quantum systems. Calculations of energy eigenspectra rely on methods such as the variational method [20] which are prone to error accumulation, especially in calculating the wavefunctions of higher energy states. We can indirectly calculate a state's time evolution from the energy eigenstates and knowledge of the energy eigenspectrum leads to predictions of other experimental quantities such as the atomic absorption spectra of light with applications ranging from identifying molecules through spectroscopy [21] to calculating the band structure of electrons in condensed matter systems [2]. It would be interesting to study whether machine learning methods could be used to learn properties of a potential, and then this knowledge could be transferred to a collection of problems from time evolution of quantum states to calculating energy eigenspectra, leading to transfer learning between a related but different set of problems [22]. There is also interest in calculating the time evolution of unbound scattering states without restriction that the particle be confined to a small region of space as we have done.

I would like to thank my supervisor, Professor Pietro Liò and my day-to-day supervisor Alexander Norcliffe for meeting with me on a regular basis to provide their valuable insights and support throughout the year.

REFERENCES

[1] Feynman, Richard Phillips and Leighton, Robert B. and Sands, Matthew L., *Feynman Lectures on Physics Vols. 5 & 6 Commemorative Issue*. Pearson Education, Limited, 1989.
[2] N. W. Ashcroft, *Solid state physics*. Holt, Rinehart and Winston, 1976.
[3] D. F. Walls and G. J. Milburn, *Quantum Optics*. Springer, 2008.
[4] D. J. Griffiths and D. F. Schroeter, *Introduction to Quantum Mechanics*. Cambridge University Press, 2016.
[5] R. Chen, Z. Xu, and L. Sun, "Finite-difference scheme to solve Schrödinger equations," *Physical Review E*, vol. 47, pp. 3799–3802, may 1993.
[6] C. Grossmann, H.-G. Roos, and M. Stynes, *Numerical Treatment of Partial Differential Equations (Universitext)*. Springer, 2007.
[7] K. A. Baseden and J. W. Tye, "Introduction to Density Functional Theory: Calculations by Hand on the Helium Atom," *Journal of Chemical Education*, vol. 91, pp. 2116–2123, nov 2014.
[8] C. M. Bishop, *Neural networks for pattern recognition*. Clarendon Press, 1995.
[9] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, jan 2015.
[10] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.
[11] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, pp. 359–366, jan 1989.
[12] M. Raissi, P. Perdikaris, and G. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational Physics*, vol. 378, pp. 686–707, feb 2019.
[13] R. van der Meer, C. W. Oosterlee, and A. Borovykh, "Optimally weighted loss functions for solving PDEs with neural networks," *Journal of Computational and Applied Mathematics*, vol. 405, p. 113887, may 2022.
[14] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. Red

Hook, NY, USA: Curran Associates Inc., 2019.

[15] W. H. PRESS, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes 3rd Edition*. Cambridge University Press, 2007.

[16] P. Bogacki and L. Shampine, "A 3(2) pair of runge - kutta formulas," *Applied Mathematics Letters*, vol. 2, no. 4, pp. 321–325, 1989.

[17] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014.

[18] Y. Bengio, "Deep learning of representations for unsupervised and transfer learning," in *Proceedings of ICML Workshop on Unsupervised and Transfer Learning* (I. Guyon, G. Dror, V. Lemaire, G. Taylor, and D. Silver, eds.), vol. 27 of *Proceedings of Machine Learning Research*, (Bellevue, Washington, USA), pp. 17–36, PMLR, 02 Jul 2012.

[19] M. Jaderberg, V. Dalibard, S. Osindero, W. M. Czarnecki, J. Donahue, A. Razavi, O. Vinyals, T. Green, I. Dunning, K. Simonyan, C. Fernando, and K. Kavukcuoglu, "Population based training of neural networks," 2017.

[20] R. Borghi, "The variational method in quantum mechanics: an elementary introduction," *European Journal of Physics*, vol. 39, p. 035410, apr 2018.

[21] C. L. Banwell, *Fundamentals for Molecular Spectroscopy*. Trade paperback, McGraw-Hill Companies.

[22] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, pp. 1345–1359, 2010.

APPENDICIES

A. Compuational Setup

We detail the computational hardware used for training and characterisation of models. In particular we discuss difficulties throughout the project concerning getting access to computing power and we detail the hardware used for characterisation of models.

During the early stages of the project we did not have access to department resources and I only had access to my personal computer and Google Colab for training. My personal computer has a NVIDIA® GeForce GTX 1650 Max-Q GPU. We were limited by the 4GB of VRAM in this device as automatic differentiation requires a large amount of VRAM to compute second derivatives. The physics-driven models took a long time to train on this machine so hyperparameter training was on viable on my personal computer. We also had access to Google Colab. We used the paid plan of Google Colab for one month which gave us access to a NVIDIA® Tesla T4 GPU which have more memory than the GPU on my personal computer. This allowed us to perform automatic differentiation to calculate second derivatives and train our models faster. However we were still limited because Google Colab automatically stopped our models from training if they took too long to train. This prevented us from fully training our physics-driven models since they take a long time to train. Further we were still not able to perform hyperparameter searches due to our finite runtime.

On 11 April 2022, just over one month before the deadline, we were given access to the Computational Biology Group GPUs. Normally the group has three GPU machines; `crunchy` is for general use, `cozy` is the Part II, Part III, MPhil and MASt use, and `kiiara` is normally reserved for PhD students. However during the project `cozy` and `crunchy` were broken. Furthermore `kiiara` was prone to regular crashes and the local filesystem was regularly full. Since `kiiara` was the only group-owned GPU machine, it was regularly being used by other Part II, Part III and PhD students, so it was difficult to train models on the machine. Despite this, we were able to perform a limited number of hyperparameter searches on the machine.

We then resorted to using Google Cloud Compute Engine to train most of our models. The Google Cloud Platform free trial gives us USD 400 credits for the University of Cambridge G Suite accounts. With these credits we were able to create our own Linux virtual machine (VM) with access to GPUs. Our VM is a based off a Debian 11 image which has constant access to two NVIDIA® Tesla T4 GPUs. We give our VM 98.2 GB of RAM and it is powered by eight virtual cores of an Intel® Xeon® CPU @ 2.00 GHz.

We create an additional virtual machine which we use for characterising our models. This virtual machine is based off a Debian 11 image and is given 98.2 GB of RAM and is powered by eight virtual cores of an Intel® Xeon® CPU @ 2.00 GHz. It is identical to the VM described above to train our models, however it is not given access to any GPUs. We use this virtual machine for characterising our models and all characterisation is done on the CPUs, giving a fair comparison between the numerical method and our models.

B. Summary of Experiments

We discuss details relating to the implementations of our models and we give a summary all the experiments we performed.

The data-driven models and the physics-driven models were implemented in Python 3.7.12. We use PyTorch 1.11 to run and train our models. Automatic differentiation was performed with `torch.autograd.functional.jvp`. We use `torch.optim.lr_scheduler.ReduceLROnPlateau` with default settings to reduce the learning rate when the training loss plateaus. We stop models from training after the training loss reaches 1×10^{-8}. We use NumPy 1.21.6. Numerical integrations were calculated with SciPy 1.7.3 using `scipy.integrate.solve_ivp` with the `RK23` integration method. We use the default tolerance values. We use this integration method because we find it gave the best performance. Figures were generated with Matplotlib 3.5.2.

Hyperparameter training was performed with Ray Tune with Ray 1.12.0. We use the ASHA scheduler with `ray.tune.schedulers.ASHAScheduler` to stop unsuccessful trials early. During hyperparameter training we use Bayesian optimsation with Dragonfly 0.1.6. We use `ray.tune.schedulers.PopulationBasedTraining` for population based training.

All code used to train models and generate figures are provided at `https://github.com/Yharooer/schrodinger-nn-solver`.

We briefly outline all the experiments performed. A flowchart of experiments performed and the process of producing the final models is given in Figure 11. The first step is to generate our training data. We can generate either the data-driven dataset consisting of input vector target vector pairs $(\boldsymbol{X}_i, \boldsymbol{Y}_i)$ or the physics-driven dataset consisting of only input vectors. The data-driven datasets lead to the data-driven training method detailed in Section II B. We also use the data-driven dataset to perform hybrid training where we include both data-driven MSE and physics-driven PDE loss terms, but we find these do not lead to models which demonstrate good performance. The physics-driven dataset is used for physics-driven training. However physics-driven training requires knowledge of the optimal com-

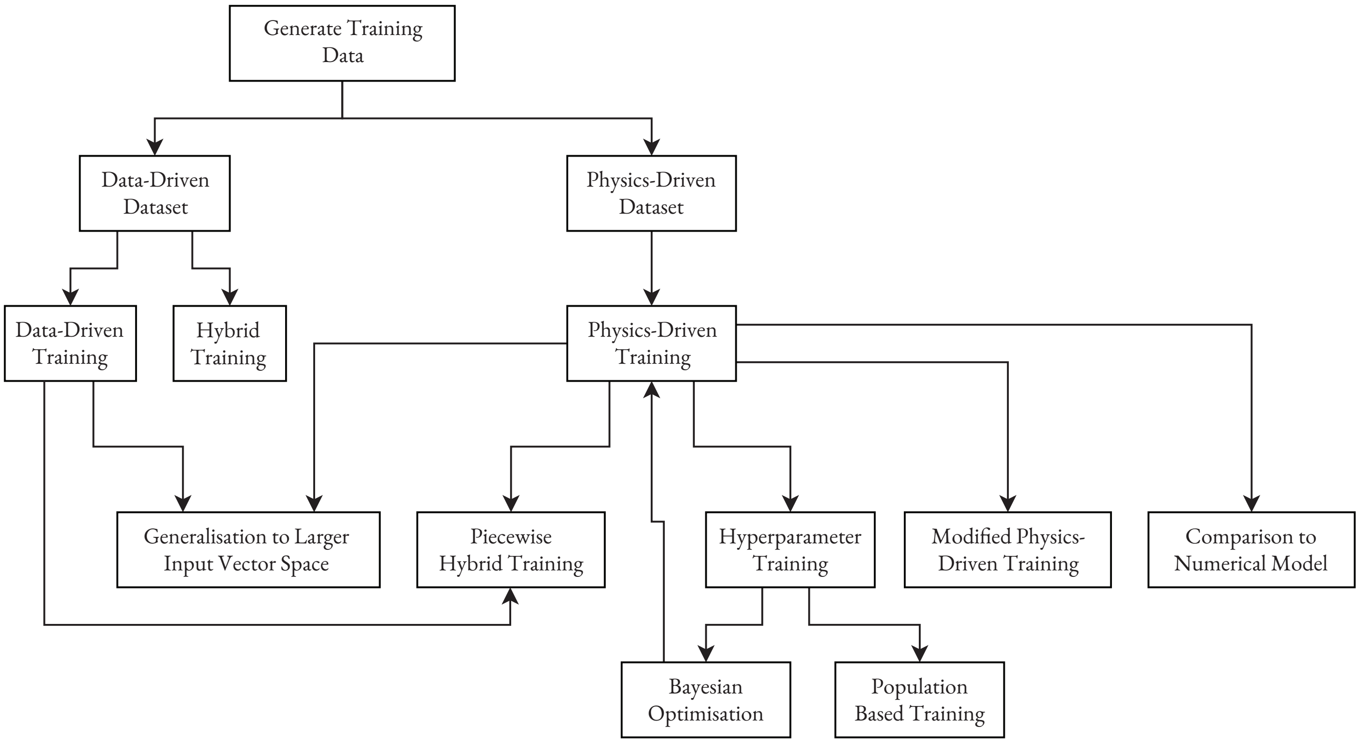

FIG. 11. Flowchart of the experiments performed and process of producing the final models. We generate our dataset which then allows us to perform physics-driven training and data-driven training. We use Bayesian optimsation to perform hyperparameter training which informs the optimal hyperparameters used in our physics-driven training method. We display the variations of training methods used to train our models including piecewise hybrid training, population based training and the modified physics-driven training. Finally we use the physics-driven training to produce a physics-driven model which we compare to the numerical Runge–Kutta method.

bination of the loss function component weights. These are discovered through hyperparameter training where we minimise the MSE loss between our model and our validation data to train our hyperparameters. We use Bayesian optimsation to find the best combination of hyperparameters. We also try the population based training method however we find this does not lead to improvements. We attempt the piecewise hybrid training approach where we perform data-driven pretraining and then fine-tune the model with physics-driven training. However we find that when we change training method the loss spikes upwards, effectively undoing the effects of data-driven pretraining. We present the modified physics-driven training method and find this leads to some improvements at the expense of the model taking much longer to train. We then compare the physics-driven model to the numerical Runge–Kutta method.

C. Training of the Data-Driven and Physics-Driven Models

In this appendix we present the our loss values as a function of epoch number for training of the data-driven model and the physics-driven model.

Firstly, we train a data-driven model from an initial state generated from five Fourier modes and potentials with RMS strength of 1.5 generated from polynomials of up to degree three. The dataset is of length 1×10^6 and is made up of 1×10^4 initial systems. Generating the training data takes approximately 3.6 hours. We train the model with an initial learning rate of 1×10^{-3} and the learning rate is reduced when the training loss plateaus using `torch.optim.lr_scheduler.ReduceLROnPlateau` with default parameters. The learning rate is dropped by a factor of 10 at epochs 75, 200, 225, 240 and 250. The model trains in roughly 160 epochs and training takes roughly 18 minutes. The training loss is presented in Figure 12. The final training loss is 1.49×10^{-3} and the final validation loss is 2.23×10^{-3}.

Secondly we train a physics-driven mode from an initial state generated from three Fourier modes and potentials with RMS strength of 1.5 generated from polynomials of up to degree three. The dataset is of length 1×10^6 and is made up of 1×10^6 initial systems. We apply random phase transformations and mix the pairing up of initial states and potentials at each epoch. Generating the training data takes 5–6 minutes. We train the model with unit weightings of the initial condition and boundary conditions losses, and a weighting of

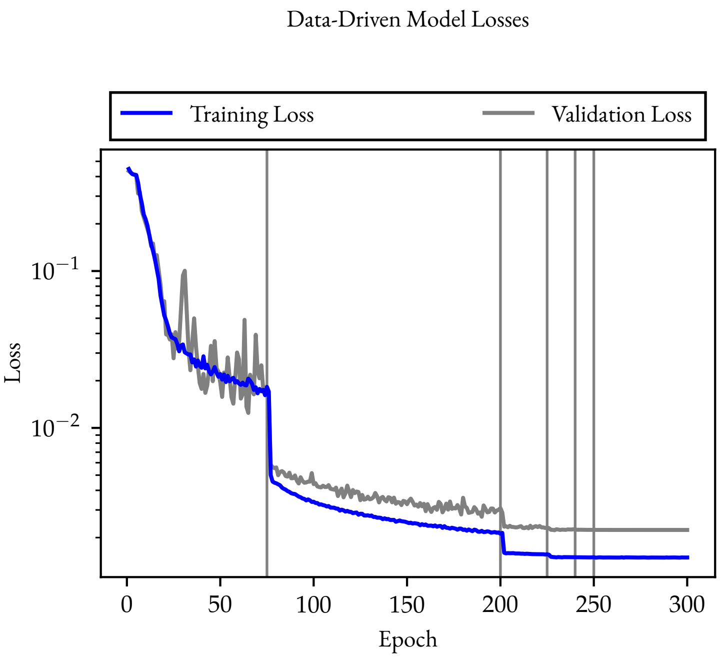

FIG. 12. Training loss and validation loss of the data-driven model. Training loss is given in blue and validation loss is given in grey. At epochs 75, 200, 225, 240 and 250 the learning rate is dropped by a factor of 10. The final training loss is 1.49×10^{-3} and the final validation loss is 2.23×10^{-3}.

1×10^{-3} for the differential loss. We train the model with an initial learning rate of 1×10^{-3} and the learning rate is reduced when the training loss plateaus using `torch.optim.lr_scheduler.ReduceLROnPlateau` with default parameters. At epochs 189, 932, 968, 978 and 992 the learning rate is dropped by a factor of 10. The model trains in roughly 1000 epochs and training takes roughly 3.8 hours. The training loss is presented in Figure 13. The final training loss is 4.71×10^{-4} and the final validation loss is 1.10×10^{-2}. At the end of training the differential loss is 7.79×10^{-2}, the initial condition loss is 1.99×10^{-5} and the boundary condition loss is 6.92×10^{-5}.

D. Alternative Procedure for Training Physics-Driven Model

We present an alternate procedure of training the physics-driven model which we refer to as the *modified physics-driven model*. We have not had time to explore the full potential of this training method but we show that it has the potential to lead to train models with better performance. We train our model piecewise by first training over a small time interval $t \in [0, t_1]$ and we reduce the learning rate when the training loss plateaus. Then we fix the learning rate and slowly extend the cutoff time from t_1 to $t_{\max}$. Then finally we train the model on the fixed time interval $t \in [0, t_{\max}]$ and we reduce the learning rate when the training loss plateaus. We find that this training method leads to better performance than directly training the model over the interval $t \in [0, t_{\max}]$.

The training data used the train the modified physics-driven model is generated from initial state up to five Fourier modes and the potentials have RMS strength of 2.0 sampled from polynomials of degree 10. The training data is length 1×10^6 and consists of 1×10^6 initial systems. We apply random phase transformations and mix the pairing up of initial states and potentials to produce new initial systems at each epoch. Throughout we use unit weightings for the initial condition and boundary conditions losses and a weighting of 1×10^{-3} for the differential loss. We train over the time interval $t \in [0, 0.25]$ for 600 epochs. We train the model with an initial learning rate of 1×10^{-3} and the learning rate is reduced when the training loss plateaus using `torch.optim.lr_scheduler.ReduceLROnPlateau` with default parameters. At epochs 310 and 472 the learning rate is dropped by a factor of 10. We then slowly extend the time cutoff at a rate of 2.5×10^{-4} over another 3000 epochs such that at the 3600th epoch the maximum cutoff time is 1. We fix the learning rate at 1×10^{-4}. The losses rise in this region due to gradually being exposed to a larger input vector set. We then train for a further 400 epochs over the fixed time interval $t \in [0, 1]$. We initially start with a learning rate of 1×10^{-4} and the learning rate is reduced when the training loss plateaued as above. At epochs 3645, 3680 and 3700 the learning rate is reduced by a factor of 10. The modified physics-model took approximately 15 hours to train. We refer to this

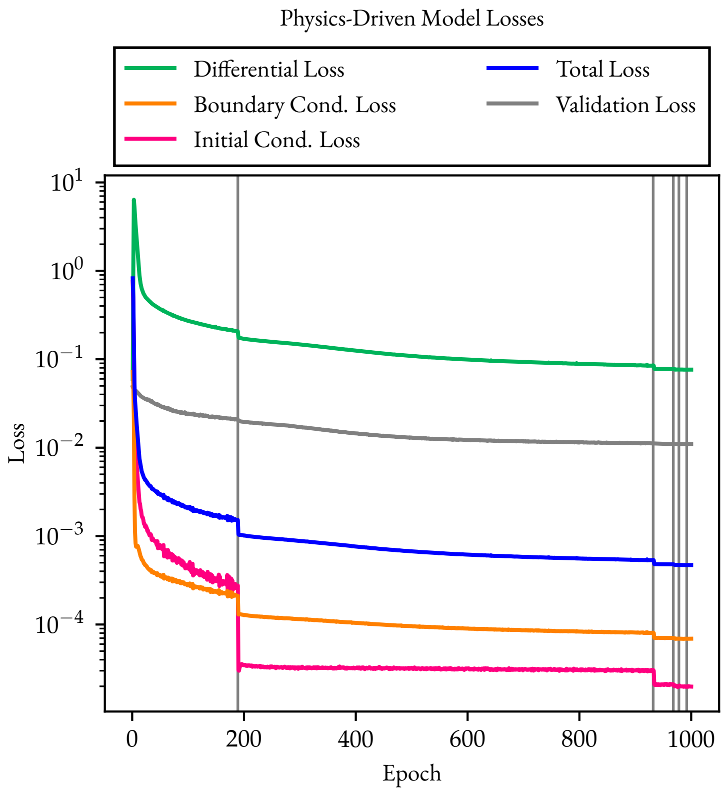

FIG. 13. Training loss, validation loss and training loss components of the physics-driven model. Training loss is given in blue, validation loss is given in grey. The differential loss is presented in green, the boundary conditions loss is in orange and the initial condition loss is pink. At epochs 189, 932, 968, 978 and 992 the learning rate is dropped by a factor of 10. The final training loss is 4.71×10^{-4} and the final validation loss is 1.10×10^{-2}. At the end of training the differential loss is 7.79×10^{-2}, the initial condition loss is 1.99×10^{-5} and the boundary condition loss is 6.92×10^{-5}.

as the modified physics-driven model. We also train a model we refer to as the control model which is trained over the fixed time interval $t \in [0,1]$ with the same training data with an initial learning rate of 1×10^{-3} We present the training losses of both the modified model and the control model in Figure 14. We find that overall all loss components of the modified method are less than the corresponding loss components of the control.

The normalisation of the fifth energy state for the zero potential system is plotted against time as predicted by both the modified physics-driven model and the control model in Figure 15. We find that the modified physics-driven model is able to produce more normalised states for 60 % longer compared to the standard physics-driven model. We therefore conclude that the modified physics-driven model has the potential to improve the performance of the physics-driven model. Hyperparameter tweaking is required to explore the full potential of the modified physics-driven training method.

E. Dynamics of the Sloped Potential

In the main text we claimed that the position expectation value obeyed Equation 19. This can be seen as the analog of Newton's second law in the classical case where the observables are given by expectation values. We give a brief account of the main dynamics of the sloped potential which we use to characterise the physics-driven model in Section III of the main text.

One may claim that the potential gradient in Equation 19 is a constant in time as it equals the slope of the potential. However this would lead to a solution of the form,

$$\langle x(t)\rangle = \langle x(0)\rangle - \frac{1}{2}\frac{\partial V}{\partial x}t^2, \tag{E1}$$

which for sufficiently large t results in $\langle x\rangle > 1$. This is unphysical as it claims that the particle would be outside the box when we enforce

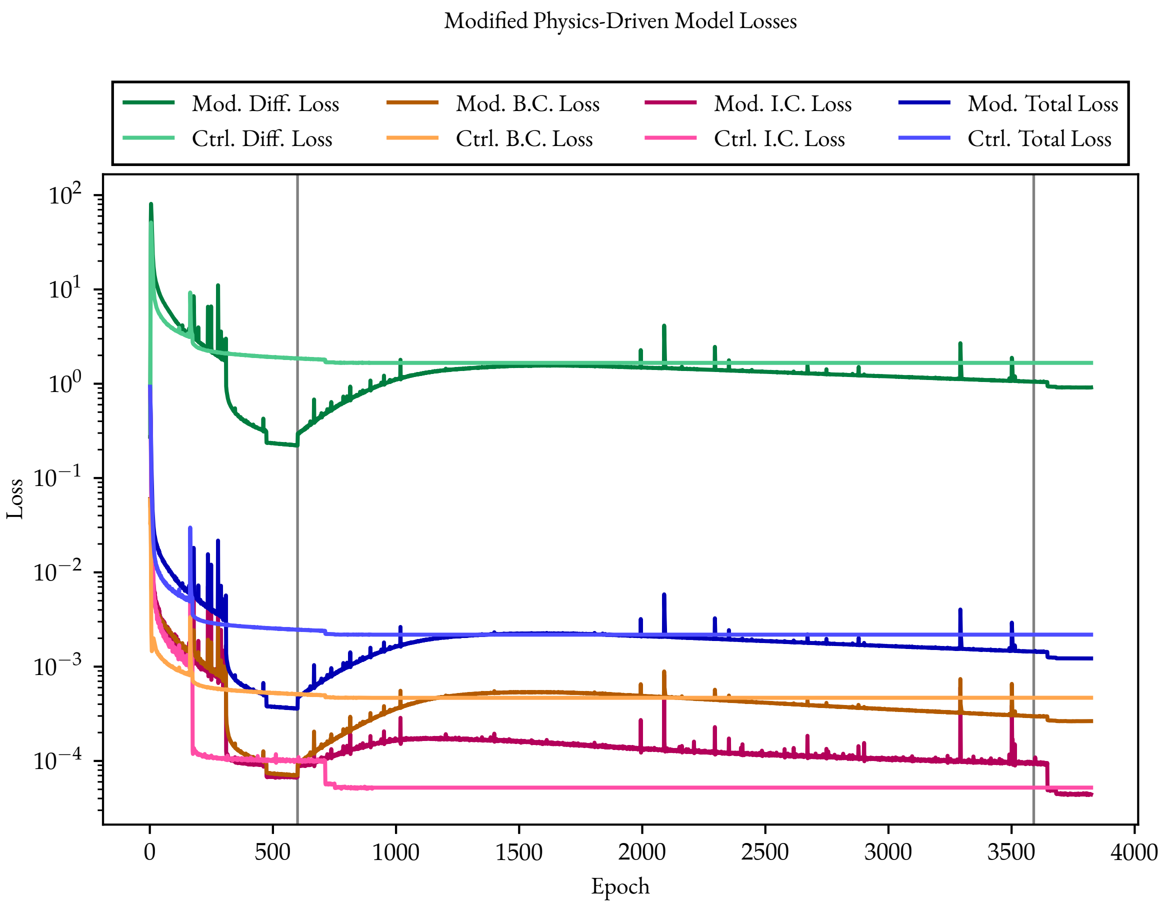

FIG. 14. Training loss and training loss components of the modified physics-driven model and the control model. The differential loss, boundary condition loss, initial condition loss and total loss are in green, orange, pink and blue respectively. The modified model losses are represented with darker colours and the control model losses are represented with a lighter colour. The final differential, boundary condition, initial condition and total loss for the modified physics-driven model are 9.13×10^{-1}, 2.65×10^{-4}, 4.42×10^{-5} and 1.22×10^{-3} respectively and for the control model are 1.66×10^{0}, 4.67×10^{-4}, 5.21×10^{-5} and 2.18×10^{-3} respectively.

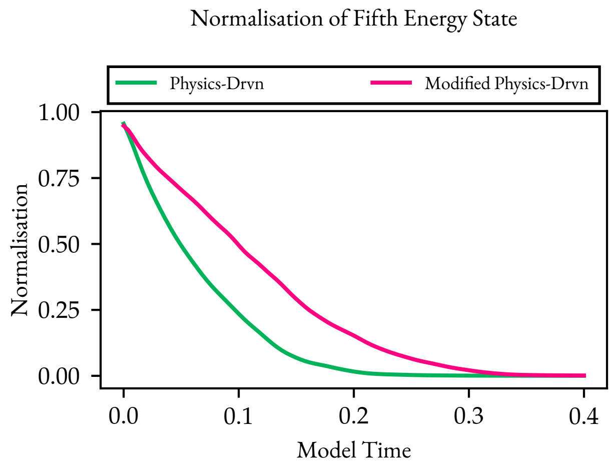

FIG. 15. Normalisation of the fifth energy state against model time as predicted by the modified physics-driven model in pink and the control physics-driven model in green.

that the particle is bound to the box. We therefore conclude the edge effects at the boundary of the box where $\frac{\partial V}{\partial x}$ is not well defined are important considerations.

We split Equation 19 up into three intervals, $x \in [0, \epsilon)$, $x \in [\epsilon, 1-\epsilon)$ and $x \in [1-\epsilon, 1]$. Then in the first and third intervals we integrate by parts. This gives us,

$$\begin{aligned}
\left\langle \frac{\partial V}{\partial x} \right\rangle &= \int_0^1 \psi^*(x,t)\ \frac{\partial V}{\partial x}\ \psi(x,t)\ \mathrm{d}x \\
&= \int_0^\epsilon \psi^*(x,t)\frac{\partial V}{\partial x}\psi(x,t)\ \mathrm{d}x + \int_\epsilon^{1-\epsilon} \psi^*(x,t)\frac{\partial V}{\partial x}\psi(x,t)\ \mathrm{d}x + \int_{1-\epsilon}^1 \psi^*(x,t)\frac{\partial V}{\partial x}\psi(x,t)\ \mathrm{d}x \\
&= [\psi^*\psi V]_0^\epsilon + [\psi^*\psi V]_{1-\epsilon}^1 - \int_0^\epsilon V(x)\frac{\partial}{\partial x}(\psi^*\psi)\ \mathrm{d}x - \int_{1-\epsilon}^1 V(x)\frac{\partial}{\partial x}(\psi^*\psi)\ \mathrm{d}x + \int_\epsilon^{1-\epsilon} \psi^*(x,t)\frac{\partial V}{\partial x}\psi(x,t)\ \mathrm{d}x \\
&= [\psi^*\psi V]_0^1 - [\psi^*\psi V]_\epsilon^{1-\epsilon} - V(0)\ [\psi^*\psi]_{x=\epsilon} + V(1)\ [\psi^*\psi]_{x=1-\epsilon} + \int_\epsilon^{1-\epsilon} \psi^*(x,t)\frac{\partial V}{\partial x}\psi(x,t)\ \mathrm{d}x. \qquad \text{(E2)}
\end{aligned}$$

Consider the simplest case of $V(x) = 0$ everywhere. Then all terms vanish to zero except for the first term. Since the walls affect all potentials including the zero potential system, we conclude that the edge effects due to the wall boundary gives rise to a force,

$$\begin{aligned}
F_{\text{wall}} &= -\ [\psi^*\psi V]_0^1 \\
&= \psi^*(0,t)\psi(0,t)V(0) - \psi^*(1,t)\psi(1,t)V(1). \qquad \text{(E3)}
\end{aligned}$$

Note that this expression is not well defined; the probability density is zero at the boundary, but the potential takes some infinite value. A better approach would include an analysis of a finite potential and then taking the limit as the potential wall tends to infinity.

We will attempt to make sense of Equation E3 qualitatively. We treat the potential at the walls $V(0) = V(1) = V$ as being a very large positive number, and the probability density at the walls, $\psi^*(0,t)\psi(0,t)$ and $\psi^*(1,t)\psi(1,t)$ is a very small positive quantity. If the probability density at the edge of the boxes are equal on both sides, then the wall exerts no net force on the particle. Furthermore, if the wavefunction is more localised towards the lefthand side of the box and the gradient is steeper at the lefthand side of the box, then $\psi^*(0,t)\psi(0,t) > \psi^*(1,t)\psi(1,t)$. We see the qualitative behaviour of the wall of box is to push a particle localised near $x = 0$ away from the wall at $x = 0$, and push a particle localised near $x = 1$ away from the wall at $x = 1$.

We therefore deduce that the behaviour of the wavefunction is to initially follow Equation E1 since initially the wavefunction is symmetric about the centre of the box so the edge effects cancel each other out. However after some amount of time, the wavefunction will be more steep towards the lower (lefthand) side of the box. The effect of this is to push the particle back towards the centre of the box. So overall, the particle's position expectation value will move towards the wall of the box, and then bounce back towards the centre of the box without the position expectation value ever touching the wall of the box. These are the dynamics which are displayed in the the physics-driven model and numerical Runge–Kutta models in Figure 8 in the main text.

F. Comparison of Physics-Driven Model to Analytical Results

At the end of Section II we characterise the physics-driven model and compare it to the numerical Runge–Kutta method for the case of the zero-potential system. However it is insightful to compare the performance of the physics-driven model in the case of the sloped potential since such potentials do not have analytical solutions and the only way to find the dynamics of such systems are through numerical approximations.

In this appendix we repeat the characterisations of Section II but we compare the two models to the a numerical solution to the sloped potential sampled on a larger grid of size 300. While this comparison isn't as true as the one presented in Section III since we have no knowledge of the true result, the numerical simulation on a large grid size is able to provide an approximation good enough for benchmarking the physics-driven model.

The performance of the model is presented in Figure 16. We note that the error initially rises above that of the numerical model, but for larger times they are able to provide similar performance. We notice that the performance does not take the usual shape as the zero potential of Figure 9. Instead the performance increases with each initial iteration.

The evaluation time of each model is presented in Figure 17. The results here are very similar to those presented in Figure 10; the numerical model evaluation time increases linearly in time and the physics-driven model evaluation time increases with each iteration. The best relative performance of 1260 is achieved at a model time of 0.80.

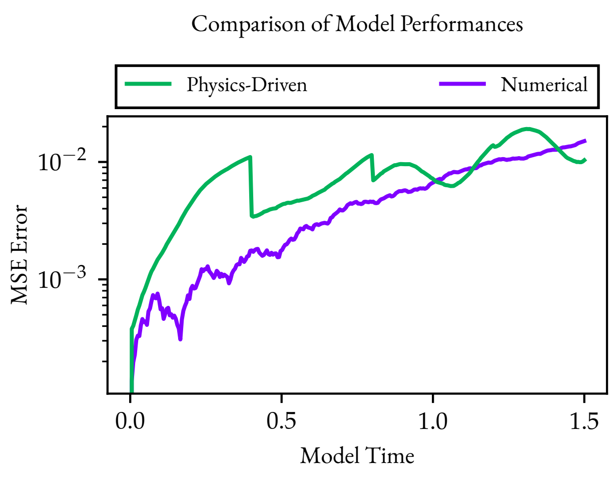

FIG. 16. MSE error between each model's output against the high resolution numerical solution of the sloped potential against model time for the physics-driven model and the numerical Runge–Kutta model. The errors of the physics-driven model are given in green while the errors of the numerical model are given in purple.

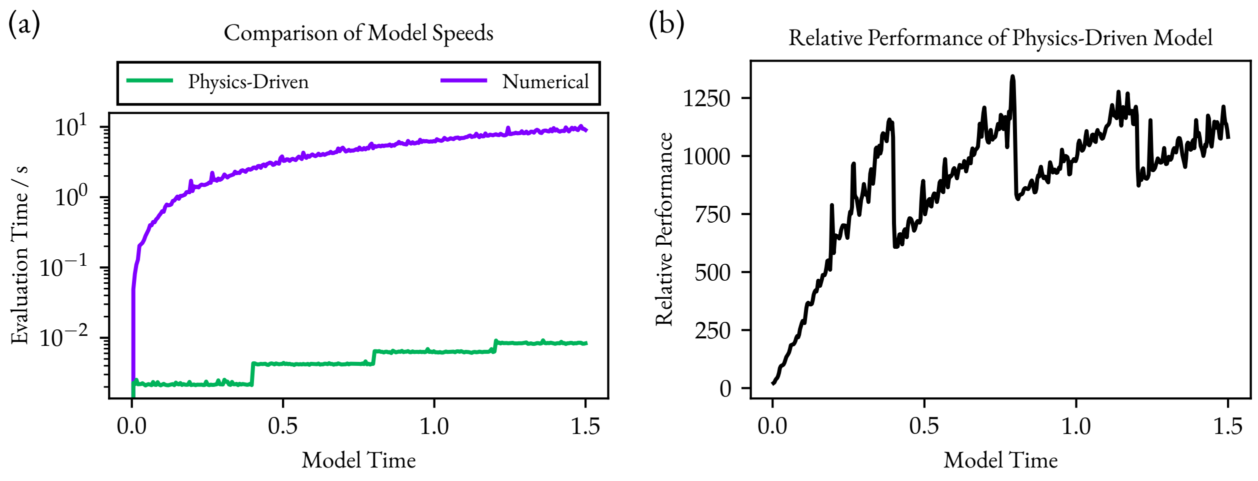

FIG. 17. Evaluation time of the physics-driven model and the numerical Runge–Kutta model against model time for the sloped potential. (a) Absolute evaluation time of the physics-driven model are given in green and the absolute evaluation time of the numerical model is given in purple. (b) The relative evaluation time between the two models is given. At a model time of 0.80, the physics-driven model is 1260 times faster than the numerical model.