I. INTRODUCTION

The Landau-Zener (LZ) transition takes place if energy levels of a two-level system undergo an avoided crossing under the action of an external drive. In 1932, Landau, Zener, Stueckelberg, and Majorana independently studied this physical phenomenon and calculated the final LZ transition probability [1–4]. The LZ model describes a fundamental physical problem and has found applications in various fields, including atomic and molecular physics [5], quantum optics [6], solid state physics [7], chemical physics [8], and quantum information science [9]. In the past decades, renewed interest in the LZ transitions has emerged in new circumstances [6–8, 10, 11], such as a nitrogen-vacancy center in a diamond lattice [12], a one-electron semiconductor double quantum dot [13], and an accelerated Bose-Einstein condensate [14].

Recently, special attention has been paid to quantum electrodynamics (QED) devices for exploration of the LZ transitions in various fields, including atomic and molecular physics, solid state physics, and quantum information science. In the past decades, renewed interest in the LZ transitions has emerged in new circumstances, such as a nitrogen-vacancy center in a diamond lattice, a one-electron semiconductor double quantum dot, and an accelerated Bose-Einstein condensate.

Approaches have been developed that model the effect of the environment by introducing a stochastic term to a two-level Hamiltonian [21]. In more recent work, the environment of a qubit is typically modelled as a bath of oscillators coupled to the qubit. For instance, Wollack et al. has treated the bath as a quantum harmonic oscillator coupled to the charge qubit [19]. System-bath interactions are complex and a variety of techniques have been developed to devise solutions. For the dissipative LZ model, final transition probabilities influenced by the bosonic bath have been investigated by Ao et al. and Häuggi et al. at zero temperature [20, 22, 23]. Thermal effects in the dissipative LZ model have been extensively studied by Nalbach et al. using the quasi-adiabatic propagator path integral (QUAPI) method and the non-equilibrium Bloch equations [24–28]. A random-variable approach has been introduced by Orth et al. [29, 30]. The hierarchical equations of motion (HEOM) method, developed and explored in [31–33], has been used by Sun et al. [34] in the framework of LZ dynamics. More generally, the role of conical intersections and avoided crossings in potential energy surfaces has been investigated in [35] for dissipative environments. HEOM and its extensions, that deal with complex multi-state systems in dissipate environments, have been developed in [36–38].

Recently, the multiple Davydov trial states has been adopted by Huang and Zhao to accurately solve the dissipative LZ dynamics [39]. However, each method has its limitations. For example, exact methods such as QUAPI and HEOM are computationally expensive, and perturbative methods such as the master equation method are efficient but inaccurate in a certain parameter space [23].

Most recently, significant progress in the artificial intelligence (AI) technology has been made in solving quantum problems because a suitable neural network can reduce the computational cost dramatically [40, 41]. Deep neural networks have been used to represent ground states of many body systems [42]. A suitable neural network together with multiscale entanglement renor-

*Electronic address: YZhao@ntu.edu.sg
malization ansatz has been used recently to tackle the problem of sampling [43]. Artificial intelligence has also been applied to treat quantum systems with both electronic and bosonic degrees of freedom. The time evolving density matrix has been combined with the transfer tensors formalism to deal with the analysis, compression and propagation of non-Markovian processes [44–46]. A reinforcement-learning scheme has been shown to be capable of describing the unitary time evolution of prototypical interacting spins models in one and two dimensions [47]. Rapid progress of AI in quantum dynamics has inspired the application of the non-linear autoregressive (NAR) neural network in dynamics of open quantum systems [48, 49], and observables of interest in the spin-boson model at long times can be modelled by the carefully trained NAR networks [50].

However, to our knowledge, neural networks have rarely been adopted to treat quantum dynamics in LZ transitions. With respect to LZ dynamics, predictions of physical observables using the neural network method have not received adequate attention. In fact, little can be found in the literature on how neural network methods are applied to provide both the final transition probability and dynamics at the intermediate times. Here, we continue our endeavor with a neural networks treatment of the LZ dynamics of a qubit coupled to a harmonic oscillator.

In this work, the recurrent neural network (RNN) will be employed to simulate LZ dynamics of a qubit coupled to single harmonic oscillator. Note that even though this quantum system cannot be termed dissipative, our work puts forth a first step towards RNN-based simulations of truly dissipative LZ dynamics in which the qubit is coupled to a large bath of oscillators.

The training data of LZ dynamics is generated by the Dirac-Frenkel variation with the multiple Davydov $D_2$ trial state. After the generation of the data, suitable nonlinear neural networks and long short-term memory (LSTM) neural networks [61] are trained and tested to possess satisfactory generalization abilities. By bypassing complex quantum mechanics, the neural network method can not only greatly accelerate the dynamics simulation at intermediate times with high accuracy, but also learn the trends of the transitions within certain parameter regimes from training data.

The remainder of the paper is structured as follows. In Sec. II, we present the Hamiltonian, our trial wave function, the multi-$D_2$ Ansatz, and several neural network methods for 2D and 3D data predictions. In Sec. III, 2D and 3D data predictions are studied using the LSTM neural network and the NAR neural network, with detailed results and discussion given. Conclusions are drawn in Sec. IV, including evaluations on 2D and 3D data predictions and further investigations.

II. METHODOLOGY

A. Model

The Hamiltonian of a driven two-level system interacting with a bosonic bath is given by

$$\hat{H} = \hat{H}_S + \hat{H}_B + \hat{H}_{SB}$$

where the system Hamiltonian is the standard LZ Hamiltonian for an isolated two-level system

$$\hat{H}_S = \hat{H}_{LZ} = \frac{vt}{2} \sigma_z + \frac{\Delta}{2} \sigma_x$$

where $\sigma_x$ and $\sigma_z$ are the Pauli matrices. The eigenstates of $\sigma_z$ are denoted by $|\uparrow\rangle$ and $|\downarrow\rangle$, and are referred to as diabatic states. Energy difference between the diabatic states, $vt$, varies linearly with time (with level-crossing speed $v > 0$), as illustrated in Fig. 1. The tunnelling strength $\Delta$ represents the intrinsic interactions between the two diabatic states, and induces transitions. Diagonalizing the time-dependent LZ Hamiltonian at each time $t$ gives rise to a corresponding pair of adiabatic states. The energies of the adiabatic states show a non-linear time-dependence, unlike those of the diabatic states (Fig. 1).

While much is still unknown about the underlying physics of the LZ model, ample attention has been devoted recently to a superconducting qubit coupled to a single harmonic oscillator [15, 51–54]. For example, a circuit QED device was fabricated by coupling the charge qubit to the superconducting resonator through an electric field [51]. We thus model the bath with a single quantum harmonic oscillator given by the Hamiltonian $\hat{H}_B$ and the qubit-bath coupling with $\hat{H}_{SB}$ [20],

$$\hat{H}_B = \hbar \omega (\hat{b}^\dagger \hat{b})$$

$$\hat{H}_{SB} = \frac{\gamma}{2} (\cos \theta \sigma_z + \sin \theta \sigma_x) (\hat{b}^\dagger + \hat{b})$$

Figure 1: Time dependence of the energies of the diabatic and adiabatic qubit states
where $\hbar = 1$ is assumed throughout, $\omega$ indicates the frequency of the bath mode with creation (annihilation) operator $\hat{b}^\dagger (\hat{b})$. Frequency $\omega = 10/\sqrt{\nu/\hbar}$ is used for convenience. $\gamma_i$ and $\theta_j$ are the qubit-bath coupling and the interaction angle, respectively. The full system is illustrated in Fig. 2. The effect of the bosonic bath is to change the energies of the qubit via the diagonal coupling ($\sigma_z$) and to induce transitions between the levels of the qubit via the off-diagonal coupling ($\sigma_x$). As the diagonal coupling is shown to have negligible influence on the transition at the intermediate times and the final transition probability, the interaction angle here is set to $\theta = \pi/2$ throughout the paper [23, 39]. As we focus on the effects of qubit-bath coupling, the tunneling strength $\Delta = 0$ is set.

B. The multi-D$_2$ state

The multiple Davydov trial states with multiplicity $M$ are essentially $M$ copies of the corresponding single Davydov Ansatz [55, 56]. They were developed to investigate the polaron model [57–59] and the spin-boson model [60] following the Dirac-Frenkel variational principle. In the two-level system, one of the multiple Davydov trial states, the multi-D$_2$ Ansatz with multiplicity $M$, can be constructed as

$$|D^M_i\rangle = \sum_{i=1}^{M} \left\{ A_i(t) |\uparrow\rangle \exp \left[f_i(t) \hat{b}^\dagger - H.c.\right] |0\rangle + B_i(t) |\downarrow\rangle \exp \left[f_i(t) \hat{b}^\dagger - H.c.\right] |0\rangle \right\},$$

where $H.c.$ denotes the Hermitian conjugate, and $|0\rangle$ the vacuum state of the bosonic bath. $A_i$ and $B_i$ are time-dependent variational parameters for the amplitudes in states $|\uparrow\rangle$ and $|\downarrow\rangle$, respectively, and $f_i(t)$ are the bosonic displacements, where $i$ labels the $i$-th coherent superposition state. If $M = 1$, the multi-D$_2$ Ansatz is reduced to the usual Davydov D$_2$ trial state.

Equations of motion of the variational parameters $u_i = A_i, B_i$ and $f_i$ are then derived by adopting the Dirac-Frenkel time-dependent variational principle,

$$\frac{d}{dt} \left( \frac{\partial L}{\partial u_i^*} \right) - \frac{\partial L}{\partial u_i} = 0.$$  \hfill (5)

For the multi-D$_2$ Ansatz, the Lagrangian $L_2$ is formulated as

$$L_2 = \langle D^M_2(t) | i\hbar \frac{\partial}{\partial t} \hat{H} | D^M_2(t) \rangle - \frac{\hbar}{2} \left[ \langle D^M_2(t) | \frac{\partial}{\partial t} | D^M_2(t) \rangle - \langle D^M_2(t) | \frac{\partial}{\partial t} | D^M_2(t) \rangle \right] - \frac{\hbar}{2} \left[ \langle D^M_2(t) | \hat{H} | D^M_2(t) \rangle \right].$$  \hfill (6)

Details of the Lagrangian, equations of motion, and initial conditions are given in Appendix A.

The observable of interest in the current work is the time evolution of the transition probability, $P_{\uparrow\rightarrow\downarrow}$, calculated as

$$P_{\uparrow\rightarrow\downarrow}(t) = \langle D^M_2(t) | \downarrow \rangle \langle D^M_2(t) | \uparrow \rangle.$$  \hfill (7)

In most parameter regimes, the graph of $P_{\uparrow\rightarrow\downarrow}(t)$ involves a sudden jump. This jump corresponds directly to the Landau-Zener transition that occurs in the system, as a result of the avoided crossing.

In the training stage, we optimize the weight parameters of our RNN neural network by feeding it with a large number of training sequences of $P_{\uparrow\rightarrow\downarrow}(t)$. 

Figure 2: A sketch of the full Hamiltonian of the qubit-oscillator system

Figure 3: The basic structure of a LSTM unit. The forgot gate $f_t$ selects the last cell state $c_{t-1}$, which determines how many of the cell states at the previous time $c_{t-1}$ are retained for those at the current time $c_t$. The input gate $i_t$ determines how many of the inputs $x_t$ at the current time are retained for the current cell state $c_t$. The output gate $o_t$ controls how many current cell states $c_t$ are retained to the final output $h_t$ of the LSTM unit. The candidate $\tilde{c}_t$ is a vector of new candidate values created by a tanh layer, that could be added to the cell state. The current unit state $c_t$ depends on both the historical memory and the current input. The final output $h_t$ is the multiplication of the output gate $o_t$ and the tanh activated current cell state $c_t$.
C. The LSTM neural network

1. RNN

The LSTM neural network [61] is a special kind of artificial RNNs which contains neural nodes connected with a time series. RNNs can describe time series because they cyclically pass states in its own network and use the memory to process the inputs, accepting a wider range of time series structure inputs. But it is difficult for a simple RNN to capture long-term dependencies, which can eliminate the difficulty of the learning from a shallow network of the last few layers. Thus, we will use the LSTM network to predict time series of LZ ties previously mentioned for simple RNNs. Thus, the learning in a deep network is equivalent to the weights of the front layers to be almost unchanged of the first few hidden layers becomes very slow, causing the vanishing gradient problem occurs when the weight update of the first few hidden layers becomes very slow, causing the weights of the front layers to be almost unchanged. The vanishing gradient problem occurs when the weight update of the last cell state becomes very slow, causing the vanishing gradient problem [61]. The vanishing gradient problem occurs when the weight update of the last cell state becomes very slow, causing the vanishing gradient problem [61]. The vanishing gradient problem occurs when the weight update of the last cell state becomes very slow, causing the vanishing gradient problem [61].

2. LSTM

The basic structure of the LSTM unit is shown in Fig. 3. Connecting multiple of LSTM units in a sequence forms the LSTM neural network to make time series predictions. The inputs of one LSTM unit consist of the actual input, time in this case, and the hidden inputs, the cell state and the output from the previous LSTM unit. So it is clearly that the network learns from the past information, data from past time steps. The LSTM cell internally can use the data from the past and current input to generate an output that weighs both the past and current information. The state transmission of the cell is like a conveyor belt, and the vector is passed through the entire cell, but with a small amount of linear operations. This structure makes it easy to pass information through the entire cell without making any changes. This is the reason why the LSTM can capture long-term correlations and save long-term memories easily. Whether we transfer the information to the next cell depends on the sigmoid function. There are six important internal properties, the forget gate, the input gate, the candidate, the output gate, the current unit state, and the final output in each LSTM unit to protect and control the information transmission.

1) The forget gate \(f_t\) selects the last cell state \(c_{t-1}\) using a sigmoid layer. The sigmoid layer can determine the number of the cell states \(c_{t-1}\) (at the previous time) that are retained to the cell states \(c_t\) (at the current time). The inputs of the sigmoid layer are the hidden state vector \(h_{t-1} \in \mathbb{R}^h\), where \(h\) is the number of LSTM cells, and the current input vector \(x_t \in \mathbb{R}^d\), where \(d\) is the dimensionality of the input vector. The output \(f_t \in \mathbb{R}^h\) is a vector that contains values between 0 and 1, representing the proportion of each number in the last cell state vector \(c_{t-1} \in \mathbb{R}^h\) that is kept. A “0” represents “completely forget” while a “1” represents “completely keep”. The output of the forget gate can be described as

\[
f_t = \sigma(W_fx_t + U_fh_{t-1} + b_f)
\]

where \(f_t\) is the output vector of the forget gate, \(\sigma\) is the sigmoid activation function, \(W_f \in \mathbb{R}^{h \times d}\) and \(U_f \in \mathbb{R}^{h \times h}\) are the weight matrices, \(x_t\) is the input vector, \(h_{t-1}\) is the hidden state vector, and \(b_f \in \mathbb{R}^h\) is the bias vector.

2) The input gate \(i_t\) determines the number of the inputs \(x_t\) at the current time which are retained for the current cell state \(c_t \in \mathbb{R}^h\) through a sigmoid layer. The sigmoid layer, with the current input vector \(x_t\) and the hidden state vector \(h_{t-1}\) as inputs, determines which information needs to be updated by outputting a vector \(i_t \in \mathbb{R}^h\) containing values between 0 and 1, where 0 means not important, and 1 means important. The output of the input gate can be described as

\[
i_t = \sigma(W_ix_t + U_ih_{t-1} + b_i)
\]

where \(i_t\) is the output vector of the input gate, \(\sigma\) is the sigmoid activation function, \(W_i \in \mathbb{R}^{h \times d}\) and \(U_i \in \mathbb{R}^{h \times h}\) are the weight matrices, \(x_t\) is the input vector, \(h_{t-1}\) is the hidden state vector, and \(b_i \in \mathbb{R}^h\) is the bias vector.

3) The candidate \(\tilde{c}_t\) is a vector of new candidate values that could be added to update the current cell state \(c_t\). The candidate vector \(\tilde{c}_t \in \mathbb{R}^h\) is generated by a tanh layer with the current input vector \(x_t\) and the hidden state vector \(h_{t-1}\) as inputs. The candidate vector is described in the following equation:

\[
\tilde{c}_t = \tanh(W_cx_t + U_c h_{t-1} + b_c)
\]

where \(\tilde{c}_t\) is the candidate vector, \(\tanh\) is the hyperbolic tangent activation function, \(W_c \in \mathbb{R}^{h \times d}\) and \(U_c \in \mathbb{R}^{h \times h}\) are the weight matrices, \(x_t\) is the input vector, \(h_{t-1}\) is the hidden state vector, and \(b_c \in \mathbb{R}^h\) is the bias vector.

4) The output gate \(o_t\) controls how many current unit states \(c_t\) are retained for the output of the LSTM using a sigmoid layer. The sigmoid layer, with the current input vector \(x_t\) and the hidden state vector \(h_{t-1}\) as inputs, outputs a vector \(o_t \in \mathbb{R}^h\) that contains values between 0 and 1, representing the parts of the current cell state \(c_t\) that is going to output. The output of the output gate can be described as

\[
o_t = \sigma(W_ox_t + U_oh_{t-1} + b_o)
\]

where \(o_t\) is the output vector of the output gate, \(\sigma\) is the sigmoid activation function, \(W_o \in \mathbb{R}^{h \times d}\) and \(U_o \in \mathbb{R}^{h \times h}\) are the weight matrices, \(x_t\) is the input vector, \(h_{t-1}\) is the hidden state vector, and \(b_o \in \mathbb{R}^h\) is the bias vector.
5) The current unit state $c_t \in \mathcal{R}^h$ is a combination of historical memory, the previous cell states, and current input, the new candidate values. The vector can be calculated using the following equation:

$$c_t = f_t \cdot c_{t-1} + i_t \cdot \tilde{c}_t$$

where $f_t$ is the output vector of the forget gate as the weight, $c_{t-1}$ is the previous cell state vector, $i_t$ is the output of the input gate as the weight vector, and $\tilde{c}_t$ is the candidate vector.

6) The final output $h_t \in \mathcal{R}^h$ is the dot product of the output gate vector $o_t$, as the weight, and the tanh layer activated current cell state vector $c_t$. The final output can be described as

$$h_t = o_t \cdot \tanh(c_t)$$

where $o_t$ is the output vector of the output gate, $\tanh$ is the hyperbolic tangent activation function, and $c_t$ is the current cell state vector.

D. The NAR neural network

1. Classical recurrent network

The autoregressive model (AR) [62] is a model that processes the time series using linear autoregression. AR models can describe the linear relationship between the current value and the historical value. The notation AR(p) indicates an AR model of order p. AR(p) can be described by

$$X_t = c + \sum_{i=1}^{p} \phi_i X_{t-i} + \varepsilon_t$$

where $X_t$ is the output, $c$ is the constant, $\phi_i$ is the parameter of the model, $\varepsilon_t$ is the random gaussian error. With fixed $p$, we can get the output using an AR model with different values of $\phi_i$.

The move-average (MA) model [63] can be used to calculate the autoregressive error. Assuming that $q$ is the number of move-average terms. The notation MA(q) indicates a move-average model of order q. MA(q) can be described by

$$X_t = \mu + \sum_{i=1}^{q} \theta_i \varepsilon_{t-i} + \varepsilon_t$$

where $X_t$ is the output, $\mu$ is the average of the time series, $\theta_i$ is the parameter of the model, $\varepsilon_t$ is the random gaussian error. With fixed $q$, we can get the output using an MA model with different values of $\theta_i$.

The autoregressive moving average (ARMA) model [64, 65] is a combination of the AR and the MA models. It has been proven useful in seasonal statistic prediction. The notation ARMA(p, q) indicates an autoregressive moving average model of orders p and q. ARMA(p, q) can be described by

$$X_t = c + \sum_{i=1}^{p} \phi_i X_{t-i} + \sum_{j=1}^{q} \theta_j \varepsilon_{t-j}$$

where $X_t$ is the output, $c$ is the constant, $\phi_i$ is the parameter of the model, $\varepsilon_t$ is the random gaussian error, $\theta_j$ is the parameter of the model. The values of $\phi_i$ and $\theta_j$ can be changed when necessary. With fixed $p$ and $q$, we can get the output using an ARMA model.

The autoregressive integrated moving average (ARIMA) model [66, 67] is updated from the ARMA model by adding the differential term to make the time series more stable. The model does not directly consider changes in other related random variables. The notation ARIMA(p, q, d) indicates an autoregressive integrated moving average model of order p and q. ARIMA(p, q, d) can be described by

$$(1 - \sum_{i=1}^{p} \phi_i L^i)(1 - L)^d X_t = (1 + \sum_{i=1}^{q} \theta_i L^i) \varepsilon_t$$

where $X_t$ is the output, $d$ is the differential term and $d > 0$, $d \in \mathcal{Z}$, $L$ is the lag operator, a time pointer, when the sequence value is multiplied by it, the sequence value time is adjusted backward. For example,

$$L^d X_t = X_{t-d}$$

$\phi_i$ and $\theta_j$ are the parameters of the model. The values of $\phi_i$ and $\theta_j$ can be changed when necessary. With fixed
where \( y \) is a time series. The NAR neural network, we can approximate the values of the time series, and the function lays, data. The time series number \((\omega, \text{ neuron biases})\) [50].

As shown in Fig. 4, the structure of the NAR neural network is shown in Fig. 4.

According to the classical autoregressive methods above, we can find that these methods always encounter difficulties when dealing with the prediction on nonlinear and long-term dependence time series.

The problem with long-term dependence always occurs when the gradient descent problem comes along. It was shown that the gradient declines quickly when \( k \ll t \). In other words, for a long-term information transmission, the weight values from the long-term time series cannot really affect the gradient of the cost far away from them, leading to the failure of prediction. However, the NAR neural network can avoid this problem.

2. NAR neural network

Similar to the disposal method used in the RNN, we use the NAR neural network with deep learning [72] to replace the classical autoregressive models. The main structure of the NAR neural network is shown in Fig. 4.

In time series modeling, the NAR neural network can be trained to predict one time series given past values of the same time series, and the feedback input. As shown in Fig. 4, we use the NAR neural network based on the neural time series toolbox of MATLAB. It can be used to predict the time series \( y(t) \) at time \( t \) given a certain number \( d \) in this case) of past values of the time series data. The time series \( y(t) \) can be described as,

\[
y(t) = F(y(t-1), y(t-2), y(t-3), ..., y(t-d))
\]

where \( y(t) \) is the output of the NAR neural network, \( d \) is the number of past values of the time series or the delays, \( y(t-1), y(t-2), y(t-3), ..., y(t-d) \) are the past values of the time series, and the function \( F \) is a nonlinear function. The learning rule is based on the Levenberg-Marquardt backpropagation algorithm [70]. After training the NAR neural network, we can approximate the time series \( y(t) \) at time \( t \) by optimizing the network weights and neuron biases [50].

3. 3D data prediction using LSTM neural network

If the probability relies on the time series as well as an evenly spaced parameter \( \gamma \), a complete relation of the probability with respect to time could be predicted by the neural network based on repetitions of the 2D time series training and prediction.

The training dataset can be viewed as a 99-by-6001 matrix. The column number \( i \) is for the \( i \)-th step in time list. Correspondingly, the row number \( j \) is for the \( j \)-th number in the list of \( \gamma \). We are aiming to find the 2D time series data with a fixed \( \gamma \) of 1.00√ℏ using the pre-generated training data. Focusing on a certain time step \( T \), all the data with the same \( T \) with \( \gamma \) varying from 0.01√ℏ to 0.99√ℏ can be regarded as a 2D time series data, which is used to predict the probability with \( \gamma = 1.00\sqrt{\hbar} \) by the LSTM neural network. After repeating the process, a time series list can be generated.

Figure 5: Structure of the 3D prediction algorithm. The training dataset can be viewed as a 99-by-6001 matrix. The column number \( i \) is for the \( i \)-th step in time list. Correspondingly, the row number \( j \) is for the \( j \)-th number in the list of \( \gamma \). We are aiming to find the 2D time series data with a fixed \( \gamma \) of 1.00√ℏ using the pre-generated training data. Focusing on a certain time step \( T \), all the data with the same \( T \) with \( \gamma \) varying from 0.01√ℏ to 0.99√ℏ can be regarded as a 2D time series data, which is used to predict the probability with \( \gamma = 1.00\sqrt{\hbar} \) by the LSTM neural network. After repeating the process, a time series list can be generated.
III. RESULTS AND DISCUSSION

A. 2D LSTM neural network

We use the LSTM neural network based on Tensor-Flow to train and predict the data. The loss function we use is the quadratic type, essentially the same as the mean square error (MSE), which can be described in the following equation,

\[ \text{MSE} = \frac{1}{N} \sum_i (f_i - y_i)^2 \]  

(22)

where \( N \) is the number of data points, \( f_i \) is the estimated value, and \( y_i \) is the value of the data point. During training, the optimizer for the LSTM neural network we choose is Adam \cite{71}, which is an adaptive learning rate method that computes individual learning rates for different parameters, achieving good results typically faster than other optimizers.

Besides, there are two more important hyperparameters to mention, the window size and the batch size. The window size is the number of time series examples to keep together sequentially. This controls the length of truncated backpropagation: smaller values mean less sequential computation, which can lead to faster training, but create a coarser approximation to the gradient. The benefit of the using window size is that it allows the network to have sliding windows overlapping with each other which fully utilize the time relationship between elements of the time series training set. Batch size is the number of windows to place together in a batch. Larger values will lead to more stable gradients during training and will improve memory utilization through parallelization and increase the gradient descent direction accuracy, but it will also be limited by the memory size.

Moreover, one significant fact to notice is that for most qubit-bath coupling strengths \( \gamma \), the graphs of \( P_{t \rightarrow t}(t) \) involve a sudden jump. At first, we used the 2D data including this jump to train the neural network, however, the future propagating trends could not fit well, because the jump gave the neural network too much interference information to predict. In order to avoid the interference, we choose the starting point of the training data from the time after the sudden jump has finished.

Taking qubit-bath coupling strength \( \gamma \) of \( 0.96\sqrt{\hbar}c \) as an example, during the training, the LSTM neural network with different training hyperparameters generates different future propagating trends. We step forward to the prediction after the LSTM neural network finishes the training when the training loss stays around a local minimum. However, for the LSTM neural network with the first set of training hyperparameters, the predicted value failed to match with the original data. To make the prediction more accurate, we have adjusted the training hyperparameters of the LSTM neural network with explanations in the following sections. With proper adjustments, the prediction can match the original data accurately.

The adjustments to the hyperparameters are tested based on trial and error. After the correct adjustments, we have successfully generated several satisfactory points whose average errors are less than a certain threshold of \( 10^{-2} \). The hyperparameters of the satisfactory models are listed in Table I. We have tried and adjusted the values of window size and batch size based on real-world conditions to fix windows size at 100 and batch size at 4 for optimal performances. The windows size and batch size are fixed at 100 and 4, respectively, because when the windows size is larger than 100 or the batch size is larger than 4, it will usually cause not enough memory to complete this operation error because of the limited memory size of our computational resource. Also, we need to try our best to control the values of the windows size and batch size as high as possible, the value of windows size and batch size can also limit each other, so windows size 100 and batch size 4 will be an acceptable choice.

In Table I, the numbers of predicted points with acceptable errors are around 250, 500, 300, and 200. We can see that the error is more acceptable when the number of hidden layers is around 252. The reason behind the choice of the number of hidden layers is that we need to find a tradeoff between the underfitting and the overfitting. The underfitting occurs when the neural network fails to fit the data well enough and capture the underlying trend of the data, whereas the overfitting occurs when the neural network captures the noise of the data or fits the data too well, not being able to generalize. A shallow neural network with a small amount of hidden layers, for example, 190 in this case, tends to underfit the training dataset, yielding a less number of satisfactory points than that of the best neural network. On the other hand, a very deep neural network with a huge number of hidden layers, for example, 573 in this case, is more likely to overfit the training dataset by giving perfect results in training but unsatisfactory results when testing. This is because the deep neural network, which overfits, fits the training dataset too well to generalize when unforeseen data come along. So the balance we find for the number of hidden layers is 252. The number of training steps for all models with different number of hidden layers is approximately 5000.

After the training for the best model with the correctly adjusted hyperparameters, we use this best model to predict. Fig. 6 shows 3 typical satisfactory results generated according to the experience above. The blue line is the prediction and the red line is the original data. The training steps are 5000, the number of the nodes in the hidden layer is 252, the batch size is 4, the window size is 100 and the number of training data is 1627. Table II shows the details regarding three successful predictions. To be more specific, the numbers of acceptable predicted points are 375, 350, and 385 for panel (a), panel (b), and panel (c) respectively. The details regarding the loss for final step, the average loss, and the average error of the three satisfactory predictions are also shown. Moreover, it's obvious that line (a) and line (c) perform better than
Table I: Training hyperparameters and prediction performance comparison of 4 different numbers of hidden layers in LSTM are shown below. Training hyperparameters include numbers of steps, hidden layers, batch size and window size. The compared parameters include the number of satisfactory points and the average error.

<table>
<thead>
<tr>
<th>Steps</th>
<th>Hidden_layers</th>
<th>Batch_size</th>
<th>Window_size</th>
<th>No. of satisfactory points</th>
<th>Average error(less than)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5000</td>
<td>190</td>
<td>4</td>
<td>100</td>
<td>250</td>
<td>$1 \times 10^{-2}$</td>
</tr>
<tr>
<td>5000</td>
<td>252</td>
<td>4</td>
<td>100</td>
<td>500</td>
<td>$1 \times 10^{-2}$</td>
</tr>
<tr>
<td>5000</td>
<td>256</td>
<td>4</td>
<td>100</td>
<td>300</td>
<td>$1 \times 10^{-2}$</td>
</tr>
<tr>
<td>5000</td>
<td>573</td>
<td>4</td>
<td>100</td>
<td>200</td>
<td>$1 \times 10^{-2}$</td>
</tr>
</tbody>
</table>

Figure 6: Satisfactory predictions are based on the parameters from testing. The blue line is the prediction and the red line is the original data. The numbers of acceptable predicted points from line (a), (b), and (c) are about 375, 350, and 385 respectively.

Figure 7: Predictions of the data for four qubit-bath coupling strengths: (a) $0.03\sqrt{\hbar\nu}$, (b) $0.06\sqrt{\hbar\nu}$, (c) $0.09\sqrt{\hbar\nu}$, and (d) $0.12\sqrt{\hbar\nu}$. The black line represents the untrained original data; the red line represents the trained original data; the blue line represents the generated value.

Figure 8: Detailed Prediction of the data for two qubit-bath coupling strengths: (a) $0.03\sqrt{\hbar\nu}$, and (b) $0.06\sqrt{\hbar\nu}$: Common characters of the predictions of the 2 datasets are that the predicted value has a larger amplitude and nearly fixed amplitude, but also has a similar phase difference, which causes the error of the predicted curve increases as time increases.

Figure 9: Detailed Prediction of the data for two qubit-bath coupling strengths: (a) $0.09\sqrt{\hbar\nu}$, and (b) $0.12\sqrt{\hbar\nu}$: For the case where qubit-bath coupling strengths is $0.09\sqrt{\hbar\nu}$, the complex network leads to fully learned the characteristic such as slowly and smoothly decreasing of the amplitude which cannot be found in the previous network. For the the case of $\gamma = 0.12\sqrt{\hbar\nu}$, the decreasing of the amplitude is not obvious, which caused the phase difference.

line (b) both in terms of the number of satisfactory points and average error.
Table II: Details of the satisfactory results are shown below, including the loss for final step, the average loss, and the average error.

<table>
<thead>
<tr>
<th>Predictions Loss for final step</th>
<th>Average loss</th>
<th>Average error</th>
<th>No. of satisfactory points</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10.582886</td>
<td>0.11918713</td>
<td>0.01847786</td>
</tr>
<tr>
<td>2</td>
<td>53.12971</td>
<td>0.15265104</td>
<td>0.02031531</td>
</tr>
<tr>
<td>3</td>
<td>31.96298</td>
<td>0.15888248</td>
<td>0.01831146</td>
</tr>
</tbody>
</table>

Table III: Prediction performance comparison for the NAR neural network with 4 different numbers of hidden layers. The compared parameters include the numbers of satisfactory points and the average error.

<table>
<thead>
<tr>
<th>Hidden layers satisfactory points</th>
<th>Average error&lt;less than)</th>
</tr>
</thead>
<tbody>
<tr>
<td>[30, 25, 30]</td>
<td>520</td>
</tr>
<tr>
<td>[15, 20, 15]</td>
<td>625</td>
</tr>
<tr>
<td>[15, 15, 15]</td>
<td>425</td>
</tr>
<tr>
<td>[30, 30, 30]</td>
<td>550</td>
</tr>
</tbody>
</table>

Table IV: Prediction time: Neural network vs the numerical method. We use a computer with Windows10_x64 operating system, Intel(R) Core i7-4720HQ CPU@2.60GHz, 7.89GB RAM, MATLAB R2017b. The computational time varies because of the dynamic condition of the computer but its value usually varies within a certain range. We can see that NAR neural network is approximately 100 times faster than the Davydov ansatz method.

<table>
<thead>
<tr>
<th>$\gamma$</th>
<th>NAR(ms) Davydov ansatz method(ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.003</td>
<td>170 $\sim$ 190  16000 $\sim$ 18000</td>
</tr>
<tr>
<td>0.006</td>
<td>170 $\sim$ 190  16000 $\sim$ 18000</td>
</tr>
<tr>
<td>0.009</td>
<td>173 $\sim$ 180  16000 $\sim$ 17500</td>
</tr>
<tr>
<td>0.012</td>
<td>175 $\sim$ 190  15500 $\sim$ 18000</td>
</tr>
</tbody>
</table>

B. 2D NAR neural network

The other kind of neural network we choose for 2D prediction is the NAR neural network. Predictions for four qubit-bath coupling strengths $\gamma$ $0.03\sqrt{\hbar v}$, $0.06\sqrt{\hbar v}$, $0.09\sqrt{\hbar v}$, $0.12\sqrt{\hbar v}$ are shown in Fig. 7. The black line represents the untrained original data; the red line represents the trained original data; the blue line represents the generated value. As you can tell with the naked eye, the predictions are reasonable as they match the expected values pretty well. In the following section, we will go into detail about the good performances in predictions.

We use the notation $[n_1, n_2, n_3, ... ]$ to describe the number of nodes for the k hidden layers in a neural network. The number of the nodes for each layer is $n_1$, $n_2$, $n_3$, ..., $n_k$ respectively. In Fig. 8, the hidden layers of the NAR neural network which predicted the data with $\gamma$ of $0.03\sqrt{\hbar v}$ are $[30, 25, 30]$. This notation means that the neural network has three hidden layers, and the number of nodes for each hidden layer is 30, 25, 30 respectively. The hidden layers of the NAR neural network which predicted the data with $\gamma$ of $0.06\sqrt{\hbar v}$ are $[15, 20, 15]$. Furthermore, we can learn some common characteristics from the two prediction plots, such as that the predicted value has not only a larger and nearly fixed amplitude, but also a similar phase difference, causing the error of the predicted curve to increase as time increases.

In Fig. 9, the hidden layers of the NAR neural network which predicted the data with $\gamma$ of $0.09\sqrt{\hbar v}$ are $[30, 30, 30]$. Moreover, this prediction plot shows that more complex neural network tends to learn the characteristics better. For example, the complex neural network learns the slowly and smoothly decreasing characteristic of the amplitude which cannot be found in previous trained neural networks. The hidden layers of the NAR neural network which predicted the data with $\gamma$ of $0.12\sqrt{\hbar v}$ are $[15, 15, 15]$. The amplitude decreasing is not obvious, causing the phase difference. It is concluded that a network with hidden layers more complex than $[30, 30, 30]$ usually can learn the decreasing amplitude characteristic. Even though these networks are tested on time series corresponding to different values of $\gamma$, such a conclusion is justified because these time series are similar in terms of complexity.

Furthermore, detailed analyses of the training process of the NAR neural network which predicted the data with qubit-bath coupling strengths $\gamma$ of $0.06\sqrt{\hbar v}$ are shown in Fig. 10. From the error histogram in Fig. 10(a), we can learn that most of the instances are large when the error is less than the threshold of $8 \times 10^{-5}$, meaning most training and validation errors are smaller than the threshold and the training process converges. During training, validating, and testing, the average error has been reduced to $1.5 \times 10^{-6}$, $8.7 \times 10^{-6}$ and $1.5 \times 10^{-5}$ respectively, as shown in Fig. 10(b), (c), (d), and (e). The overall error is about $2 \times 10^{-6}$. This proves that the error is acceptable enough and the prediction is reliable. In Fig. 10(f) and (g), we can see that as the time progresses, the error can be still limited in $2 \times 10^{-4}$ efficiently, because although the neural network becomes more and more complex during the training, it can adjust the parameters and fit the data well with its strong optimization ability.

We use a computer with Windows10_x64 operating system, Intel(R) Core i7-4720HQ CPU@2.60GHz, 7.89GB RAM, MATLAB R2017b. From Table IV, the computational time varies because of the dynamic condition of the computer but its value usually varies within a certain range. The NAR neural network and the Davydov ansatz method are implemented in MATLAB. We can see that NAR neural network is approximately 100 times faster than the Davydov ansatz method, which proves that the
Figure 10: These three pictures show the detailed analyses of the training process of the NAR neural network which predicted the data with $\gamma$ of 0.06$\sqrt{\hbar v}$, where panel (a) shows the error histogram, panels (b)(c)(d)(e) show the regression, and panels (f)(g) show the time series response.

neural network is a great improvement for the calculation speed.

C. 3D data prediction using LSTM neural network.

The goal of 3D data prediction is to predict the data with $\gamma$ of 1.00$\sqrt{\hbar v}$ based on data with $\gamma$ from 0.01$\sqrt{\hbar v}$ to 0.99$\sqrt{\hbar v}$. The 3D data prediction plots are shown in Figure 11: Line1 & Line2: The green line is the original data and the red line is the prediction. The number of the training data for each time step is 99 and the number of time steps in the training dataset is 6001, so we have predicted 6001 points to generate the predicted line. Our predictions are acceptable after training of 10400 epochs for Line1 and 7600 epochs for Line2.

Table V: 3D prediction: Errors of the prediction for both Line1 and Line2. The prediction errors of Line1 and Line2 are almost the same, so we only list them in 1 table.

<table>
<thead>
<tr>
<th>Point</th>
<th>First 3 points</th>
<th>Last 3 points</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$2.41 \times 10^{-10}$</td>
<td>$1.60 \times 10^{-4}$</td>
</tr>
<tr>
<td>2</td>
<td>$3.21 \times 10^{-9}$</td>
<td>$1.59 \times 10^{-4}$</td>
</tr>
<tr>
<td>3</td>
<td>$1.09 \times 10^{-8}$</td>
<td>$1.58 \times 10^{-4}$</td>
</tr>
</tbody>
</table>

Fig. 11. The green line is the original data and the red line is the prediction. The number of the training data for each time step is 99 and the number of time steps in the training dataset is 6001. Therefore, we predict 6001 points with a shallow neural network by repeating 2D prediction for 6001 times to generate the predicted line.

In Fig. 11, we can see that the prediction error is acceptable with minimum prediction line mismatches that are detectable with the naked eye. To illustrate this point, the error rate for the first few data points has the same order of magnitude as $10^{-10}$ and the error rate for the last few data points has the same order of magnitude as $10^{-4}$, which are both extremely small. This shows the good performances in our predictions. We have trained 10400 epochs for Line1 and 7600 epochs for Line2, where one epoch is one training step going through the entire training dataset. Table V shows the errors of the first three points of Line1 and Line2 are both approximately $2.41 \times 10^{-10}$, $3.21 \times 10^{-9}$ and $1.09 \times 10^{-8}$, and the errors of the last three points are approximately $1.60 \times 10^{-4}$, $1.59 \times 10^{-4}$ and $1.58 \times 10^{-4}$.

The high prediction accuracy achieved by the 3D prediction approach is owed to the fact that the data at each time step $T$ is treated separately, by separately trained neural networks. Hence, the neural networks do not need to learn the complex dependence of $P_{\uparrow \rightarrow \downarrow}$ on time, such
as the sharp jumps; the networks only need to learn the relatively simple and smooth dependence of $P_{\gamma \rightarrow \lambda}$ on $\gamma$.

Admittedly, it must be noted that neural networks cannot achieve perfect accuracy. Furthermore, obtaining statistical confidence intervals for the networks’ predictions is not straightforward. The lack of such performance guarantees may make these methods unattractive to applications with very low error tolerances. In fact, one way to improve the accuracy is to use classical methods because they are more suitable for the prediction with a small amount of the data. We have also included a discussion of one classical method in Appendix C. However, there is the other way for neural networks to improve 3D prediction accuracy that is to increase the amount of training data to create a more complex neural network. Moreover, it is worth noting that the initial values in the neural network are random, so the prediction lines are different even with the same training hyperparameters. The training results are sensitive to initial values of the neural network, so another possible improvement is to fix the initial values of the neural networks at each training. This improvement can be achieved by either setting the random seed variable, determined by an effective estimate of the favorable initial values, or by using the weights of another pre-trained neural network as the initial values of the next training process to reduce the training time and increase the prediction accuracy.

IV. CONCLUSION

To our knowledge, the artificial neural networks have rarely been applied to the simulation of the dynamics of quantum systems such as the LZ model. This motivated us to use the state-of-the-art neural network models to carry out such simulations. In this work, the LZ dynamics of a qubit-oscillator system is simulated by NAR and LSTM neural networks, which are trained and successfully validated in 2D prediction, implying that sufficient information is provided to the neural networks to predict the time series in this work. Considerable accuracy is also achieved in 3D prediction by repeating the 2D predictions with one parameter varying. These results suggest that our neural network-based approaches provide a quick and reliable way to predict and simulate the dynamics of the LZ system.

Admittedly, statistical and machine learning models such as neural networks cannot achieve perfect accuracy, and are bound to make errors. But, overall, the accuracy is relatively high and the errors are minimal. Furthermore, the methods developed in this paper prove to be useful because of their low computational complexities. Before using standard methods such as Davydov trial states and QUAPI to carry out heavy simulations that could run for days, a researcher could use a trained neural network to obtain a reliable preview of the results to come.

Finally, our approach can potentially be extended to simulate truly dissipative LZ dynamics. This is owed to the fact that the Davydov trial states are well-equipped to accurately simulate LZ dynamics with a large bath of oscillators, as described in Ref. [39]. This is achieved by using a trial state with a large number of bath modes $N$, and a high value of multiplicity $M$, in Eq. (5) of Ref. [39]. Once this method is used to generate training data consisting of time series of the transition probability, our neural network analysis can be applied to dissipative systems without any need for modification.

Acknowledgments

We thank Zhongkai Huang, Yubing Liao and Frank Grossmann for helpful discussion and graphics assistance. Competitive Research Programme (CRP) under Project No. NRFCRP5-2009-04 and from the Singapore Ministry of Education Academic Research Fund Tier 1 (Grant Nos. RG106/15, RG102/17, and RG190/18) is gratefully acknowledged.

Appendix A: The time dependent variational approach for the Landau-Zener model

In order to apply the Dirac-Frenkel time-dependent variational principle, we first need to calculate the Lagrangian $L_2$,

$$L_2 = \frac{i}{2} \sum_{i,j} \left( A_j^* \dot{A}_i - \dot{A}_j^* A_i + B_j^* \dot{B}_i - \dot{B}_j^* B_i \right) S_{ji},$$

$$+ \frac{i}{2} \sum_{i,j} \left( A_j^* A_i + B_j^* B_i \right) \left[ \frac{\dot{f}_j^* f_j + f_j^* \dot{f}_j}{2} \right]$$

$$- \frac{\dot{f}_j f_j^* + f_j \dot{f}_j^*}{2} + f_j^* f_j - f_j f_j^* \right] S_{ji},$$

$$- \left\langle D^M_{2j}(t) \mid \dot{H} \mid D^M_{2j}(t) \right\rangle,$$

where the Debye-Waller factor is $S_{ji} = \exp \left\{ - \left( |f_j|^2 + |f_i|^2 \right) / 2 + f_j^* f_j \right\}$, and the last term in Eq. (A1) can be obtained as

$$\left\langle D^M_{2j}(t) \mid \dot{H} \mid D^M_{2j}(t) \right\rangle = \frac{\omega f_j^* f_i + \Delta}{2} \sum_{i,j} \left( A_j^* B_i - B_j^* A_i \right) S_{ji}$$

$$+ \sum_{i,j} \left( A_j^* A_i + B_j^* B_i \right) \omega f_j^* f_i S_{ji}$$

$$+ \frac{1}{2} \sum_{i,j} \left( A_j^* A_i - B_j^* B_i \right) \gamma \cos \theta \left( f_i + f_j^* \right) S_{ji}$$

$$+ \frac{1}{2} \sum_{i,j} \left( A_j^* B_i + B_j^* A_i \right) \gamma \cos \theta \left( f_i + f_j^* \right) S_{ji}.$$

(A2)
The Dirac-Frenkel variational principle results in equations of motion for $A_i$ and $B_i$,

$$
-i \sum_i \dot{A}_i S_{ki} - i \sum_i A_i \left[ - \hat{f}_i f_i^* + f_i \hat{f}_i^* \right] S_{ki} = -i \frac{vt}{2} \sum_i S_{ki} \Delta - \sum_i A_i \omega f_i^* f_i S_{ki} - \frac{1}{2} \sum_i A_i \gamma \cos \theta (f_i + f_i^*) S_{ki} - \frac{1}{2} \sum_i B_i \gamma \sin \theta (f_i + f_i^*) S_{ki},
$$

(A3)

and

$$
-i \sum_i \dot{B}_i S_{ki} - i \sum_i B_i \left[ - \hat{f}_i f_i^* + f_i \hat{f}_i^* \right] S_{ki} = +i \frac{vt}{2} \sum_i S_{ki} \Delta - \sum_i A_i \omega f_i^* f_i S_{ki} + \frac{1}{2} \sum_i B_i \gamma \cos \theta (f_i + f_i^*) S_{ki} - \frac{1}{2} \sum_i A_i \gamma \sin \theta (f_i + f_i^*) S_{ki}.
$$

(A4)

The equations of motion for $f_i$ are

$$
-i \sum_i \left[ \left( A_i^* \dot{A}_i + B_i^* \dot{B}_i \right) f_i - (A_i^* A_i + B_i^* B_i) \frac{f_i}{\hbar} \right] S_{ki} = -i \frac{vt}{2} \sum_i \left( A_i^* A_i + B_i^* B_i \right) f_i S_{ki} \times \left( 2\hat{f}_i \hat{f}_i - \hat{f}_i \hat{f}_i^* - \hat{f}_i^* \hat{f}_i \right) - \frac{1}{2} \sum_i \left( A_i^* A_i - B_i^* B_i \right) f_i S_{ki} - \frac{1}{2} \sum_i \left( A_i^* A_i - B_i^* B_i \right) \gamma \cos \theta S_{ki} - \frac{1}{2} \sum_i \left( A_i^* A_i - B_i^* B_i \right) \gamma \sin \theta S_{ki} - \frac{1}{2} \sum_i \left( A_i^* B_i + B_i^* A_i \right) f_i S_{ki} - \frac{1}{2} \sum_i \left( A_i^* B_i + B_i^* A_i \right) f_i \gamma \cos \theta (f_i + f_i^*) S_{ki} - \frac{1}{2} \sum_i \left( A_i^* B_i + B_i^* A_i \right) f_i \gamma \sin \theta (f_i + f_i^*) S_{ki} - \frac{1}{2} \sum_i \left( A_i^* A_i + B_i^* B_i \right) f_i S_{ki}.
$$

(A5)

It should be noted that the main results of this work are calculated from the above equations of motion. The equations of motion are solved numerically by means of the fourth-order Runge-Kutta method. In this work, the qubit is assumed to initially occupy the state $\left| \uparrow \right>$, i.e., $A_1(0) = 1$, $B_1(0) = 0$, and $A_i(0) = B_i(0) = 0 (i \neq 1)$. The initial bosonic displacement is set to zero ($f_i (t \to -\infty) = 0$), though the LZ transitions have been demonstrated to depend also on various types of initial coherent superposition states [52, 73].

Appendix B: A series of 3D data generated by adjusting $\Delta$ and $\gamma$.

In the 3D data prediction process, original data are generated by the variational method. In addition to the original data in the 3D data prediction, we vary the tunneling strength and coupling strength for further study. As shown in Fig. 12, single stage transitions occur in the absence of the tunneling strength and the presence of the off-diagonal qubit-bath coupling. As the coupling strength increases from zero to 0.19$\sqrt{\hbar v}$, the final transition probability gradually grows to about 1, as presented in the first row of Fig. 12. That is to say: a larger off-diagonal coupling strength leads to more transitions at intermediate times. As shown in the lower panels, as the coupling strength grows larger, the final transition probabilities cannot be increased because the population of the downstate has been very close to 1 and cannot be substantially changed. Although final transition probabilities can hardly be manipulated in the strong off-diagonal coupling regime, the increasing coupling strength can still make a difference, rendering the transition time longer. As shown in Ref. [39], a large tunneling strength leads to both a high final transition probability and a longer transition time. This phenomenon can be explained that the sufficiently large off-diagonal coupling strength results in a wide gap between the two adiabatic states and acts like a large tunneling strength. In addition, high-frequency oscillations become more obvious as the transition time gets longer, implying rich physics during the transition process. This varied data can be used for future prediction.

Appendix C: Classical methods for data-driven prediction

The 3D prediction approach described in section II E can be implemented using classical data-driven methods such as interpolation or curve fitting. Though these methods might not be able to achieve that accuracy or generalizability provided by LSTM neural networks, they
fitting process is very quick, and does not need large amounts of data to produce decent predictions.

Here, we demonstrate prediction by interpolation. First, we note that the transition probability is a function of time $t$ and coupling strength $\gamma$:

$$P_{\uparrow \rightarrow \downarrow} = P_{\uparrow \rightarrow \downarrow}(t; \gamma)$$  \hspace{1cm} (C1)

When time is fixed at some $t = T$, we can view $P_{\uparrow \rightarrow \downarrow}$ as a function of only $\gamma$. Also, for most values of $t = T$, the graph of $P_{\uparrow \rightarrow \downarrow}$ as a function of only $\gamma$ is smooth.

![Figure 12: $P_{\uparrow \rightarrow \downarrow}(t)$ of the Landau-Zener model without tunneling strength $\Delta$. The coupling strength $\gamma$ varies from $0.01\sqrt{\hbar v}$ to $0.99\sqrt{\hbar v}$. Frequency $\omega = 10\sqrt{v/\hbar}$ is used.](image)

Figure 12: $P_{\uparrow \rightarrow \downarrow}(t)$ of the Landau-Zener model without tunneling strength $\Delta$. The coupling strength $\gamma$ varies from $0.01\sqrt{\hbar v}$ to $0.99\sqrt{\hbar v}$. Frequency $\omega = 10\sqrt{v/\hbar}$ is used.

This provides some distinct advantages: the interpolation or fitting process is very quick, and does not need large amounts of data to produce decent predictions.

Hence, from a few pairs of training data $(\gamma, P_{\uparrow \rightarrow \downarrow})$, we can compute an interpolating function, which can then predict the value of $P_{\uparrow \rightarrow \downarrow}$ for any desired value of $\gamma$, for the fixed time step $T$. This process can be repeated for all time steps to generate a full time series of $P_{\uparrow \rightarrow \downarrow}$ for the desired value of $\gamma$.

As an example here, we predict the time series for $\gamma = 1.00\sqrt{\hbar v}$, using the data from time series for $\gamma = 0.01\sqrt{\hbar v}, 0.03\sqrt{\hbar v}, \ldots, 0.99\sqrt{\hbar v}$. This is half the amount of data used for the LSTM-based prediction. The interpolation is carried using MATLAB’s implementation of the Piecewise Cubic Hermite Interpolating Polynomial (PCHIP) method. Fig. 13 shows the prediction results; the true time series is closely approximated by the predicted one.

![Figure 13: Prediction of the time series for $\gamma = 1.00\sqrt{\hbar v}$ using the PCHIP method.](image)

Figure 13: Prediction of the time series for $\gamma = 1.00\sqrt{\hbar v}$ using the PCHIP method

On a standard laptop, the training stage of computing the interpolating functions, and the testing stage of predicting the time series, took only 1.1607 s and 0.2859 s respectively. The computational cost can be reduced further by using simpler interpolation or fitting methods.
