# Supplemental Material: Predicting quantum many-body dynamics with transferable neural networks

Ze-Wang Zhang,[1] Shuo Yang,[2] Yi-Hang Wu,[3] Chen-Xi Liu,[3] Yi-Min Han,[3] Ching-Hua Lee,[4,5] Zheng Sun,[3] Guang-Jie Li,[3] and Xiao Zhang[3,*]

[1]*School of Physics, Sun Yat-sen University, Guangzhou 510275, China. Telephone: 15602298593*
[2]*State Key Laboratory of Low-Dimensional Quantum Physics and Department of Physics, Tsinghua University, Beijing 100084, China*
[3]*School of Physics, Sun Yat-sen University, Guangzhou 510275, China*
[4]*Department of Physics, National University of Singapore, 117542, Singapore*
[5]*Institute of High Performance Computing, 138632, Singapore*

## I. SPIN AND CHAIN ENCODING MECHANISMS

As a prelude to the unified framework, we proposed two universal encoding mechanisms for any wavefunctions of 1D Ising spin chain, they are spin encoding and chain encoding. Both encodings are actually implemented as feed-forward neural networks, the weights of them are initially sampled from the normal distribution [1]. We present the details of spin encoding and chain encoding in Fig. I1.

The input of our SRU-based framework is composed of spin encoded vector and chain encoded vector. First, we randomly initialize two spin encoding embedding matrices $E^{re}_{spin}, E^{im}_{spin} \in \mathbb{R}^{2^n \times m}$ from normal distribution, here $n$ is the maximum size of spin variables, and $m$ is the hidden units for representing information of spin feature. "re" denotes the embedding matrix of real part and "im" denotes embedding matrix of the imaginary part. In this paper, we set the $n$ to be 10 (Note that $2^{10} = 1024$) and the $m$ to be 256. Specifically, taking 7-spin system as example, a state sequence of its $T$ timesteps can be represented as a complex matrix $S \in \mathbb{R}^{T \times 2^7}$. We split the complex matrix $S$ into its real part $S^{re}$ and its imaginary part $S^{im}$. Since $2^7 = 128$, we fetch the beginning 128 rows from $E^{re}_{spin}$ and $E^{im}_{spin}$, and then multiply them by $S^{re}$ and $S^{im}$, respectively. We take such process as "lookup" shown in Fig. I1. Finally, we concatenate the real part encoded vector and imaginary part encoded vector to form the spin encoded vector ($\mathbb{R}^{T \times 512}$).

Similarly, we randomly initialize a chain embedding matrix $E_{chain} \in \mathbb{R}^{n \times m}$, where $n$ (=10) denotes the maximum size of spin variables of a chain, and $m$ (=512) is the hidden unit for storing chain feature. Specifically, taking 7-spin system as example, we adopt one-hot vector $[0, 0, 0, 0, 1, 0, 0, 0, 0, 0]$ to represent such system, and multiply it by chain embedding matrix $E_{chain}$ to obtain the chain encoded vector of $\in \mathbb{R}^{T \times 512}$. Obtaining the spin encoded vector ($\mathbb{R}^{T \times 512}$) and chain encoded vector ($\mathbb{R}^{T \times 512}$), we take the sum of them as the input to SRU-based framework at each timestep. The workflow of spin and chain encoding is illustrated in Fig. I1.

## II. SRU CELL

In most RNN architectures, including Long Short-term Memory (LSTM)[2], the new state of each timestep must be suspended until completing execution of both input and previous state. In contrast, comparing with the mainstream recurrent neural networks (RNNs), each SRU cell having a cell state and a forget gate is advantagous in terms of its efficient and parallelizable element-wise matrix product. The SRU cell is very crucial for predicting dynamics of 1D Ising model, and we assume that it can be also applied to other sequential physical tasks.

A single layer of SRU employs the following computations:

$$\begin{aligned} f_t &= \sigma(W_f x_t + v_f \odot c_{t-1} + b_f) \\ c_t &= f_t \odot c_{t-1} + (1 - f_t) \odot (W x_t) \\ r_t &= \sigma(W_r x_t + v_r \odot c_{t-1} + b_r) \\ h_t &= r_t \odot c_t + (1 - r_t) \odot x_t \end{aligned} \tag{1}$$

where $f_t$ is a forget gate that can discard the redundant information several timesteps ago and unnecessary information of old spin chains in fusion training. It is computed from a sigmoid function ($\sigma$, $\sigma(x) = \frac{1}{1+e^{-x}}$) activated sum of input $x_t$ weighted by matrix $W_f$, the previous cell state $c_{t-1}$ weighted by vector $v_f$ and a bias vector $b_f$. $c_t$ is the

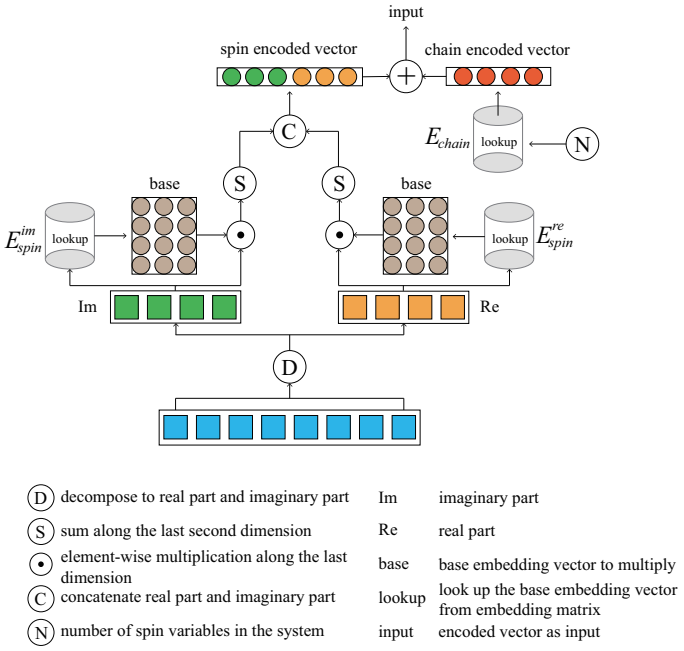


FIG. I1: Detailed calculation of spin and chain encoding mechanisms. The spin and chain encoded vector are concatenated into the final input vector for feed-forward layers.

cell state that serves to give feedbacks of latent correlated features from previous to next recursively, as the moving average of previous cell state $c_t$ weighted by $f_t$ plus the weighted input $Wx_t$. $c_t$ is crucial for the accurate prediction of dynamics based on just one initial quantum state. $r_t$ is the skip gate of highway network [3] that is a shortcut for information flow with learnable weights to determine the extent of the information skip, and is computed from a sigmoid function activated sum of weighted input $W_r x_t$, weighted previous cell state $v_r c_{t-1}$ and a bias vector $b_r$. The skip gate $r_t$ adaptively averages the input $x_t$ and the cell state $c_t$ produced from the light recurrent connection. $(1-r_t)\odot x_t$ is a residual connection to settle the gradient dimishing problem, and the useage of highway network has been proved to improve the stability. Four SRU layers are stacked to learn better context-dependent latent information and transferable features.

An obvious difference between SRU and common RNN architectures is the way of $c_{t-1}$ used in the sigmoid gate. Generally, $c_{t-1}$ is involved in matrix multiplication with weight matrix to compute each gate. For example, the forget gate of a commonly adopted LSTM architecture is computed as $f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f)$. Obviously, the matrix multiplication between $c_{t-1}$ and $W_{cf}$ makes the parallelization of computing state $f_t$ almost impossible, because each dimension of state $f_t$ can't be derived until the whole dimensions of $c_{t-1}$ are available. In Eq. (1), we substitute all matrix multiplications associated with $c_{t-1}$ with element-wise multiplications (denoted as $\odot$). Based on this simplication, we make each state of SRU independent and parallelizable, and it is efficient and lossless.

To make SRU work efficiently, we follow the guidances of parallelized implementation performed in the context of CUDA programming [5]. Eq. (1) has totally three weight matrices: $W$, $W_f$ and $W_r$, and all of them need to be multiplied by the input sequence $\{x_1 \dots x_L\}$, where $L$ is the sequence length. To fully utilize the computational intensity of GPU, we do the batched matrix multiplications across all timesteps as:

$$U^T = \begin{pmatrix} W \\ W_f \\ W_r \end{pmatrix} [x_1, x_2, \dots, x_L], \tag{2}$$

$U$ is the computed matrix. If the hidden unit of weight is $d$ in demension and the input is a mini-batch of $BS$ sequences, then $U$ would be a tensor of size $[L, BS, 3d]$.

Kernel fusion[6, 7] is an optimization method to reduce overhead of data transfer from global memory by fusing some sequential kernels into a single large one, to improve performance and memory locality. Specifically, we compile all element-wise multiplications into a single fused CUDA kernel function and parallelize the computation across each dimension of cell state. The complexity of each SRU layer is $\mathcal{O}(L \cdot B \cdot d)$ whereas the complexity of a vanilla LSTM layer is $\mathcal{O}(L \cdot B \cdot d^2)$.

## III. SPIN DECODING MECHANISM

When the hidden states of sequential modeling are obtained, we need to restore the actual predicted states. Correspondingly, we propose the spin decoding mechanism (detailed in Fig. III2). To get the final predicted output, we need the explicit size of spin variable $N$, and then look up two embedding matrices of feature dimension $2^N$. We then transpose each embedding vector and then do matrix multiplication between embedding vector and hidden state. Finally, we concatenate the real part and imaginary part to get the predicted state. The embedding matrix $E_{dec} \in \mathbb{R}^{1024\times512}$ of spin decoding layer is also randomly initialized from normal distribution.

The overall methodology for the design of our SRU-based framework follows the criteria of unity, efficiency and transferability. In addition to the spin encoding, chain encoding, SRU module and spin decoding described above, the feed-forward layers are the bottleneck layers containing full connections with a *tanh* nonlinearity and dropout[8], whose purpose is to distill the high-level latent feature and enable the framework to generalize well to unseen input. Such a gap between training and inference as context mismatch, mechanisms to alleviate such effect in sequence models are scheduled sampling[9], professor training[10] and generative adversial network (GAN)[11, 12]. We empirically observe that just adding dropout layers also yields better scalability in the context of 1D Ising spin chain. The trainable weights and biases are initialized with zero-mean and unit-variance random distributions throughout the entire framework. Each input state (the real and imaginary parts are concatenated together) is represented by the spin encoding layer and chain encoding layer, and then concatenated into two stacked feed-forward layers with *tanh* activation. The dropout layer is also applied to the inputs prior to the second feed-forward layer for regularization. To be specific, we set the dropout rate to be 0.005, that is to say, during training, about 0.5% of neurons are set to be zero randomly. The SRU module, taking these hidden features from the dropout layer as input to construct the equivalent contextual level representation, updates the cell state and output the hidden state. Finally, a feed-forward layer with the spin decoding generates a new physical state.

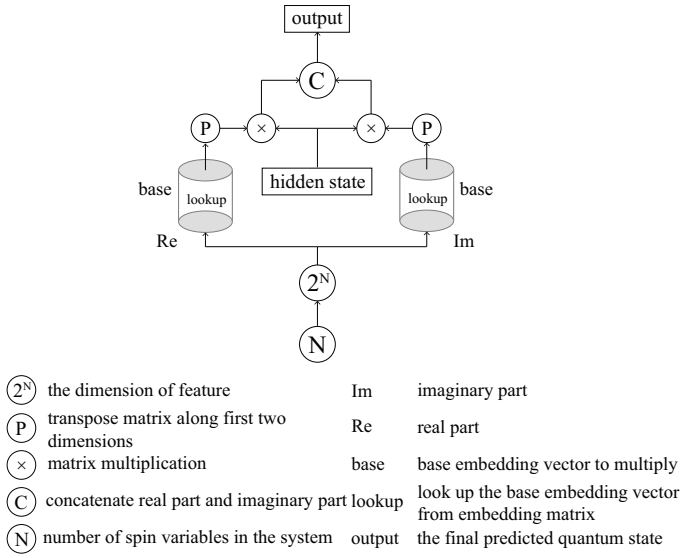


FIG. III2: Detailed calculation of spin decoding mechanism. The hidden state is multiplied by spin decoding embedding to form the final predicted quantum state.

## IV. TRAINING AND INFERENCE PROCEDURE

**Algorithm 1** Learning quantum many-body dynamics by SRU-based framework

**Require:** $\theta$: the parameters of whole network composed of spin and chain encoding, SRU and spin decoding layers.
**Input:** $B_S$ coefficient sequences in a batch with $T_L$ timesteps each.
**Output:** same as input but with one timestep offset.
**Preparation:** concatenate the real part $x_r$ and imaginary part $x_i$ of each complex coefficient into $x$.

**Training Stage:**
**Initialize** $\theta$
**for** each training epoch **do**
  **for** $k$ steps **do**
    number of particles is $N$
    obtain the output of spin encoding for both real part and imaginary part:
      $feature_{SE} = [Emb_{SE}(x_r)x_r; Emb_{SE}(x_i)x_i]$
    obtain the output of chain encoding:
      $feature_{CE} = Emb_{CE}(N)$
    get the sum as input:
      $input = feature_{SE} + feature_{CE}$
    pass through two feed-forward layers, apply dropout and layer normalization (LN):
      $output_{f1} = LN(dropout(ReLU(W_1 input))W_2 + input)$
    pass through block of four stacked SRU layers:
      $output_s = SRU_{block}(output_{f1})$
    pass through two feed-forward layers, apply dropout and layer normalization (LN):
      $output_{f2} = LN(dropout(ReLU(W_1 output_s))W_2 + output_s)$
    obtain the real part of output coefficient from spin decoding:
      $c_r = Emb^R_{SD} output_{f2}$
    obtain the imaginary part of output coefficient from spin decoding:
      $c_i = Emb^I_{SD} output_{f2}$
    Update $\theta$ by ascending the stochastic gradient descent:
    $$\nabla(\theta)\left\{\frac{1}{T_L}\sum_{T_L}\frac{1}{B_S}\sum_{B_S}\left[(x_r - c_r)^2 + (x_i - c_i)^2\right]\right\}$$
  **end for**
**end for**

**Inference Stage:**
**for** $g$ steps **do**
  **if** $g == 1$ **then**
    compute the first predicted coefficient vector $c_1$ by feeding with initial coefficient sequence as *input*:
      $c_1 = framework(input; \theta)$
  **else**
    compute new coefficients $c_g$ by feeding with previous output $c_{g-1}$:
      $c_g = framework(c_{g-1}; \theta)$
  **end if**
**end for**

At the stage of training (IV3), given a state sequence of $n$ timesteps $\{\Psi_1, ..., \Psi_n\}$, we can split it into two subsequences: $seq_i = \{\Psi_1, \Psi_2, ..., \Psi_{n-2}, \Psi_{n-1}\}$ and $seq_o = \{\Psi_2, \Psi_3 ..., \Psi_{n-1}, \Psi_n\}$. The state sequences are divided into training, validation and test sets, with an 80%-10%-10% split. Considering the different scales in different systems, we observed no empirical benefits from introducing data normalization pipeline. It therefore makes sense to use raw data generated from exact diagnolization (ED) method and avoid the need of data normalization and denormalization. We use a sequence learning method which passes the input $seq_i$ into the network at each timestep, and computes output $seq_o$ of the final layer based on a linear activation function. In contrast, at the stage of inference (Fig.1(d)), we only pass the initial wavefunction into the network as input, and the network then autoregressively generates the following states at future timesteps, by consuming the previously generated states as additional input. The full procedure of training and inference is detailed in Algorithm 1. The training loss and inference loss is shown in IV4.

To be more explicit, our dataset is gathered by computing the time evolution of wavefunctions from the two to seven-spin systems every 0.002 second, using ED method[13]. Treated as a standard ML process, we divide the whole dataset into three parts: a training set with $10,000$ sequences, a validation set with $1,000$ sequences and a test set

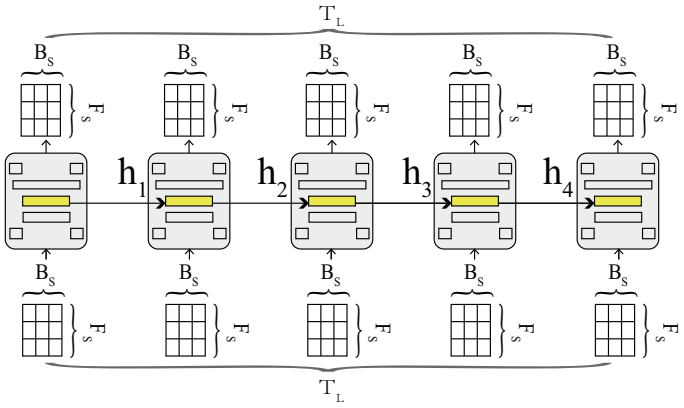


FIG. IV3: Traing the network in a teacher forcing mode[4], we take the current wave functions as input and next state as the ground label. The ground label sequence is one timestep off the input sequence.

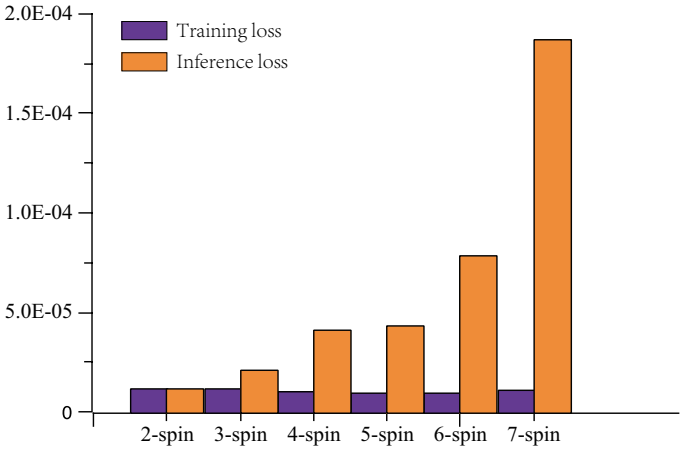


FIG. IV4: Training loss and inference loss of different 1D Ising systems. Inference loss represents the average loss of wavefunction evolution in 100 timesteps, the maximum inference loss is within 2.0E-4.

with $1,000$ sequences. During training, we use the mini-batch gradient descent framework to optimize our network and $m = 16$ is the batch size ($B_S$) we use, considering both NN's converging speed and GPU memory.

The target of training is to minimize the mean squared error (MSE) at all timesteps between output and target states. Before training, we initialize all weight variables based on normal distribution and set all bias variables to be zero vectors. We use the adaptive moment estimation (Adam) optimizer [14] to update the parameters for that it's very suitable for tasks with datasets of large dimension and needs less memory space, with an initial learning rate of 0.001 decayed by a factor of 0.95 every epoch until the learning rate falls below $5.0e-5$. We train our base model for 300 epoches and keep the best model with a lowest validation loss. Our model is implemented in PyTorch [15] and TensorFlow [16] framework, and all simulations are run on single NVIDIA Tesla P40 GPU and Intel Xeon CPU E5-2680. Our SRU-based framework is predominantly run on a single GPU, whereas the conventional ED-based method is run on a CPU totally. The hyper-parameters are presented in Table IV1.

In the transfer learning experiment, we select the base model and start training the new dataset using the same Adam optimizer with an initial learning rate of 0.0001 decayed by a factor of 0.97. The batch size is 16 and the total training epochs are 300.

In the main text we only showed the result of predicting state sequences of 100 timesteps. To demonstrate the ability of generating longer state sequences for different quantum many-body systems, we showed the performance of predicting state sequences as long as 250 timesteps evaluated by mean relative entropy (MRE). From Fig. IV5, the MRE increases linearly with the timesteps because of the error accumulation during the autoregressive generation. We will try to apply non-autoregressive generative model with inverse autoregressive flows[17] to better estimate the probability density of longer state sequences in our future work.

TABLE IV1: Network architecture hyper-parameters.

| layer | configuration |
|---|---|
| spin encoding embedding | size=1024, unit=512 |
| chain encoding embedding | size=1024, unit=512 |
| feed-forward layer | unit=512 |
| dropout layer | dropout rate=0.005 |
| feed-forward layer | unit=512 |
| SRU layer | layers=4, unit=512 |
| feed-forward layer | unit=512 |
| dropout layer | dropout rate=0.005 |
| spin decoding embedding | size=1024, unit=512 |
| batch size | 32 |
| learning rate (lr) | initial lr=0.001, decay rate=0.95 |

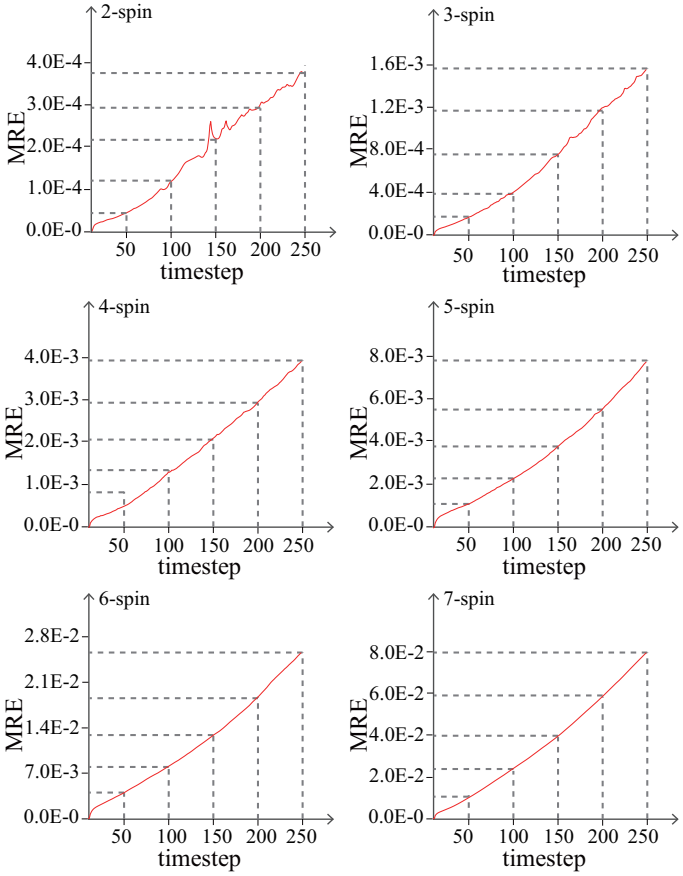


FIG. IV5: MRE of generating longer sequences of 250 timesteps in different systems. The MRE increases linearly with timesteps.

To quantify our model's performance by a physical variable, we draw the magnetization intensity calculated from both predicted (SRU) and simulated (ED) wavefunctions in Fig. IV6, which have a nice agreement. Specifically, for smaller-sized systems, such as 2-spin and 3-spin, the predicted magnetization intensity has a very nice agreement with simulated one. With the increase of spin variables, our SRU-based framework has a performance drop due to the exponentially increased computation complexity. Meanwhile, with the increase of timesteps, the difference between predicted and simulated magnetization intensity also becomes larger, which is due to the error accumulation during the autoregressive generation without any external guidance.

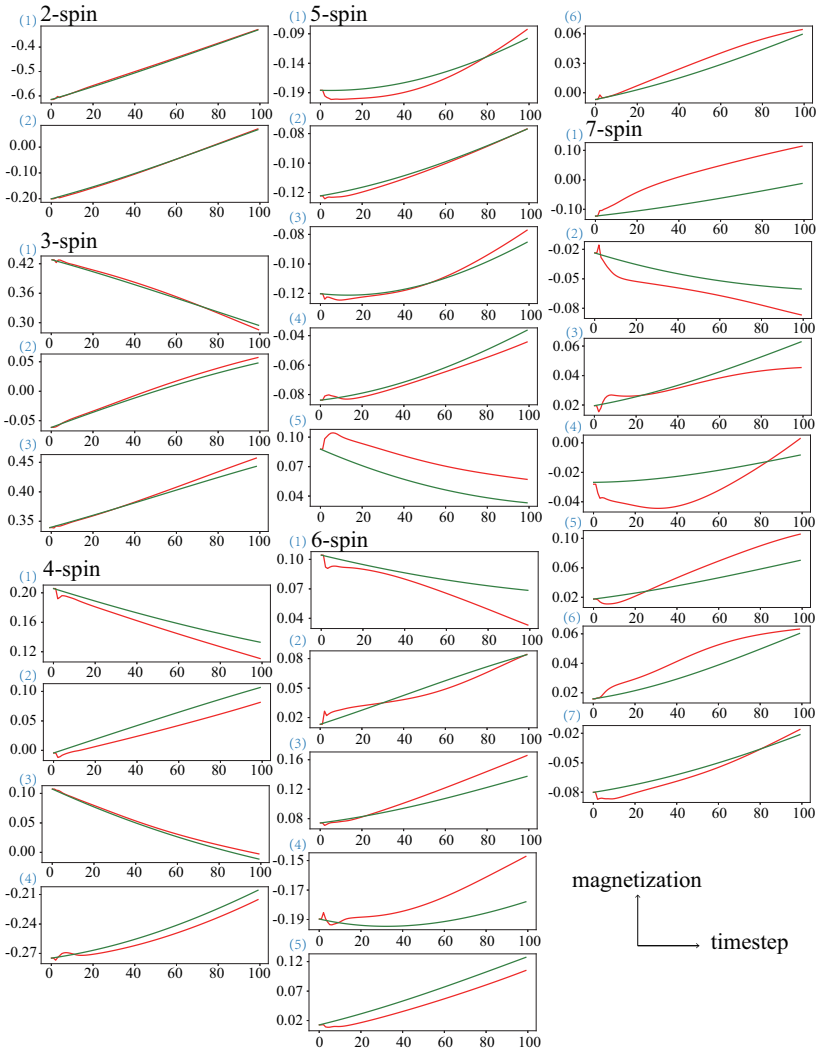


FIG. IV6: Magnetization varies with timestep calculated by both SRU (red curve) and ED (green curve) based wavefunctions for all six systems.

## V. INITIAL STATES USED FOR PREDICTION

### A. The initial state of two-spin system for sequence generation

(-0.078+0.203j)|00⟩+(-0.458+0.359j)|01⟩
+( 0.372+0.602j)|10⟩+(-0.274+0.197j)|11⟩

### B. The initial state of three-spin system for sequence generation

(-0.127+0.134j)|000⟩+(-0.012+0.357j)|001⟩
+( 0.218+0.271j)|010⟩+(-0.082+0.158j)|011⟩
+( 0.101+0.217j)|100⟩+(-0.175+0.499j)|101⟩
+(-0.253+0.274j)|110⟩+(-0.123+0.442j)|111⟩

### C. The initial state of four-spin system for sequence generation

(-0.091+0.008j)|0000⟩+(-0.141+0.123j)|0001⟩
+( 0.002+0.382j)|0010⟩+( 0.169+0.190j)|0011⟩
+( 0.015+0.153j)|0100⟩+( 0.042+0.028j)|0101⟩
+(-0.075+0.177j)|0110⟩+( 0.137+0.255j)|0111⟩
+(-0.236+0.037j)|1000⟩+(-0.085+0.331j)|1001⟩
+( 0.140+0.069j)|1010⟩+(-0.230+0.270j)|1011⟩
+( 0.071+0.319j)|1100⟩+(-0.162+0.272j)|1101⟩
+(-0.149+0.132j)|1110⟩+(-0.016+0.169j)|1111⟩

### D. The initial state of five-spin system for sequence generation

( 0.037+0.077j)|00000⟩+(-0.110+0.026j)|00001⟩
+( 0.072+0.028j)|00010⟩+(-0.026+0.036j)|00011⟩
+(-0.088+0.026j)|00100⟩+(-0.102+0.268j)|00101⟩
+(-0.099+0.253j)|00110⟩+(-0.042+0.073j)|00111⟩
+(-0.087+0.052j)|01000⟩+( 0.055+0.120j)|01001⟩
+( 0.003+0.128j)|01010⟩+(-0.086+0.181j)|01011⟩
+(-0.110+0.241j)|01100⟩+( 0.110+0.050j)|01101⟩
+( 0.107+0.249j)|01110⟩+(-0.021+0.083j)|01111⟩
+( 0.036+0.099j)|10000⟩+( 0.119+0.257j)|10001⟩
+(-0.072+0.160j)|10010⟩+(-0.133+0.033j)|10011⟩
+(-0.119+0.121j)|10100⟩+(-0.113+0.213j)|10101⟩
+(-0.026+0.105j)|10110⟩+(-0.081+0.038j)|10111⟩
+(-0.125+0.246j)|11000⟩+( 0.027+0.185j)|11001⟩
+(-0.009+0.112j)|11010⟩+(-0.037+0.204j)|11011⟩
+(-0.005+0.121j)|11100⟩+(-0.017+0.015j)|11101⟩
+(-0.126+0.214j)|11110⟩+(-0.091+0.228j)|11111⟩

### E. The initial state of six-spin system for sequence generation

( 0.067+0.060j)|000000⟩+(-0.094+0.072j)|000001⟩
+(-0.034+0.174j)|000010⟩+(-0.067+0.151j)|000011⟩
+( 0.005+0.069j)|000100⟩+(-0.037+0.151j)|000101⟩
+( 0.034+0.131j)|000110⟩+( 0.023+0.038j)|000111⟩
+(-0.064+0.013j)|001000⟩+(-0.084+0.030j)|001001⟩
+( 0.082+0.061j)|001010⟩+( 0.051+0.146j)|001011⟩
+( 0.043+0.022j)|001100⟩+( 0.001+0.141j)|001101⟩
+( 0.058+0.015j)|001110⟩+( 0.065+0.014j)|001111⟩
+(-0.041+0.035j)|010000⟩+( 0.007+0.131j)|010001⟩
+( 0.002+0.126j)|010010⟩+( 0.025+0.158j)|010011⟩
+( 0.042+0.013j)|010100⟩+(-0.006+0.006j)|010101⟩
+( 0.090+0.086j)|010110⟩+( 0.093+0.022j)|010111⟩
+( 0.082+0.154j)|011000⟩+(-0.011+0.114j)|011001⟩
+(-0.040+0.081j)|011010⟩+(-0.004+0.107j)|011011⟩
+(-0.088+0.199j)|011100⟩+(-0.053+0.074j)|011101⟩
+( 0.093+0.159j)|011110⟩+(-0.088+0.119j)|011111⟩
+( 0.055+0.134j)|100000⟩+( 0.015+0.126j)|100001⟩
+( 0.031+0.006j)|100010⟩+( 0.069+0.094j)|100011⟩
+( 0.003+0.180j)|100100⟩+(-0.058+0.109j)|100101⟩
+(-0.034+0.075j)|100110⟩+( 0.071+0.022j)|100111⟩

+( 0.031+0.153j)|101000⟩+( 0.033+0.191j)|101001⟩
+(-0.046+0.193j)|101010⟩+( 0.025+0.013j)|101011⟩
+( 0.062+0.165j)|101100⟩+(-0.045+0.068j)|101101⟩
+( 0.048+0.111j)|101110⟩+( 0.001+0.065j)|101111⟩
+(-0.008+0.029j)|110000⟩+(-0.071+0.106j)|110001⟩
+( 0.092+0.185j)|110010⟩+(-0.066+0.075j)|110011⟩
+(-0.001+0.129j)|110100⟩+(-0.075+0.030j)|110101⟩
+(-0.031+0.079j)|110110⟩+( 0.005+0.056j)|110111⟩
+( 0.088+0.080j)|111000⟩+(-0.076+0.084j)|111001⟩
+( 0.029+0.003j)|111010⟩+(-0.072+0.074j)|111011⟩
+(-0.076+0.175j)|111100⟩+(-0.075+0.009j)|111101⟩
+(-0.009+0.187j)|111110⟩+( 0.032+0.199j)|111111⟩

### F. The initial state of seven-spin system for sequence generation

( 0.067+0.007j)|0000000⟩+( 0.018+0.119j)|0000001⟩
+(-0.013+0.033j)|0000010⟩+( 0.050+0.115j)|0000011⟩
+(-0.039+0.098j)|0000100⟩+(-0.036+0.055j)|0000101⟩
+(-0.020+0.011j)|0000110⟩+(-0.005+0.073j)|0000111⟩
+(-0.004+0.125j)|0001000⟩+(-0.050+0.098j)|0001001⟩
+( 0.010+0.111j)|0001010⟩+( 0.056+0.085j)|0001011⟩
+(-0.008+0.092j)|0001100⟩+( 0.017+0.003j)|0001101⟩
+( 0.057+0.012j)|0001110⟩+(-0.040+0.058j)|0001111⟩
+( 0.021+0.053j)|0010000⟩+(-0.047+0.053j)|0010001⟩
+( 0.057+0.111j)|0010010⟩+( 0.052+0.006j)|0010011⟩
+( 0.010+0.044j)|0010100⟩+(-0.010+0.105j)|0010101⟩
+( 0.052+0.128j)|0010110⟩+(-0.005+0.072j)|0010111⟩
+(-0.034+0.119j)|0011000⟩+(-0.015+0.083j)|0011001⟩
+( 0.065+0.100j)|0011010⟩+(-0.018+0.009j)|0011011⟩
+( 0.019+0.072j)|0011100⟩+(-0.065+0.129j)|0011101⟩
+( 0.035+0.013j)|0011110⟩+( 0.040+0.117j)|0011111⟩
+( 0.011+0.114j)|0100000⟩+( 0.059+0.039j)|0100001⟩
+( 0.057+0.035j)|0100010⟩+( 0.034+0.108j)|0100011⟩
+( 0.066+0.131j)|0100100⟩+(-0.002+0.078j)|0100101⟩
+( 0.009+0.108j)|0100110⟩+( 0.020+0.087j)|0100111⟩
+( 0.010+0.069j)|0101000⟩+( 0.018+0.090j)|0101001⟩
+(-0.052+0.070j)|0101010⟩+( 0.031+0.022j)|0101011⟩
+( 0.001+0.106j)|0101100⟩+(-0.063+0.062j)|0101101⟩
+(-0.049+0.057j)|0101110⟩+(-0.014+0.043j)|0101111⟩
+(-0.048+0.029j)|0110000⟩+( 0.047+0.049j)|0110001⟩
+( 0.002+0.097j)|0110010⟩+( 0.065+0.124j)|0110011⟩
+( 0.026+0.034j)|0110100⟩+(-0.014+0.083j)|0110101⟩
+( 0.021+0.080j)|0110110⟩+(-0.051+0.001j)|0110111⟩
+(-0.052+0.044j)|0111000⟩+( 0.002+0.092j)|0111001⟩
+( 0.053+0.050j)|0111010⟩+( 0.063+0.013j)|0111011⟩
+( 0.023+0.082j)|0111100⟩+(-0.014+0.001j)|0111101⟩
+( 0.011+0.123j)|0111110⟩+(-0.025+0.066j)|0111111⟩
+(-0.038+0.019j)|1000000⟩+(-0.000+0.049j)|1000001⟩
+(-0.034+0.018j)|1000010⟩+(-0.056+0.056j)|1000011⟩
+(-0.028+0.047j)|1000100⟩+(-0.016+0.057j)|1000101⟩
+( 0.032+0.094j)|1000110⟩+( 0.056+0.114j)|1000111⟩
+(-0.006+0.132j)|1001000⟩+( 0.042+0.071j)|1001001⟩
+( 0.011+0.036j)|1001010⟩+( 0.034+0.093j)|1001011⟩
+( 0.040+0.047j)|1001100⟩+(-0.003+0.069j)|1001101⟩

+(-0.022+0.075j)|1001110⟩+( 0.030+0.078j)|1001111⟩
+(-0.014+0.049j)|1010000⟩+(-0.059+0.120j)|1010001⟩
+( 0.024+0.050j)|1010010⟩+( 0.008+0.118j)|1010011⟩
+(-0.045+0.047j)|1010100⟩+( 0.060+0.047j)|1010101⟩
+( 0.030+0.089j)|1010110⟩+(-0.031+0.007j)|1010111⟩
+( 0.066+0.051j)|1011000⟩+( 0.035+0.001j)|1011001⟩
+(-0.010+0.055j)|1011010⟩+(-0.062+0.043j)|1011011⟩
+( 0.039+0.015j)|1011100⟩+(-0.015+0.084j)|1011101⟩
+( 0.018+0.124j)|1011110⟩+( 0.021+0.029j)|1011111⟩
+( 0.037+0.061j)|1100000⟩+( 0.053+0.027j)|1100001⟩
+( 0.059+0.127j)|1100010⟩+(-0.037+0.064j)|1100011⟩
+(-0.041+0.072j)|1100100⟩+( 0.007+0.121j)|1100101⟩
+(-0.054+0.045j)|1100110⟩+( 0.010+0.063j)|1100111⟩
+(-0.041+0.122j)|1101000⟩+(-0.064+0.089j)|1101001⟩
+( 0.037+0.053j)|1101010⟩+(-0.050+0.085j)|1101011⟩
+( 0.067+0.120j)|1101100⟩+( 0.030+0.055j)|1101101⟩
+(-0.007+0.063j)|1101110⟩+(-0.062+0.022j)|1101111⟩
+( 0.032+0.077j)|1110000⟩+( 0.044+0.102j)|1110001⟩
+( 0.067+0.116j)|1110010⟩+( 0.061+0.009j)|1110011⟩
+(-0.062+0.122j)|1110100⟩+(-0.065+0.037j)|1110101⟩
+(-0.015+0.024j)|1110110⟩+( 0.049+0.067j)|1110111⟩
+(-0.027+0.082j)|1111000⟩+( 0.041+0.003j)|1111001⟩
+( 0.062+0.074j)|1111010⟩+(-0.049+0.118j)|1111011⟩
+( 0.034+0.121j)|1111100⟩+( 0.022+0.063j)|1111101⟩
+(-0.008+0.132j)|1111110⟩+(-0.066+0.069j)|1111111⟩

---

* zhangxiao@mail.sysu.edu.cn


[1] X. Glorot and Y. Bengio, in *Proceedings of the thirteenth international conference on artificial intelligence and statistics* (2010) pp. 249–256.
[2] F. A. Gers, J. Schmidhuber, and F. Cummins, (1999).
[3] R. K. Srivastava, K. Greff, and J. Schmidhuber, arXiv preprint arXiv:1505.00387 (2015).
[5] T. Lei, Y. Zhang, S. I. Wang, H. Dai, and Y. Artzi, in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing* (2018) pp. 4470–4481.
[6] G. Wang, Y. Lin, and W. Yi, in *2010 IEEE/ACM Int'l Conference on Green Computing and Communications & Int'l Conference on Cyber, Physical and Social Computing* (IEEE, 2010) pp. 344–350.
[7] J. Filipovič, M. Madzin, J. Fousek, and L. Matyska, The Journal of Supercomputing **71**, 3934 (2015).
[8] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, The journal of machine learning research **15**, 1929 (2014).
[9] S. Bengio, O. Vinyals, N. Jaitly, and N. Shazeer, in *Advances in Neural Information Processing Systems* (2015) pp. 1171–1179.
[10] A. M. Lamb, A. G. A. P. Goyal, Y. Zhang, S. Zhang, A. C. Courville, and Y. Bengio, in *Advances In Neural Information Processing Systems* (2016) pp. 4601–4609.
[11] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, in *Advances in neural information processing systems* (2014) pp. 2672–2680.
[12] H. Guo, F. K. Soong, L. He, and L. Xie, arXiv preprint arXiv:1904.04775 (2019).
[4] R. J. Williams and D. Zipser, Neural computation **1**, 270 (1989).
[13] R. Moessner, S. L. Sondhi, and P. Chandra, Physical Review Letters **84**, 4457 (2000).
[14] D. P. Kingma and J. Ba, arXiv preprint arXiv:1412.6980 (2014).
[15] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, (2017).
[16] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, in *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)* (2016) pp. 265–283.
[17] D. P. Kingma, T. Salimans, R. Jozefowicz, X. Chen, I. Sutskever, and M. Welling, in *Advances in Neural Information Processing Systems* (2016) pp. 4743–4751.