Neural Differential Recurrent Neural Network with Adaptive Time Steps

Yixuan Tan[1], Liyan Xie[2], and Xiuyuan Cheng*[1]

[1]Department of Mathematics, Duke University
[2]School of Data Science, The Chinese University of Hong Kong, Shenzhen

Abstract

The neural Ordinary Differential Equation (ODE) model has shown success in learning continuous-time processes from observations on discrete time stamps. In this work, we consider the modeling and forecasting of time series data that are non-stationary and may have sharp changes like spikes. We propose an RNN-based model, called *RNN-ODE-Adap*, that uses a neural ODE to represent the time development of the hidden states, and we adaptively select time steps based on the steepness of changes of the data over time so as to train the model more efficiently for the "spike-like" time series. Theoretically, *RNN-ODE-Adap* yields provably a consistent estimation of the intensity function for the Hawkes-type time series data. We also provide an approximation analysis of the RNN-ODE model showing the benefit of adaptive steps. The proposed model is validated on simulated spiral data, point process data, and a real electrocardiography dataset. It was shown to achieve higher prediction accuracy with a reduced computational cost.

1 Introduction

We consider the modeling and forecasting of time series with *irregular* time steps and *non-stationary* patterns, as commonly observed in various applications [10, 3]. We treat the data as a sequence of observations from an unknown continuous-time process, sampled at discrete times. We follow the previous continuous-time RNN neural-ODE approach [1, 6] to model the hidden dynamics $h(t)$ as

$$h'(t) = f(h(t), x(t); \theta_h), \tag{1}$$

where f is a neural network parameterized by θ_h. The model (1) incorporates the observed incoming time series data $x(t)$ as an input to f. The time evolution of the observed series $x(t)$ is modeled by an output neural network g that maps the hidden value $h(t)$ to $x(t)$ as

$$\hat{x}(t) = g(h(t); \theta_d), \tag{2}$$

where g is called the output function parameterized by θ_d.

For highly non-stationary data, such as a time series with sudden spikes (see examples in Figure 1 (Left)), it becomes imperative for the classical neural ODE type approaches to select sufficiently small time steps to ensure accurate modeling. To train the neural ODE for data with non-stationary patterns more effectively, we propose an approach that employs *adaptive time steps* in the neural ODE model, which we refer to as *RNN-ODE-Adap*. The model adaptively selects the time steps based on the local variation of the time series, enabling it to capture underlying trends with potentially fewer time steps. The algorithm is described and analyzed in Section 2.

2 Method and Theory

2.1 Adaptive Time Steps

*Email: xiuyuan.cheng@duke.edu

DLDE-III Workshop in the 37th Conference on Neural Information Processing Systems (NeurIPS 2023).

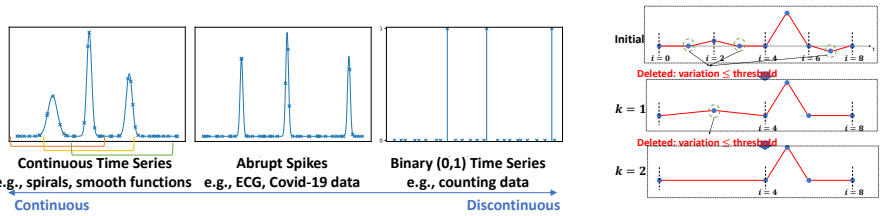

Figure 1: Left: Illustration of spike-like time series. The crosses denote the discrete (irregular) time steps. The subsequences enclosed with brackets represent training windows. Right: Illustration of adaptive time steps constructed from Algorithm 1. In this example, $N = 8$ and $L = 2$.

The construction of adaptive time steps in *RNN-ODE-Adap* is summarized in Algorithm 1. The intuition is to assign longer (rough) time intervals during time regions where the time series is slowly time-varying (such as "flat" curves), while assigning shorter (fine) time intervals during regions with "spikes" (highly non-stationary and fast time-varying regimes). For constructing the adaptive time stamps, we assume the initial time grid is sufficiently fine and adopt a dyadic-partition type algorithm detailed below.

Algorithm 1 A dyadic algorithm for selecting adaptive time steps.

Input: Data series $\{x_0, x_1, x_2, \ldots, x_N\}$; threshold $\epsilon > 0$; $L \in \mathbb{Z}_+$.
Initialize: $\mathcal{D} = \emptyset$. A flag vector $\texttt{Flag} = \{0, 0, \ldots, 0\}$ of length N.
for $l = 1$ **to** L **do**
  Define a new flag: $\texttt{Flag}_{\texttt{new}} = \{0, 0, \ldots, 0\}$ of length $\lfloor N/2^l \rfloor$.
  for $i = 1$ **to** $\lfloor N/2^l \rfloor$ **do**
    if $\texttt{Flag}[2(i-1)+1] = \texttt{Flag}[2i] = 0$ **then**
      Compute the monitoring function $M(\{x_{2^l(i-1)}, x_{2^l(i-1)+2^{l-1}}, x_{2^l i}\})$.
      if $M < \epsilon$ **then**
        $\mathcal{D} = \mathcal{D} \cup (2^l(i-1) + 2^{l-1})$.
      else
        Mark $\texttt{Flag}_{\texttt{new}}[i] = 1$.
      end if
    else
      Mark $\texttt{Flag}_{\texttt{new}}[i] = 1$.
    end if
  end for
  Update $\texttt{Flag} = \texttt{Flag}_{\texttt{new}}$.
end for
Output: Indexes of removed time steps $\mathcal{D}$.

Given a raw (discrete-time) training window $x(t_0), \ldots, x(t_N)$ sampled at the finest level of time stamps $0 \le t_0 < \cdots < t_N \le T$. For simplicity, below we write it as $x_0, x_1, x_2, \ldots, x_N$. We first define a monitor function $M(\cdot)$ that measures the variation of the sub-sequence $\{x_i, \ldots, x_j\}$, $i < j$. In this paper, we mainly adopt the *maximum variation* defined as

$$M(\{x_i, \ldots, x_j\}) := \max_{i<k\le j} \frac{\|x_k - x_{k-1}\|_2}{|t_k - t_{k-1}|}, \quad (3)$$

which captures the maximum variation among any two adjacent time stamps. We then screen from the finest level of time grids and adaptively merge neighboring time grids if their maximum variation is below a pre-specified threshold $\epsilon > 0$. The selection procedure is repeated similarly for $l = 1, 2, 3, \ldots$ until a pre-specified maximum integer L. In other words, we only keep the time stamps on which the maximum variation exceeds ϵ.

The output of Algorithm 1 is the set $\mathcal{D}$ of removed time stamps. The model is trained on the remaining time steps only. We provide an illustration in Figure 1 of the algorithm for selecting adaptive time steps. From the final results in Figure 1 (Right), it can be seen that the output of the adaptive time steps uses longer time steps to model stationary periods and uses *shorter* time steps to model *spikes*.

2.2 Approximation Analysis of RNN-ODE-Adap

In this section, we analyze the approximation error of the RNN-ODE model, revealing the benefit of adaptive step size. Further details on the theory, including consistency estimation for binary event data, continuous-time model approximation, and additional content from this section, can be found in Appendix B. All proofs can be found in Appendix C.

For theoretical generality, we consider the continuous-time process $y(t) \in \mathbb{R}^{D'}$ satisfying

$$h'(t) = f(h(t), x(t)), \quad y(t) = g(h(t)), \quad h(0) = h_0, \quad t \in [0, T], \quad (4)$$

where $x(t) \in \mathbb{R}^D$ is the observable input data and $h(t) \in \mathbb{R}^{d_h}$ is the underlying hidden process. Taking $y(t)$ to be $x(t)$ reduces the model to the case (1)-(2) considered in the other parts of the work. We assume that $x(t)$ is only observed at discrete time grids $\{t_i\}_{i=1}^N$ and the time grids can be chosen

adaptively. For $\Delta t_i := t_i - t_{i-1}$ and $\hat{h}_{\text{NN}}(0) = h_0$, the forward Euler scheme is applied on the approximation of (4) by neural networks as follows:

$$\hat{h}_{\text{NN}}(t_i) = \hat{h}_{\text{NN}}(t_{i-1}) + \Delta t_i f_\theta(\hat{h}_{\text{NN}}(t_{i-1}), x(t_{i-1})),\ \hat{y}_{\text{NN}}(t_i) = g_\phi(\hat{h}_{\text{NN}}(t_i)),\ i = 1, \ldots, N. \quad (5)$$

Theorem 2.1 below provides an upper bound of the approximation error using $x(t)$ observed at discrete time grids.

Theorem 2.1. *Under mild Assumption B.8 on the Lipschitz continuity and given a time grid* $\{t_i\}_{i=1}^N$ *on* $[0, T]$ *at which* $x(t)$ *is observed. Suppose* $\epsilon_f, \epsilon_g > 0$ *and* $\epsilon_f, \{\Delta t_j\}_j$ *satisfy* $T \exp(\sum_{i=1}^N L_i^{f,h} \Delta t_i)(\epsilon_f + \max_j\{\mu_j \Delta t_j\}) < 0.1$. *Let* f_θ, g_ϕ *be the neural networks approximating* f, g *up to* ϵ_f, ϵ_g *in* L_∞ *norm, then*

$$\max_i \|y(t_i) - \hat{y}_{NN}(t_i)\| \leq \epsilon_g + L_g T \exp(\sum_{i=1}^N L_i^{f,h} \Delta t_i) \left(\epsilon_f + \max_j\{\mu_j \Delta t_j\}\right), \quad (6)$$

where L_g, $\{L_j^{f,h}\}_j$ *and* $\{\mu_j\}_j$ *denote the (global or local) Lipschitz constants of* g, f, *and* $x(t)$.

Theorem 2.1 provides insights into the utility of the adaptive steps for improving the model fitting performance, which is reflected in the last term in Eq. (6) involving $\max_j\{\mu_j \Delta t_j\}$. Specifically, time grids may be selected such that Δt_i is small if L_i^x is great, indicating a steep change in $x(t)$ for $t \in [t_{i-1}, t_i]$. On the contrary, when the variation in $x(t)$ is smaller, we employ larger Δt_i to reduce the total number of required time grids.

3 Numerical Experiments

We validate the performance of the proposed method using three datasets: (1) the simulated spiral series, (2) the simulated event data, and (3) a real ECG dataset. Experimental details can be found in Appendix D. We examine and report the performance of two models, *RNN-ODE* and *RNN-ODE-Adap*, both are trained to minimize the weighted Mean Squared Errors (MSEs) defined in (A13). Their difference lies in the choice of the time grid: *RNN-ODE* is trained using regular (non-adaptive) time steps, while *RNN-ODE-Adap* is trained with adaptive time steps from Algorithm 1.

We compare the tradeoff curves, between complexity (average length of the training windows) and accuracy, for different methods and demonstrate the advantage of *RNN-ODE-Adap*.

Simulated Spiral Data: We first investigate the capability of our method to fit and capture the underlying dynamics of the simulated spiral data. For a given matrix $A \in \mathbb{R}^{2\times 2}$, one spiral is generated by integrating the ODE

$$x'(t) = f(x(t)) = \|x(t)\|^{-2} A x(t), \quad (7)$$

over the time span $[0, T]$, with the initial value $x(0) = x_0 \in \mathbb{R}^2$. The initial training and testing windows are of length $N = 64$, corresponding to the largest complexity shown in Figure 2.

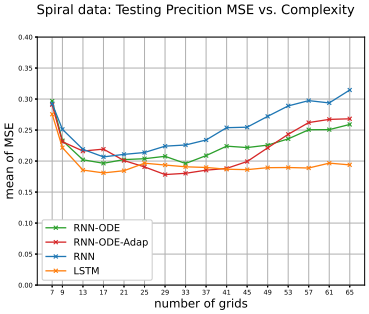

Figure 2: Comparison of the MSE prediction errors on the simulated spiral data for RNN, LSTM, RNN-ODE, RNN-ODE-Adap. The x-axis is the average length of the training windows, reflecting the complexity of the models (see Appendix D.2).

We first compare the on-sample prediction performance with RNN and LSTM [14] under different computational complexities. For each testing window, we use the first half as historical data and predict the second half. Figure 2 shows the averaged MSEs computed as in Eq. (A15) for the models, with varying complexities. We observe from Figure 2 that for the same complexity, RNN-ODE significantly improves the forecasting performance compared to the vanilla RNN, especially when the number of grids is not too small so that the models begin to learn the dynamics well. Additionally, RNN-ODE-Adap further achieves smaller prediction errors than RNN-ODE since it selects data points more informatively with the same number of grids. Finally, we note that while LSTM performs best in most cases, it possesses a more complex network structure. We refer to Appendix D.7 for additional results about the LSTM and Lipschitz-RNN [6] variants of the adaptive model. We also present the reconstructions performance in Appendix D.7 (Figure A6).

Simulated Point-Process Data: We further apply our method to a simulated example of event times data generated from temporal Hawkes processes [23] as described in Appendix B.1. We train the model (1)-(2) and estimate the true intensity function $\lambda(t)$ using the output $x(t)$. The mean squared loss $L = \int_0^T (dN(t)/dt - \lambda(t))^2 dt$ is used when fitting the neural ODE model.

The fitting errors of the four models versus the complexity are shown in Figure 3 (left). It can be observed that RNN-ODE-Adap achieves the best fitting performance. The right panel shows the log-log plot of RNN-ODE and RNN-ODE-Adap, from which we can see more clearly that for fixed model complexity (network structure), the proposed model recovers the true intensity function. Figure 3 (right) shows two examples of fitting performance. In this example, all models use 33 grids on average. Thus, the complexity is 50% of the largest one. It can be observed that RNN fails to capture the smooth decay of the kernel, while RNN-ODE-Adap can learn the dynamics of the intensity function much better – it can estimate the "jump" in the intensity accurately.

Real ECG Data: We validate the proposed RNN-ODE-Adap on one public electrocardiography (ECG) dataset PTB-XL [30, 9]. We focus on learning the underlying dynamics of ECG signals and use adaptive time steps for "spikes" in data series. We remark that windows of the highest sampling rate are chosen to have $N = 96$ time grids, in which usually two cycles are contained. In this way, the prediction of the second half given the first half would be more meaningful.

Figure 4 (left) shows the on-sample prediction MSEs of the four methods for two different prediction lengths, 24 and 48, which are $1/4$ and $1/2$ of the whole window. Here, the prediction is performed with the original finest grids by integrating the ODE function. It can be seen that RNN-ODE has smaller prediction errors than RNN on average, and adding addictive steps helps achieve slightly better performance. Furthermore, LSTM still achieves the smallest error most of the time. The reason for this is similar to the spiral data and may be due to its more complex network structure.

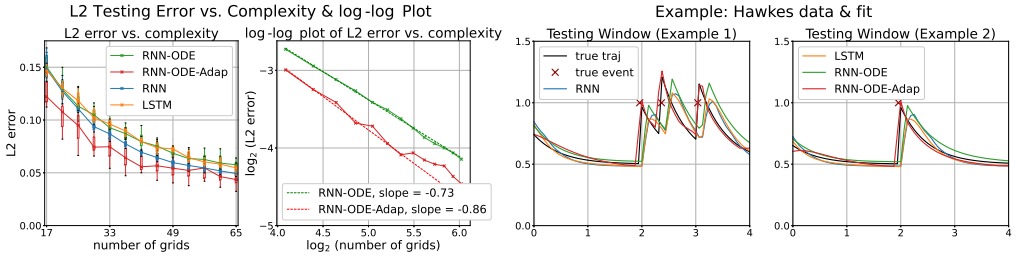

Figure 3: Left: Comparison of the fitting errors of the underlying intensity function of the simulated event-type data generated from the Hawkes process. x-axis represents computational complexity, y-axis is the fitting error computed as in Eq. (A16). Right: Examples of fitted intensity function of the simulated event times data generated from the Hawkes process.

Figure 4 (right) and Figure A13 present examples of prediction on the testing windows. These examples demonstrate that RNN-ODE-Adap captures the trends of the ECG more effectively than RNN.The good performance implies that the proposed algorithm could be used to fit and predict the ECG-type signal well.

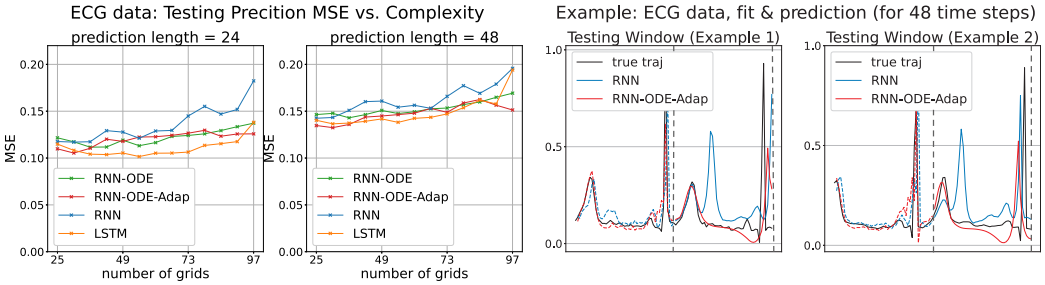

Figure 4: Left: Comparison of the prediction errors on the real ECG data under two different prediction lengths (24 and 48) for RNN, LSTM, RNN-ODE, RNN-ODE-Adap. Right: Examples of 48 steps ahead prediction for the testing ECG data using RNN (marked in blue) and RNN-ODE-Adap (marked in red). The predicted region is marked between dashed lines.

References

[1] Bo Chang, Minmin Chen, Eldad Haber, and Ed H Chi. AntisymmetricRNN: A dynamical system view on recurrent neural networks. *arXiv preprint arXiv:1902.09689*, 2019.

[2] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

[3] Changqing Cheng, Akkarapol Sa-Ngasoongsong, Omer Beyca, Trung Le, Hui Yang, Zhenyu Kong, and Satish TS Bukkapatnam. Time series forecasting for nonlinear and non-stationary processes: a review and comparative study. *Iie Transactions*, 47(10):1053–1071, 2015.

[4] Tommy WS Chow and Xiao-Dong Li. Modeling of continuous time dynamical systems with input by recurrent neural networks. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 47(4):575–578, 2000.

[5] Chengyu Dong, Liyuan Liu, Zichao Li, and Jingbo Shang. Towards adaptive residual network training: A Neural-ODE perspective. In *International conference on machine learning*, pages 2616–2626. PMLR, 2020.

[6] N Benjamin Erichson, Omri Azencot, Alejandro Queiruga, Liam Hodgkinson, and Michael W Mahoney. Lipschitz recurrent neural networks. *arXiv preprint arXiv:2006.12070*, 2020.

[7] Richard FitzHugh. Mathematical models of threshold phenomena in the nerve membrane. *The bulletin of mathematical biophysics*, 17:257–278, 1955.

[8] Ken-ichi Funahashi and Yuichi Nakamura. Approximation of dynamical systems by continuous time recurrent neural networks. *Neural networks*, 6(6):801–806, 1993.

[9] Ary L Goldberger, Luis AN Amaral, Leon Glass, Jeffrey M Hausdorff, Plamen Ch Ivanov, Roger G Mark, Joseph E Mietus, George B Moody, Chung-Kang Peng, and H Eugene Stanley. PhysioBank, PhysioToolkit, and PhysioNet: components of a new research resource for complex physiologic signals. *circulation*, 101(23):e215–e220, 2000.

[10] Gloria Gonzalez-Rivera and Javier Arroyo. Time series modeling of histogram-valued data: The daily histogram time series of S&P500 intradaily returns. *International Journal of Forecasting*, 28(1):20–33, 2012.

[11] Sam Greydanus, Stefan Lee, and Alan Fern. Piecewise-constant Neural ODEs. *arXiv preprint arXiv:2106.06621*, 2021.

[12] Mansura Habiba and Barak A Pearlmutter. Neural ordinary differential equation based recurrent neural network model. In *2020 31st Irish Signals and Systems Conference (ISSC)*, pages 1–6. IEEE, 2020.

[13] Stefan Heinrich, Tayfun Alpay, and Yukie Nagai. Learning timescales in gated and adaptive continuous time recurrent neural networks. In *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 2662–2667. IEEE, 2020.

[14] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[15] Anil Kag and Venkatesh Saligrama. Time adaptive recurrent neural network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15149–15158, 2021.

[16] Anil Kag, Ziming Zhang, and Venkatesh Saligrama. RNNs incrementally evolving on an equilibrium manifold: A panacea for vanishing and exploding gradients? In *International Conference on Learning Representations*, 2020.

[17] Patrick Kidger, James Morrill, James Foster, and Terry Lyons. Neural controlled differential equations for irregular time series. *Advances in Neural Information Processing Systems*, 33:6696–6707, 2020.

[18] Patrick Kidger, James Morrill, James Foster, and Terry Lyons. Neural controlled differential equations for irregular time series. *Advances in Neural Information Processing Systems*, 33:6696–6707, 2020.

[19] Xiao-Dong Li, John KL Ho, and Tommy WS Chow. Approximation of dynamical time-variant systems by continuous-time recurrent neural networks. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 52(10):656–660, 2005.

[20] Zhong Li, Jiequn Han, E Weinan, and Qianxiao Li. Approximation and optimization theory for linear continuous-time recurrent neural networks. *J. Mach. Learn. Res.*, 23:42–1, 2022.

[21] Yiping Lu, Aoxiao Zhong, Quanzheng Li, and Bin Dong. Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations. In *International Conference on Machine Learning*, pages 3276–3285. PMLR, 2018.

[22] James Morrill, Cristopher Salvi, Patrick Kidger, and James Foster. Neural rough differential equations for long time series. In *International Conference on Machine Learning*, pages 7829–7838. PMLR, 2021.

[23] Jakob Gulddahl Rasmussen. Temporal point processes: The conditional intensity function. *Lecture Notes, Jan*, 2011.

[24] Frank Rosenblatt. *Principles of neurodynamics.* Spartan Books, 1962.

[25] Yulia Rubanova, Ricky TQ Chen, and David K Duvenaud. Latent ordinary differential equations for irregularly-sampled time series. *Advances in neural information processing systems*, 32, 2019.

[26] T Konstantin Rusch and Siddhartha Mishra. Coupled oscillatory recurrent neural network (coRNN): An accurate and (gradient) stable architecture for learning long time dependencies. *arXiv preprint arXiv:2010.00951*, 2020.

[27] T Konstantin Rusch, Siddhartha Mishra, N Benjamin Erichson, and Michael W Mahoney. Long expressive memory for sequence modeling. *arXiv preprint arXiv:2110.04744*, 2021.

[28] T Konstantin Rusch, Siddhartha Mishra, N Benjamin Erichson, and Michael W Mahoney. Long expressive memory for sequence modeling. *arXiv preprint arXiv:2110.04744*, 2021.

[29] Hiroaki Sakoe and Seibi Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE transactions on acoustics, speech, and signal processing*, 26(1):43–49, 1978.

[30] Patrick Wagner, Nils Strodthoff, Ralf-Dieter Bousseljot, Dieter Kreiseler, Fatima I Lunze, Wojciech Samek, and Tobias Schaeffter. PTB-XL, a large publicly available electrocardiography dataset. *Scientific data*, 7(1):1–15, 2020.

[31] E Weinan. A proposal on machine learning via dynamical systems. *Communications in Mathematics and Statistics*, 1(5):1–11, 2017.

[32] Dmitry Yarotsky. Error bounds for approximations with deep relu networks. *Neural Networks*, 94:103–114, 2017.

[33] Tianjun Zhang, Zhewei Yao, Amir Gholami, Joseph E Gonzalez, Kurt Keutzer, Michael W Mahoney, and George Biros. ANODEV2: A coupled Neural ODE framework. *Advances in Neural Information Processing Systems*, 32, 2019.

A Related Works

The related works are provided in this section.

Neural ODE. Our work is closely related to the neural ODE [2] model, which parameterizes the derivative of the hidden state using a neural network. In [2], a generative time-series model was proposed, which takes the neural ODE as the decoder. Furthermore, [25] proposed a non-generative model with continuous-time hidden dynamics to handle irregularly sampled data based on [2]. Compared with existing works related to neural ODE [25, 31, 33, 5, 21, 17, 22, 12, 11], we model the ODE that determines the progression of hidden states by including the data itself in the derivative of the hidden state. In contrast to existing works on non-stationary environments such as the piecewise-constant ODE [11], our work proposes to use adaptive time steps to automatically adapt to sparse spikes in the time series, without pre-defining the time period for each piece of ODE.

Neural CDE. We note that the Neural Controlled Differential Equation (CDE) [18] also incorporates the observations into the model continuously. Specifically, the hidden states in [18] follow the CDE $h(t) = h(t_0) + \int_{t_0}^{t} f_\theta(h(s))\mathrm{d}X_s$, where the integral is a Riemann-Stieltjes integral. We would like to emphasize some key differences between model (1) and Neural CDE. The X_s in Neural CDE is the natural cubic spline of $\{(x(t_i), t_i)\}_i$, and $f_\theta : \mathbb{R}^{d_h} \to \mathbb{R}^{d_h \times (D+1)}$, where d_h is the number of hidden units and D is the data dimension. Thus, for the same number of hidden units, Neural CDE requires a more complex parameterized f_θ to model $h(t)$. Moreover, since X_s is obtained by cubic spline, it is less naturally adapted to the prediction task that requires extrapolation to the time stamps not seen when computing the spline. Therefore, it is hard to evaluate the prediction performance of Neural CDE and thus we defer the evaluation under the Neural CDE setting for future work.

Continuous-Time RNNs. Our model belongs to the extensive family of continuous-time RNNs, originating from [24]. Several existing studies explore various RNN architectures, such as [1, 16, 6, 26, 15] These RNN models leverage their structures to address the exploding and vanishing gradient problem. Our model also adopts a continuous-time ODE framework for time series data, and the proposed adaptive time stamp selection method can be viewed as effectively reducing the length of the discrete sequence when a significant part of the process is changing slowly. Meanwhile, our approach can also be used concurrently with the methodologies such as in [6]. As the focus of our work is to model the "spike-like" time series data, the combination of our model and the existing continuous-time RNN models can further improve the efficiency when applied to such data.

Time Adaptivity. Previous studies have investigated the incorporation of time adaptivity in continuous-time RNNs, such as GACTRNN [13], TARNN [15], and LEM [27]. In these works, time adaptivity was incorporated by multiplying the ODE with an adaptively learned time modulator, usually parametrized by another sub-network. In contrast, our method adaptively selects time steps during the preprocessing phase, where the selection process only utilizes the steepness of change of the time series data. Therefore, the proposed model does not involve the training of a sub-network for the time modulator as in the previous models, which may incur an increase in model size and additional computational costs.

B More Details on Theory

B.1 Function Estimation for Event-type Data

We present the theoretical analysis for function estimation based on the proposed model under counting-type time series. Counting-type time series represent a special class of continuous-time models since they exemplify the *extreme* case of "spike-like" data, as shown at the right end of Figure 1.

For event-type sequences, the raw data contains a list of event times $0 < t_1 < t_2 < \ldots < t_n < T$ on the time horizon $[0, T]$. Each timestamp is the time when an event happens. In practice, the estimation is performed on *discrete-time grids*. Define the counting process $N(t) := \sum_{i=1}^{n} \mathbf{1}(t_i \leq t)$ as the total number of events happened before time t. We convert such continuous-time data into discrete observations by discretizing the time interval $[0, T]$ into M intervals of equal length $\Delta t = T/M$, and

then let $x_m = N(m\Delta t) - N((m-1)\Delta t)$, $m = 0, 1, \ldots, M$ (by convention $x_0 = 0$). When Δt is chosen sufficiently small, it becomes the Bernoulli process where $x_i \in \{0, 1\}$.

We consider the temporal Hawkes processes [23], in which the values x_i are mostly zero under mild assumptions, corresponding to sparse "spikes". Such temporal Hawkes processes can be characterized by its *conditional intensity* function defined as

$$\lambda^*(t) = \lim_{\Delta \to 0} \Delta^{-1} \mathbb{E}[N(t+\Delta) - N(t) | \mathcal{F}_t],$$

where the filtration $\mathcal{F}_t$ stands for the information available up to time t. In the case of Hawkes processes, $\lambda(t) = \mu + \alpha \int_0^t \phi(t-s) dN(s)$ is simply a linear function of past jumps of the process, where $\phi(\cdot)$ is the influence kernel. For example, under the special case of exponential kernels, the intensity function becomes $\lambda^*(t) = \mu + \alpha\beta \int_0^t e^{-\beta(t-\tau)} dN(\tau)$.

The intensity function recovery consistency by minimizing least-square population loss is proved in Theorem B.2 under a memory constraint. We parameterize the function by a neural network (NN) based structure characterized as in (1)-(2). We define the prototypical network architecture below.

Definition B.1. *Define the function class NN-ODE$(d_{\text{out}}, L_h, p_h, L_d, p_d)$ as*

$$\begin{aligned} &\textit{NN-ODE}(d_{\text{out}}, L_h, p_h, L_d, p_d) \coloneqq \{F : \mathbb{R} \mapsto \mathbb{R}^{d_{\text{out}}} | F(t) = g(h(t)), h'(t) = f(h(t), x(t)), \\ &\quad g \textit{ is NN with } L_d \textit{ layers and max-width } p_d, h \textit{ is NN with } L_h \textit{ layers and max width } p_h.\} \end{aligned} \tag{A1}$$

Theorem B.2. *Assume there exist d buffer time steps with samples $x_{-d}, \ldots, x_{-1}$ prior to the Hawkes count data $\{x_0, \ldots, x_M\}$ and each time step has duration $\Delta t = T/M$. We further assume NN-ODE$(d_{\text{out}}, L_h, p_h, L_d, p_d)$ is rich enough to model the true intensity function. Then the minimizer F^* to the population loss function*

$$\Psi(F) \coloneqq \sum_{m=1}^{M} \mathbb{E}\big[(x_m - F(m\Delta t)\Delta t)^2 \big| x_{m-d} \ldots x_{m-1}\big],$$

optimized within the neural network class $F \in$ NN-ODE(D, L_h, p_h, L_d, p_d), satisfies $F^*(m\Delta t) = \tilde{\lambda}(m) := \frac{1}{\Delta t} \int_{(m-1)\Delta t}^{m\Delta t} \lambda^*(t) dt$, which is the discretized intensity.

Remark B.3. We have the recovered intensity function $F^*(m\Delta t) = \tilde{\lambda}(m)$ and is extendable to the entire time horizon as $F^*(t) = F^*(m\Delta t)\mathbf{1}\{(m-1)\Delta t < t \le m\Delta t\}$ for any $t \in [0, T]$. In Appendix C.1 it is shown that under the asymptotic scenario when $M \to \infty$, $\int_0^T |F^*(t) - \lambda^*(t)| dt \to 0$.

B.2 Approximation Analysis of RNN-ODE-Adap

B.2.1 Approximation of the Continuous-Time Model

We will use neural network functions f_θ and g_ϕ to approximate the functions f and g, respectively, see Lemma B.4. Given $x(t)$ on $[0, T]$, let $h_{\text{NN}}(t)$ be the solution to the hidden-process ODE $h'_{\text{NN}}(t) = f_\theta(h_{\text{NN}}(t), x(t))$ from $h_{\text{NN}}(0) = h_0$. This leads to the output process $y_{\text{NN}}(t)$ defined by

$$h'_{\text{NN}}(t) = f_\theta(h_{\text{NN}}(t), x(t)), \quad y_{\text{NN}}(t) = g_\phi(h_{\text{NN}}(t)), \quad h_{\text{NN}}(0) = h_0, \quad t \in [0, T]. \tag{A2}$$

The approximation of $y_{\text{NN}}(t)$ to $y(t)$ will be based on the approximation of f_θ and g_ϕ, which calls for the regularity condition of the system (4).

The next lemma directly follows by applying [32] to the case where f and g have 1st-order regularity (Lipschitz continuity). The proof is given in appendix C.2.

Lemma B.4. *For any $\epsilon_f, \epsilon_g > 0$, there exist neural networks f_θ, g_ϕ such that*

$$\max_{\eta \in [-1.1, 1.1]^{d_h}, x \in [-1,1]^D} \|f(\eta, x) - f_\theta(\eta, x)\|_2 < \epsilon_f, \max_{\eta \in [-1.1, 1.1]^{d_h}} \|g(\eta) - g_\phi(\eta)\|_2 < \epsilon_g, \tag{A3}$$

and

- *f_θ has $O(\ln \frac{C_f}{\epsilon_f} + \ln d_h + 1)$ layers and $O((C_f/\epsilon_f)^{d_h+D}(\ln \frac{C_f}{\epsilon_f} + \ln d_h + 1))$ trainable parameters.*
- *g_ϕ has $O(\ln \frac{C_g}{\epsilon_g} + \ln D' + 1)$ layers and $O((C_g/\epsilon_g)^{d_h}(\ln \frac{C_g}{\epsilon_g} + \ln D' + 1))$ trainable parameters.*

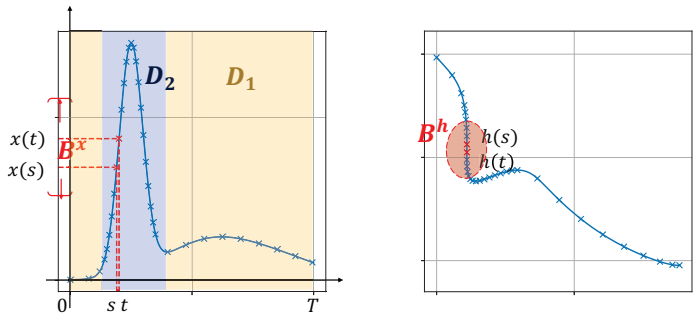

Figure A1: Demonstration of the domains B^x and B^h defined as in (A4) for the time interval $[s,t]$ (here $d_h = 2, D = 1$). The domains $\mathcal{D}_1$ and $\mathcal{D}_2$ that correspond to slowly and fast varying regions are colored in orange and blue respectively.

The constants in big-O may depend on D, D', and d_h. Here $C_f := \max\{L^{f,h}, L^{f,x}, M_f\}$, where $M_f = \sup_{(\eta,x)\in[-1.1,1.1]^{d_h}\times[-1,1]^D} \|f(\eta,x)\|$ and $L^{f,h}, L^{f,x}$ are denote the Lipschitz constant of f on $[-1.1,1.1]^{d_h}\times[-1,1]^D$ (see formal definitions in (A9) *in the proof of Lemma B.4 in Appendix C.2). $C_g := \max\{L_g, M_g\}$, and $M_g = \sup_{\eta\in[-1.1,1.1]^{d_h}} \|g(\eta)\|$.*

For the spike-like data, the majority of the regions are slow-varying, with the spikes occupying only a minor part of the whole interval $[0,T]$. Thus, the whole interval $[0,T]$ may be partitioned into two disjoint sets $\mathcal{D}_1$ and $\mathcal{D}_2$, each of which consisting of unions of disjoint intervals in $[0,T]$. To characterize this partition more precisely, we define the constants related to an interval in $[0,T]$ as follows:

- For an interval $[s,t] \subset [0,T]$, we define the domains B^h, B^x as

$$B^h := (h([s,t]) + B_r^{d_h}) \subset [-1.1,1.1]^{d_h}, \quad B^x := (x([s,t]) + B_r^D) \cap [-1,1]^D, \tag{A4}$$

  with $r = 0.1$, and $B_r^{d_h}, B_r^D$ represent balls with radius r in $\mathbb{R}^{d_h}, \mathbb{R}^D$ respectively (see Figure A1 for illustration). Here, $h([s,t]) + B_r^{d_h}$ means the Minkowski addition, namely $\{h_1 + h_2, h_1 \in h([s,t]), h_2 \in B_r^{d_h}\}$, and $x([s,t]) + B_r^D$ is defined in the same way. Then, we denote

$$\begin{aligned} L_{[s,t]}^{f,h} &:= \sup_{x\in B^x} \sup_{\eta_1,\eta_2\in B^h} \frac{\|f(\eta_1,x) - f(\eta_2,x)\|}{\|\eta_1 - \eta_2\|}, \\ L_{[s,t]}^{f,x} &:= \sup_{h\in B^h} \sup_{x_1,x_2\in B^x} \frac{\|f(\eta,x_1) - f(\eta,x_2)\|}{\|x_1 - x_2\|}, \end{aligned} \tag{A5}$$

  as the local Lipschitz constants of f within the domain $B^h \times B^x$, and

$$M_{[s,t]}^f := \sup_{(\eta,x)\in B^h\times B^x} \|f(\eta,x)\|_2. \tag{A6}$$

Suppose that any time grid $[s_1,t_1]$ in $\mathcal{D}_1$ corresponds to a local Lipschitz constant $L_{[s_1,t_1]}^{f,h} \le L_{\text{low}}$. On contrast, if $[s_2,t_2]$ belongs to $\mathcal{D}_2$, the local Lipschitz constant $L_{\text{low}} < L_{[s_2,t_2]}^{f,h} \le L_{\text{high}} (\le L^{f,h})$. Here, $\mathcal{D}_1$ is comprised of regions with slow variations, while $\mathcal{D}_2$ encompasses regions with sharp changes, as demonstrated in Figure A1. It may often be the case that $|D_1|$ is greater than $|D_2|$. Then, we define

$$L^{(\text{avg})} := \frac{1}{T}(L_{\text{low}}|D_1| + L_{\text{high}}|D_2|). \tag{A7}$$

Following Lemma B.4 and the partition described above, Theorem B.5 below provides the approximation results for the continuous-time process $y(t)$ using (A2).

Theorem B.5. *Under Assumption B.8 and for $L^{(\text{avg})}$ defined as in* (A7)*, suppose $\epsilon_f, \epsilon_g > 0$ and ϵ_f satisfies $Te^{L^{(\text{avg})}T}\epsilon_f < 0.1$, and let f_θ, g_ϕ be the neural networks satisfying* (A3) *(the model complexity is bounded as in Lemma B.4), then*

$$\max_{t\in[0,T]} \|y(t) - y_{NN}(t)\| < \epsilon_g + L_g T e^{L^{(\text{avg})}T}\epsilon_f. \tag{A8}$$

Remark B.6 (Interpretation of $L^{(\text{avg})}$ and local Lipschitz constants). (A8) can provide an improved bound because when the data have sharp changes, L_{high} (as the ∞-norm of the Lipschitz constant) can be large while $L^{(\text{avg})} = (L_{\text{low}}|D_1| + L_{\text{high}}|D_2|)/T$ (as certain L^1-norm of the Lipsthictz constant) may stay at a smaller value. The partition $\mathcal{D}_1 \cup \mathcal{D}_2$ reflects how adaptively choosing grids may help improve the theoretical results, and this will be further explored in the next subsection.

Remark B.7 (Arbitrary desired accuracy in (A8)). For any $\varepsilon > 0$, we can choose $\epsilon_f < \frac{1}{T\exp(L^{(\text{avg})}T)} \min\{0.1, \frac{\varepsilon}{2L_g}\}, \epsilon_g < \frac{\varepsilon}{2}$, then the right-hand side of (A8) is bounded by ε.

B.2.2 More Details about Approximation under Time Discretization in Section 2.2

Assumption B.8. *(A1) The observed process $x : [0, T] \to [-1, 1]^D$ and is Lipschitz continuous over t; the hidden process $h : [0, T] \to [-1, 1]^{d_h}$.*

(A2) $f : [-1.1, 1.1]^{d_h} \times [-1, 1]^D \to \mathbb{R}^{d_h}, (\eta, x) \mapsto f(\eta, x)$, and is Lipschitz continuous with respect to both η and x.

(A3) $g : [-1.1, 1.1]^{d_h} \to [-1, 1]^{D'}, \eta \mapsto g(\eta)$ is Lipschitz continuous.

We let L_g denote the global Lipschitz constant of g on $[-1.1, 1.1]^{d_h}$. For f, both global and local Lipschitz constants on the domain $[-1.1, 1.1]^{d_h} \times [-1, 1]^D$ are used.

Given the time grids $\{t_i\}_{i=1}^N$, we define the following local constants used in Theorem 2.1:

- By (A2), for each i, let $L_i^{f,h}, L_i^{f,x}$ and M_i^f be defined as in (A5) and (A6) respectively, where we take the interval as $[t_{i-1}, t_i]$.
- By (A1), for each i, let L_i^x be the Lipschitz constant of $x(t)$ on $t \in [t_{i-1}, t_i]$. $i = 1, \ldots, N+1$.

We define

$$\mu_i \coloneqq L_i^{f,h} M_i^f + L_i^{f,x} L_i^x,$$

which characterizes the Lipschitz continuity of the system on on $t \in [t_{i-1}, t_i]$.

The condition $T\exp(\sum_{i=1}^N L_i^{f,h}\Delta t_i)\,(\epsilon_f + \max_j\{\mu_j \Delta t_j\}) < 0.1$ in Theorem 2.1 is imposed to guarantee that the numerically integrated hidden states $\{\hat{h}_{\text{NN}}(t_i)\} \subset [-1.1, 1.1]^{d_h}$, so that the approximation results in Lemma B.4 are applicable.

Remark B.9 (Arbitrary desired accuracy in (6)). For any $\varepsilon > 0$, suppose the time grids satisfy that $\max_j\{\mu_j\Delta t_j\} < \frac{1}{T\exp(\sum_{i=1}^N L_i^{f,h}\Delta t_i)} \min\{0.05, \frac{\varepsilon}{3L_g}\}$, then we can choose $\epsilon_f < \frac{1}{T\exp(\sum_{i=1}^N L_i^{f,h}\Delta t_i)} \min\{0.05, \frac{\varepsilon}{3L_g}\}, \epsilon_g < \frac{\varepsilon}{3}$, to make the right-hand side of (6) bounded by ε.

Remark B.10 (Extension to higher-order integration schemes). The numerical integration scheme (5) can be extended to the multi-step explicit methods of higher orders (e.g. Runge-Kutta methods), given that the time grid selection appropriately fulfills the requirements of the integration scheme. For example, we may choose $t_{i+1} - t_i = t_i - t_{i-1}$ for adjacent sub-intervals $[t_{i-1}, t_i], [t_i, t_{i+1}]$ to apply the commonly used RK4 method .

C Proofs

C.1 Proofs in Section B.1

Proof of Proposition B.2. We consider the case with discretized and finite time grids. We assume there exists d buffer time steps with samples $x_{-d}, \ldots, x_{-1}$. The samples used for estimation are $x_0, \ldots, x_M$, and each time step is with duration Δt. Thus the whole time duration is $T = M\Delta t$. The random process we observe on discrete time horizon $\{m : 1 \le m \le M\}$ is as follows. At time m we observe integer variable $x_m \in \{0, 1, 2, \ldots\}$. Here x_m means the number of event happening within $((m-1)\Delta t, m\Delta t]$ and $x_m = 0$ means no event happening. Note that for the Hawkes process, which is essentially an inhomogeneous Poisson process, the variable x_m is just a Poisson random variable with the intensity parameter depending on the historical observations. We denote the average intensity function within the time interval $((m-1)\Delta t, n\Delta t]$ as:

$$\tilde{\lambda}(m) = \frac{1}{\Delta t}\int_{(m-1)\Delta t}^{m\Delta t} \lambda^*(t)dt,$$

where $\lambda^*(t) = \mu + \alpha \int_0^t \phi(t-s)dN(s)$ is the true (continuous-time) intensity function.

By the properties of the Poisson distribution, we have $\mathbb{E}[x_m|\mathcal{F}_{m-1}] = \tilde{\lambda}(m)\Delta t$ and $\mathrm{Var}[x_m|\mathcal{F}_{m-1}] = \tilde{\lambda}(m)\Delta t$. Our goal is to recover the intensity function $\lambda(\cdot)$ using the given observations. We consider the population loss function:

$$\begin{aligned}\Psi(\theta_h, \theta_d) &= \sum_{m=1}^{M} \mathbb{E}[(x_m - F(m; \theta_h, \theta_d)\Delta t)^2 | x_{m-d}, \ldots, \omega_{m-1}] \\ &= \sum_{m=1}^{M} \left\{ \mathbb{E}[x_m^2 | x_{m-d}^{m-1}] - 2\mathbb{E}[x_m \cdot F(m; \theta_h, \theta_d)\Delta t | x_{m-d}^{m-1}] + \mathbb{E}[F^2(m; \theta_h, \theta_d)(\Delta t)^2 | x_{m-d}^{m-1}] \right\}. \\ &\propto \sum_{m=1}^{M} (\tilde{\lambda}(m) - F(m; \theta_h, \theta_d))^2\end{aligned}$$

Thus the optimizer will equal to $\tilde{\lambda}(n)$ as long as the function class *RNN-ODE*$(d_{\text{out}}, L_h, p_h, L_d, p_d)$ is rich enough to model the structure of the true intensity function. □

Proof of the claim in Remark B.3. Note that when there is no event happening within the time interval $((m-1)\Delta t, m\Delta t]$ or when there is one event happening at $m\Delta t$, we have $|\tilde{\lambda}(m) - \lambda^*(t)| \leq C\alpha\Delta t$ where C is a constant related to the Lipschitz constant of the influence kernel $\phi(\cdot)$. And when there is one event happening in $((m-1)\Delta t, m\Delta t)$, we have $|\tilde{\lambda}(n) - \lambda^*(t)| \leq \alpha$. By the concentration of Poisson distribution, there exists positive constant M' such that there are at most M' events happening within $[0, T]$ with high probability, and there is at most one event in each sub-interval $[(m-1)\Delta t, m\Delta t]$ for M sufficiently large, we have that $\int_0^T |\tilde{\lambda}(t) - \lambda^*(t)|dt \leq C\alpha T\Delta t + M'\alpha\Delta t \to 0$ as $M \to \infty$. □

C.2 Proofs in Section 2.2

Remark C.1 (Expressiveness of the model). The hidden state $h(t) \in \mathbb{R}^{d_h}$ in (4) encodes the historical data, enabling $x(t)$ to be time-inhomogeneous. This raises the question regarding the expressiveness of Eq. (4) in representing a general dynamical system described by $x'(t) = F(x(t), t)$. There exist works that explored the expressiveness of the system $h'(t) = f_\theta(h(t), x(t)), y(t) = g_\phi(h(t))$, where f_θ, g_ϕ are neural networks and f_θ possesses a RNN structure [8, 4, 19, 20]. Among these works, [8, 4, 19] assumed that $x(t)$ was generated from the underlying dynamics (4), and thus the approximation problem was reduced to estimating f and g using neural networks f_θ and g_ϕ. On the other hand, [20] took into account a broader range of input-output relationships. Specifically, it studied the expressiveness of the linear RNN structure in representing functionals H_t that determined the output at time t according to $H_t(\{x(\tau), \tau \in \mathcal{T}\})$, where $\mathcal{T}$ is an ordered index set (e.g., $\mathcal{T} = [0, T]$). [20] mainly focused on the case when $\{H_t(\{x(\tau), \tau \in \mathcal{T}\})\}$ is linear and time-homogeneous.

Our approximation analysis bears more resemblance to the first category of studies and examines the approximation error for the discretely observed data.

Remark C.2 (Time-homogeneous dynamical systems). For a time-homogeneous dynamical system $x'(t) = F(x(t))$, it can be represented as Eq. (4) by setting $d_h = D$, $f(h, x) = F(h)$, and $g(h) = h$. Theorems B.5 and 2.1 indicate that neural networks f_θ, g_ϕ can be configured such that the observed data is approximated to any pre-specified accuracy. Prior studies [8, 4, 19, 20] proved that the system $x'(t) = F(x(t))$ could also be approximated using a continuous-time RNN, although without upper bounding the network size.

Proof of Lemma B.4 Following the notations in [32], we consider Sobolev space $\mathcal{W}^{n,\infty}([-1,1]^d)$, with $n = 1, 2, \ldots$, defined as the space of functions on $[-1,1]^d$ lying in L^∞ with their weak derivatives up to order n. From the proof of [32, Theorem 1], for any $f : [-1,1]^d \to \mathbb{R}$ such that $f \in \mathcal{W}^{n,\infty}([-1,1]^d)$ and $\epsilon > 0$, there exists a neural network $\widetilde{f}$ such that $\max_{x\in[-1,1]^d} |f(x) - \widetilde{f}(x)| < \epsilon$, and $\widetilde{f}$ has $O(\ln(d+1)(\ln(\frac{\alpha_f}{\epsilon})+1))$ layers and $O(2^{d(d+1)} d^{d+2} \ln(d+1)^2 (\frac{2\beta_f}{\epsilon})^{\frac{d}{n}} (\ln(\frac{\alpha_f}{\epsilon})+1))$ trainable parameters, where $\alpha_f = \|f\|_{\mathcal{W}^{n,\infty}([-1,1]^d)} \coloneqq \max_{\mathbf{n}:|\mathbf{n}|\leq n} \text{ess sup}_{x\in[-1,1]^d} |D^{\mathbf{n}} f(x)|, \beta_f \coloneqq \max_{\mathbf{n}:|\mathbf{n}|=1} \text{ess sup}_{x\in[-1,1]^d} |D^{\mathbf{n}} f(x)|$.

In our case, we take $n = 1$. For $f : [-1.1, 1.1]^{d_h} \times [-1, 1]^D \to \mathbb{R}^{d_h}$ that is Lipschitz in both η and x, we define $L^{f,h}, L^{f,x}$ as follows:

$$\begin{aligned} L^{f,h} &\coloneqq \sup_{x\in[-1,1]^D} \sup_{\eta_1,\eta_2\in[-1.1,1.1]^{d_h}} \frac{\|f(\eta_1, x) - f(\eta_2, x)\|}{\|\eta_1 - \eta_2\|}, \\ L^{f,x} &\coloneqq \sup_{h\in[-1.1,1.1]^{d_h}} \sup_{x_1,x_2\in[-1,1]^D} \frac{\|f(\eta, x_1) - f(\eta, x_2)\|}{\|x_1 - x_2\|}. \end{aligned} \tag{A9}$$

For $\tilde{f} = (\tilde{f}_1, \ldots, \tilde{f}_{d_h}) : [-1, 1]^{d_h} \times [-1, 1]^D \to \mathbb{R}^{d_h}$ defined as $\tilde{f}(\tilde{\eta}, x) \coloneqq f(1.1\tilde{\eta}, x)$, we have that $\alpha_{\tilde{f}_i} \le 1.1C_f, \beta_{\tilde{f}_i} \le 1.1C_f$, $i = 1, \ldots, d_h$. Therefore, there exist d_h subnetworks, denoted as $\hat{f}_1, \ldots, \hat{f}_{d_h}$, such that

$$\max_{\tilde{\eta}\in[-1,1]^{d_h}, x\in[-1,1]^D} |\tilde{f}_i(\tilde{\eta}, x) - \hat{f}_i(\tilde{\eta}, x)| < \frac{\epsilon_f}{\sqrt{d_h}},$$

and each subnetwork has $O(\ln(\frac{1.1C_f}{\epsilon_f}) + \ln d_h + 1)$ layers and $O((\frac{2.2C_f}{\epsilon_f})^{d_h+D}(\ln(\frac{1.1C_f}{\epsilon_f}) + \ln d_h + 1))$ weights, where the constants of big-O notations depend on d_h and D.

Thus, we can construct $\tilde{f}_\theta$ as a network consisting of d_h parallel sub-networks that implement each of $\hat{f}_i$. Then, for $f_\theta(\eta, x) \coloneqq \tilde{f}_\theta(\frac{1}{1.1}\eta, x)$,

$$\begin{aligned} \max_{\eta\in[-1.1,1.1]^{d_h}, x\in[-1,1]^D} \|f(\eta, x) - f_\theta(\eta, x)\|_2 &= \max_{\tilde{\eta}\in[-1,1]^{d_h}, x\in[-1,1]^D} \|\tilde{f}(\tilde{\eta}, x) - \tilde{f}_\theta(\tilde{\eta}, x)\|_2 \\ &\le \sqrt{d_h} \max_{\tilde{\eta}\in[-1,1]^{d_h}, x\in[-1,1]^D} \|\tilde{f}(\tilde{\eta}, x) - \tilde{f}_\theta(\tilde{\eta}, x)\|_\infty < \epsilon_f. \end{aligned}$$

f_θ has $O(\ln(\frac{1.1C_f}{\epsilon_f}) + \ln d_h + 1)$ layers and $O((\frac{2.2C_f}{\epsilon})^{d_h+D}(\ln(\frac{1.1C_f}{\epsilon_f}) + \ln d_h + 1))$ weights, where the constants of big-O notations depend on d_h and D. Specifically,

$$\begin{aligned} &\#(\text{layers of } f_\theta) \le C \ln(d_h + D + 1)(\ln(\frac{1.1C_f}{\epsilon_f}) + \ln d_h + 1), \\ &\#(\text{weights of } f_\theta) \le C2^{(d_h+D)(d_h+D+1)}(d_h + D)^{d_h+D+2} d_h^{\frac{d_h+D+2}{2}} \ln(d_h + D + 1)^2 \\ &\qquad \cdot (\frac{2.2C_f}{\epsilon})^{d_h+D}(\ln(\frac{1.1C_f}{\epsilon_f}) + \ln d_h + 1), \end{aligned}$$

for some absolute constant $C > 0$. g_ϕ can be constructed similarly. We define $\tilde{g} = (\tilde{g}_1, \ldots, \tilde{g}_{D'}) : [-1, 1]^{d_h} \to \mathbb{R}^{D'}$ as $\tilde{g}(\tilde{\eta}) \coloneqq g(1.1\tilde{\eta})$. Then, there exist D' subnetworks, denotes as $\hat{g}_1, \ldots, \hat{g}_{D'}$, such that

$$\max_{\tilde{\eta}\in[-1,1]^{d_h}} \|\tilde{g}_i(\tilde{\eta}) - \hat{g}_i(\tilde{\eta})\|_2 < \frac{\epsilon_g}{\sqrt{D'}}, \quad i = 1, \ldots, D',$$

and each subnetwork has $O(\ln(\frac{1.1C_g}{\epsilon_g}) + \ln D' + 1)$ layers and $O((\frac{2.2C_g}{\epsilon_g})^{d_h}(\ln(\frac{1.1C_g}{\epsilon_g}) + \ln D' + 1))$ weights, where the constants of big-O notations depend on d_h and D'. We construct $\tilde{g}_\phi$ as a network consisting of D' parallel subnetworks that implements $\{\hat{g}_i\}$. Then, for $g_\phi(\eta) \coloneqq \tilde{g}_\phi(\frac{1}{1.1}\eta)$,

$$\max_{\eta\in[-1.1,1.1]^{d_h}} \|g(\eta) - g_\phi(\eta)\|_2 < \epsilon_g,$$

and

$$\begin{aligned} &\#(\text{layers of } g_\phi) \le C \ln(d_h + 1)(\ln(\frac{1.1C_g}{\epsilon_g}) + \ln D' + 1), \\ &\#(\text{weights of } g_\phi) \le C2^{d_h(d_h+1)} d_h^{d_h+2} D'^{\frac{d_h+2}{2}} \ln(d_h + 1)^2 (\frac{2.2C_g}{\epsilon_g})^{d_h} \cdot (\ln(\frac{1.1C_g}{\epsilon_g}) + \ln D' + 1). \end{aligned}$$

This proves the claim.

Proof of Theorem B.5.

Proof of Theorem B.5. We denote $u(t) \coloneqq \|h(t) - h_{\text{NN}}(t)\|$, and $t_0 = \inf_{t\in[0,T]}\{u(t) \geq 0.1\}$. Since $u(0) = 0$ and $u(t)$ is continuous, we know that $t_0 > 0$. In the following, we show that $t_0 = T$ by contradiction. Otherwise, suppose that $t_0 < T$. Then for $t \in [0, t_0]$, $u(t) = \|h(t) - h_{\text{NN}}(t)\| \leq 0.1$, which implies that $h_{\text{NN}}(t) \in [-1.1, 1.1]^{d_h}$.

Then, by (A3), for $t \in [0, t_0]$,

$$\begin{aligned} u(t) &= \left\| \int_0^t \left(f(h(s), x(s)) - f_\theta(h_{\text{NN}}(s), x(s))\right) \mathrm{d}s \right\| \\ &\leq \int_0^t \|f(h(s), x(s)) - f_\theta(h_{\text{NN}}(s), x(s))\| \, \mathrm{d}s \\ &\leq \int_0^t \left(\epsilon_f + L(s)\|h(s) - h_{\text{NN}}(s)\|\right) \mathrm{d}s, \end{aligned}$$

where

$$L(s) = L_i^{f,h}, \quad \text{if } s \in [t_{i-1}, t_i], i = 1, \ldots, n+1,$$

and $\{t_i\}_{i=1}^n$ the time grid corresponding to the partition $\mathcal{D}_1 \cup \mathcal{D}_2$ such that $\frac{1}{T}\sum_{i=1}^{n+1} L_i^{f,h}(t_i - t_{i-1}) \leq \frac{1}{T}(L_{\text{low}}|D_1| + L_{\text{high}}|D_2|) = L^{(\text{avg})}$, and $L_i^{f,h}$ is defined as in (A5) with taking the interval $[s, t] = [t_{i-1}, t_i]$. Therefore,

$$u(t) \leq \epsilon_f t + \int_0^t L(s)u(s)\mathrm{d}s, \quad t \in [0, t_0].$$

By the Grönwall's inequality,

$$u(t) \leq \epsilon_f t \exp\left(\int_0^t L(s)\mathrm{d}s\right) \leq \epsilon_f T \exp\left(\sum_{i=1}^{n+1} L_i^{f,h}(t_i - t_{i-1})\right) \leq \epsilon_f T \exp(L^{(\text{avg})}T) < 0.1, \quad t \in [0, t_0].$$

Specifically,

$$u(t_0) \leq \epsilon_f T \exp(L^{(\text{avg})}T) < 0.1.$$

Since $u(t)$ is continuous, there exists a sufficiently small $\delta > 0$, such that $u(t) < 0.1$ for $t \in [t_0, t_0+\delta]$. This is a contradiction to the definition of t_0. Thus, we conclude that $t_0 = T$, and therefore $h_{\text{NN}}(t) \in [-1.1, 1.1]^{d_h}$, $\forall t \in [0, T]$. By the similar analysis above, we have that

$$u(t) \leq \epsilon_f T \exp(L^{(\text{avg})}T), \quad t \in [0, T].$$

Thus, for $t \in [0, T]$,

$$\begin{aligned} \|y(t) - y_{\text{NN}}(t)\| &\leq \|g(h(t)) - g(h_{\text{NN}}(t))\| + \|g(h_{\text{NN}}(t)) - g_\phi(h_{\text{NN}}(t))\| \\ &\leq L_g T \exp(L^{(\text{avg})}T)\epsilon_f + \epsilon_g, \end{aligned}$$

which proves the claim. □

Proof of Theorem 2.1.

Proof of Theorem 2.1. Denote $\varepsilon_i = h(t_i) - \hat{h}_{\text{NN}}(t_i)$, then $\varepsilon_0 = 0$. In the following, we apply the induction argument, iteratively showing that

$$\|\varepsilon_i\| \leq e_h < 0.1, \quad \hat{h}_{\text{NN}}(t_i) \in [-1.1, 1.1]^{d_h}, \quad i = 0, \ldots, N, \tag{A10}$$

where

$$e_h \coloneqq T \exp\left(\sum_{i=1}^{N} L_i^{f,h}\Delta t_i\right)\left(\epsilon_f + \max_j\{\mu_j \Delta t_j\}\right).$$

For $i = 0$, (A10) naturally holds since $h(0) = \hat{h}_{\text{NN}}(0)$. If (A10) holds For $i \leq k$, then for $i = k+1$, by $\|\varepsilon_k\| < 0.1$, $\hat{h}_{\text{NN}}(t_k) \in [-1.1, 1.1]^{d_h}$. From the definition of ε_{k+1},

$$\begin{aligned}\varepsilon_{k+1} &= \left\| \left(h(t_k) + \int_{t_k}^{t_{k+1}} f(h(s), x(s)) \mathrm{d}s \right) - \left(\hat{h}_{\text{NN}}(t_k) + \Delta t_{k+1} f_\theta(\hat{h}_{\text{NN}}(t_k), x(t_k)) \right) \right\| \\ &\leq \varepsilon_k + \int_{t_k}^{t_{k+1}} \left\| f(h(s), x(s)) - f_\theta(\hat{h}_{\text{NN}}(t_k), x(t_k)) \right\| \mathrm{d}s.\end{aligned}$$

Next, we upper bound the second term. By the triangle inequality, $\hat{h}_{\text{NN}}(t_k) \in [-1.1, 1.1]^{d_h}$ and (A3), for $s \in [t_k, t_{k+1}]$,

$$\begin{aligned}&\left\| f(h(s), x(s)) - f_\theta(\hat{h}_{\text{NN}}(t_k), x(t_k)) \right\| \\ &\leq \left\| f(h(s), x(s)) - f(\hat{h}_{\text{NN}}(t_k), x(t_k) \right\| + \left\| f(\hat{h}_{\text{NN}}(t_k), x(t_k) - f_\theta(\hat{h}_{\text{NN}}(t_k), x(t_k)) \right\| \\ &\leq \|f(h(s), x(s)) - f(h(t_k), x(s))\| + \|f(h(t_k), x(s)) - f(h(t_k), x(t_k))\| \\ &\quad + \left\| f(h(t_k), x(t_k)) - f(\hat{h}_{\text{NN}}(t_k), x(t_k)) \right\| + \epsilon_f \\ &\leq (L_{k+1}^{f,h} M_{k+1}^f + L_{k+1}^{f,x} L_{k+1}^x) \Delta t_{k+1} + L_{k+1}^{f,h} \Delta t_{k+1} \varepsilon_k + \epsilon_f,\end{aligned}$$

where the first component is due to $\|h(s) - h(t_k)\| = \| \int_{t_k}^s f(h(u), x(u)) du\| \leq M_{k+1}^f \Delta t_{k+1}$ and the second term results from $|x(s) - x(t_k)| \leq L_{k+1}^x \Delta t_{k+1}$.

This implies that

$$\varepsilon_{k+1} \leq (1 + L_{k+1}^{f,h} \Delta t_{k+1}) \varepsilon_k + \gamma_{k+1}, \tag{A11}$$

where

$$\gamma_{k+1} \coloneqq \epsilon_f \Delta t_{k+1} + (L_{k+1}^{f,h} M_{k+1}^f + L_{k+1}^{f,x} L_{k+1}^x) \Delta t_{k+1}^2 = \epsilon_f \Delta t_{k+1} + \mu_{k+1} \Delta t_{k+1}^2.$$

From (A11), we obtain that

$$\varepsilon_{k+1} \leq \sum_{j=1}^{i+1} \left(\gamma_j \cdot \prod_{l=j+1}^{k+1} (1 + L_l^{f,h} \Delta t_l) \right).$$

Since $1 + x \leq \exp(x)$,

$$\prod_{l=j+1}^{k+1} (1 + L_l^{f,h} \Delta t_l) \leq \exp(\sum_{l=j+1}^{k+1} L_l^{f,h} \Delta t_l) \leq \exp(\sum_{l=1}^{N} L_l^{f,h} \Delta t_l).$$

Hence, we have

$$\begin{aligned}\varepsilon_{k+1} &\leq \exp(\sum_{i=1}^{N} L_i^{f,h} \Delta t_i) \sum_{j=1}^{N} \gamma_j = \exp(L^{f,h} T) \left(\sum_{j=1}^{N} \epsilon_f \Delta t_j + \mu_j \Delta t_j^2 \right) \\ &\leq T \exp(\sum_{i=1}^{N} L_i^{f,h} \Delta t_i) \left(\epsilon_f + \max_j \{\mu_j \Delta t_j\} \right) = e_h.\end{aligned}$$

Therefore, (A10) holds for $i = k+1$. By the induction argument, (A10) is true for $i = 1, \ldots, N$.

Finally, for $i = 1, \ldots, N$, by triangle inequality and the fact that $\hat{h}_{\text{NN}}(t_i) \in [-1.1, 1.1]^{d_h}$, applying (A3) results in

$$\begin{aligned}\|y(t_i) - \hat{y}_{\text{NN}}(t_i)\| &= \|g(h(t_i)) - g_\phi(\hat{h}_{\text{NN}}(t_i))\| \\ &\leq \|g(h(t_i)) - g(\hat{h}_{\text{NN}}(t_i))\| + \|g(\hat{h}_{\text{NN}}(t_i)) - g_\phi(\hat{h}_{\text{NN}}(t_i))\| \\ &\leq \epsilon_g + L_g \varepsilon_{k+1} < \epsilon_g + L_g e_h.\end{aligned} \tag{A12}$$

This proves the claims in Theorem 2.1. □

D Experimental Details

We provide More details of the experiment settings in this section, with the boxplots of the error plots and additional results provided in Appendix D.7.

D.1 Training Objective

Given the time horizon $[0, T]$, suppose that we have a collection of training windows with the k-th one denoted as $\boldsymbol{x}^{(\mathrm{Tr},k)} = \{x^{(\mathrm{Tr},k)}(t_1^{(\mathrm{Tr},k)}), \ldots, x^{(\mathrm{Tr},k)}(t_N^{(\mathrm{Tr},k)})\}$. We train the model (1)-(2) parametrized by neural networks with trainable parameters Θ using the *mean-squared regression loss* function

$$\mathcal{L}(\Theta; \{\boldsymbol{x}^{(\mathrm{Tr},k)}\}_{k=1}^{K^{(\mathrm{Tr})}}) = \sum_{k=1}^{K^{(\mathrm{Tr})}} \sum_{i=1}^{N} \|\hat{x}^{(\mathrm{Tr},k)}(t_i^{(\mathrm{Tr},k)}) - x^{(\mathrm{Tr},k)}(t_i^{(\mathrm{Tr},k)})\|^2 |t_i^{(\mathrm{Tr},k)} - t_{i-1}^{(\mathrm{Tr},k)}|, \tag{A13}$$

where $\hat{x}^{(k)}(t)$ is the output of the neural ODE model under parameter Θ conditioned on all past observation.

D.2 Computational Complexity

The computational complexity of applying Algorithm 1 in the preprocessing stage to $K^{(\mathrm{Tr})}$ training windows is $O(K^{(\mathrm{Tr})}ND)$, where D is the data dimension. For the neural ODE model described as in (1)-(2), when f possesses the same network structure as a vanilla RNN with d_h hidden units and g is a one-layer fully connected network, the complexity in the training process is $O(n_e K^{(\mathrm{Tr})} \bar{N}_a d_h (d_h + D))$, where n_e and $\bar{N}_a$ represent the number of training epochs and the average length of the adaptive windows, respectively.

Since the computational cost in the training process usually dominates that in the preprocessing step (which happens as long as $n_e d_h \geq 2^L$), the overall complexity of the ***RNN-ODE-Adap*** model is $O(n_e K^{(\mathrm{Tr})} \bar{N}_a d_h (d_h + D))$. This is of the same order as the complexity of training a vanilla RNN with d_h hidden units (we refer to Appendix D.4 for the architecture) in n_e epochs, using $K^{(\mathrm{Tr})}$ training windows with the same length $\bar{N}_a$. Therefore, compared with the complexity when training with the original finest N time grids, the complexity associated with the adaptive method will be reduced by a factor of $\bar{N}_a/N$. The smallest achievable complexity will be reduced by a factor of $1/2^L$ when choosing a sufficiently large threshold ϵ.

D.3 Implementation Details

Given the training windows $\{\boldsymbol{x}^{(\mathrm{Tr},k)}\}_{k=1}^{K^{(\mathrm{Tr})}}$, the algorithm 1 is used as a preprocessing step to prepare *each* training window to the irregular sub-window with adaptive time steps. The resulting adaptive training windows are then used to train the neural ODE model (1)-(2) using the mean-squared loss function (A13). During the inference phase, the learned ODE model (1) will be used for fitting and prediction tasks. It can be used for arbitrary and irregular (future) time steps.

Choice of Monitor Functions. The monitor function in Algorithm 1 can be chosen flexibly, not restricted to the maximum variation defined in (3). We may also choose ℓ_p norms for any $p \geq 1$. Since this work mainly uses non-stationary time series with "spike"-like patterns as an example, the monitor function (3) is a natural choice for identifying abrupt "spikes". In general, the monitor function may be designed case-by-case depending on the problem context. For example, when modeling the event-type counting process (such as the simulated Hawkes process), where $x_i \in \mathbb{N}$ is the number of events in the current time interval, we may choose to use the maximum counts $M(x_i, \ldots, x_j) := \max\{x_i, \ldots, x_j\}$. By setting the threshold $\epsilon \in (0, 1)$, such a monitor function will assign the finest time steps to intervals with events ($x_i > 0$) and use rough time steps for regions without events ($x_i = 0$).

Choice of Threshold in Algorithm 1. The choice of the selection threshold ϵ used in Algorithm 1 can be selected from training data via simulation. In detail, note that a larger threshold ϵ would lead to a sparser set of selected time steps (the output of Algorithm 1). Therefore, we primarily determine the threshold ϵ by calibrating the number of remaining time stamps after applying Algorithm 1, allowing

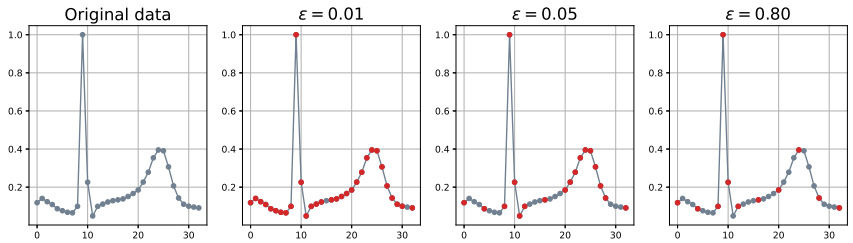

Figure A2: Illustration of adaptively selected time steps with different thresholds (ECG data). The gray dots depict the original data points and the red points illustrate the time steps selected by Algorithm 1 using threshold $\epsilon = 0.01$, 0.05, and 0.8, and $L = 3$.

us to control the desired efficiency. Specifically, we employ a validation data set to calibrate the selection of threshold values, ensuring that the chosen ϵ yields the desired average lengths for the adaptively selected time steps. As illustrated in Figure A2, an example of ECG data demonstrates the influence of the threshold parameter ϵ on the chosen grids. It can be observed that an increasing ϵ leads to a reduction in the number of selected grids. Furthermore, for each value of ϵ, the chosen grids correspond to sub-intervals with greater variation.

Evaluation Metric. To compare the *multi-step prediction* performance of different time series models, we use the mean-squared multi-step ahead prediction error as follows. After obtaining the fitted model, we can use the trained networks (1)-(2) to make predictions on the testing windows $\{\boldsymbol{x}^{(\text{Te},k)}\}_{k=1}^{K^{(\text{Te})}}$, where $\boldsymbol{x}^{(\text{Te},k)} = \{x^{(\text{Te},k)}(t_1^{(\text{Te},k)}), \ldots, x^{(\text{Te},k)}(t_n^{(\text{Te},k)})\}$. Given a historical trajectory $\{x(t_1), \ldots, x(t_n)\}$, we can apply the fitted model to perform multi-step ahead prediction

$$\begin{aligned} \hat{h}(t_{i+1}) &= \hat{h}(t_i) + \begin{cases} \int_{t_i}^{t_{i+1}} f(\hat{h}(s), x(s); \theta_h) ds, & \text{when } i \le n, \\ \int_{t_i}^{t_{i+1}} f(\hat{h}(s), \hat{x}(s); \theta_h) ds, & \text{when } n < i \le n+m, \end{cases} \\ \hat{x}(t_{i+1}) &= g(\hat{h}(t_{i+1}); \theta_d), \end{aligned} \tag{A14}$$

which will be iteratively solved for $i = 1, \ldots, n+m$. The first ODE can be solved by, for instance, the Euler method. When comparing the m-step ahead prediction performance of different methods, we use the averaged ℓ_2 norm of the prediction error of length m; specifically, we take $n = \lfloor N/2 \rfloor, m = N - n$ in the experiments in Section 3 and perform the prediction in Eq. (A14) for each testing window $\boldsymbol{x}^{(\text{Te},k)}$ with the predicted value denoted as $\hat{x}^{(\text{Te},k)}(\cdot)$, then the resulting prediction performance on test data is measured as follows

$$\text{MSE}_{\text{pred}} = \frac{1}{K^{(\text{Te})}} \sum_{k=1}^{K^{(\text{Te})}} \left(\frac{1}{m} \sum_{i=n+1}^{n+m} \left\| x^{(\text{Te},k)}(t_i^{(\text{Te},k)}) - \hat{x}^{(\text{Te},k)}(t_i^{(\text{Te},k)}) \right\|^2 \right)^{1/2}. \tag{A15}$$

We could also use other reasonable metrics that measure the discrepancy between times series data, such as the averaged ℓ_1 norm or the dynamic time warping distance [29].

To compare the *one-step prediction* performance of different methods on the intensity function of the event data generated from Hawkes processes, we employ the error as defined in (A16), which has the similar form to the training loss (A13):

$$\text{MSE}_{\text{fit}} = \frac{1}{K^{(\text{Te})}} \sum_{k=1}^{K^{(\text{Te})}} \sum_{i=1}^{N} \left\| \hat{\lambda}^{(\text{Te},k)}(t_i^{(\text{Te},k)}) - \lambda^{(\text{Te},k)}(t_i^{(\text{Te},k)}) \right\|^2 |t_i^{(\text{Te},k)} - t_{i-1}^{(\text{Te},k)}|, \tag{A16}$$

here the superscript $^{(\text{Te})}$ denotes that the error is evaluated on the testing data. λ and $\hat{\lambda}$ denote the true and fitted intensity functions of the event data, respectively. In this case, $\hat{\lambda}$ is obtained by iteratively solving (A14) with $n = N - 1$ and $m = 0$.

Buffer Steps. To facilitate training and improve the performance of the models, we leverage additional "buffer steps" at the beginning of each window to mitigate the effect of the zero initialization

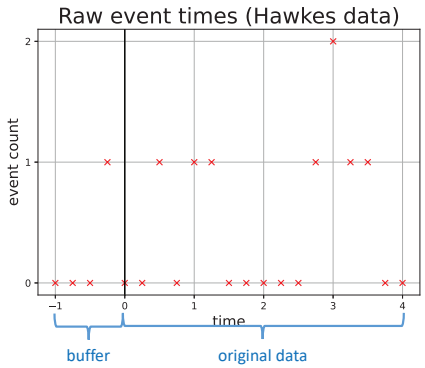

Figure A3: Illustration of buffer steps constructed on a discrete-time event data generated from a Hawkes process.

of the hidden states. Buffer steps refer to the additionally padded time stamps before each window. Specifically, for the k-th training window $\{x^{(\mathrm{Tr},k)}(t_i^{(\mathrm{Tr},k)})\}_{i=1}^N$, adding m buffer steps means that the original time series is augmented to $\{x^{(\mathrm{Tr},k)}(t_i^{(\mathrm{Tr},k)})\}_{i=-m}^N$, where for $i = -m, \ldots, -1$, $t_{i+1}^{(\mathrm{Tr},k)} - t_i^{(\mathrm{Tr},k)} = \Delta t := \min_{i=0,\ldots,N-1}\{t_{i+1}^{(\mathrm{Tr},k)} - t_i^{(\mathrm{Tr},k)}\}$. An illustration of the buffer steps for the discrete point process data is shown in Figure A3. Detailed information on buffer steps, pertaining to the experiments in Section 3, along with additional experiments examining the impact of incorporating buffer steps and selecting the appropriate number of buffer steps using validation data, can be found in Appendix D.6.

D.4 Network Structure

In the experiments, we use the same network structure for RNN, RNN-ODE, and RNN-ODE-Adap, namely the ODE function f follows the vanilla RNN structure

$$f(h, x; \Theta_f) = \tanh(W_f[h, x] + b_f),$$

where $h \in \mathbb{R}^{d_h}, W_f \in \mathbb{R}^{d_h \times (d_h + D)}, b_f \in \mathbb{R}^{d_h}$, and $\Theta_f = \{W_f, b_f\}$. RNN updates the hidden states discretely by $h_{t_n} = f(h_{t_{n-1}}, x_{t_n}; \Theta_f)$.

Furthermore, in this paper, the output function g for RNN, RNN-ODE, and RNN-ODE-Adap is taken as a fully connected (FC) layer

$$g(h; \Theta_g) = W_g h + b_g, \tag{A17}$$

where $h \in \mathbb{R}^{d_h}, W_g \in \mathbb{R}^{D \times d_h}, b_g \in \mathbb{R}^D$, and $\Theta_g = \{W_g, b_g\}$. In all the experiments, we take $d_h = 128$.

For the LSTM model, we use the same output function as in (A17) to decode hidden states and the vanilla LSTM block to update hidden states. The latter is detailed as

$$f_{\mathrm{LSTM}}(h, c, x; \Theta_{f_{\mathrm{LSTM}}}) = o(h, x; \Theta_o) \odot \tanh\left(\mathsf{c}(h, c, x; \Theta_{\mathsf{c}})\right),$$

where

$$\begin{aligned}
o(h, x; \Theta_o) &= \sigma(W_o[h, x] + b_o), \\
\mathsf{c}(h, c, x; \Theta_{\mathsf{c}}) &= p(h, x; \Theta_p) \odot c + i(h, x; \Theta_i) \odot q(h, x; \Theta_q), \\
i(h, x; \Theta_i) &= \sigma(W_i[h, x] + b_i), \\
p(h, x; \Theta_p) &= \sigma(W_p[h, x] + b_p), \\
q(h, x; \Theta_q) &= \sigma(W_q[h, x] + b_q),
\end{aligned}$$

here $h, c \in \mathbb{R}^{d_h}, W_o, W_i, W_p, W_q \in \mathbb{R}^{d_h \times (d_h + D)}, b_o, b_i, b_p, b_q \in \mathbb{R}^{d_h}$, and the parameters in the LSTM model is denoted as $\Theta_{f_{\mathrm{LSTM}}} = \{W_o, W_i, W_p, W_q, b_o, b_i, b_p, b_q\}$. LSTM updates the cell states and hidden states iteratively by $c_{t_n} = \mathsf{c}(h_{t_{n-1}}, c_{t_{n-1}}, x_{t_n}; \Theta_{\mathsf{c}}), h_{t_n} = f_{\mathrm{LSTM}}(h_{t_{n-1}}, c_{t_{n-1}}, x_{t_n}; \Theta_f) = o(h_{t_{n-1}}, x_{t_n}; \Theta_o) \odot \tanh(c_{t_n})$.

D.5 Training, Validation, and Testing Data Sets

D.5.1 Windows of the Finest Grids

In all the experiments, the results are obtained from multiple replicas. In each replica, training, validation, and testing windows of the finest grids are independently generated, and then used for training the neural networks, validating, and evaluating performance.

For the data in the spiral example, 50 of the total 500 training windows of length 65 are randomly chosen as validation data, and there are 500 testing windows of the same length. Each spiral follows the ODE system described in Section 3, with A perturbed. 5 windows are randomly chosen for each spiral sampled at 200 regular time steps.

For event-time data generated from Hawkes process, 200 of the total 2000 training Hawkes sequences are randomly chosen as validation data, and there are 1000 testing sequences. Each sequence is generated with $\alpha = 0.5, \mu(t) \equiv 0.5$ and an exponential kernel $\varphi(t) = 2e^{-2t}$. The data lies in physical time $[1, 5]$. Note that the original data set only consists of the time stamps when the events happen. We need to further preprocess the original data to time series that indicate the number of events happening in small intervals. Specifically, we discretize the time space into uniform bins, transforming the continuous-time event times into discrete counts.

For ECG data, we select ten patients from the PTB-XL ECG dataset. For each patient, we have the 12-lead ECGs of 10-second length, with 50Hz frequency. We use 3-lead in our training and testing. Thus, there are in total 30 trajectories of length 500. The first 70% and last 30% of each trajectory are used to extract training and testing windows respectively to avoid overlapping. 300 of the total 3000 training windows are randomly chosen as validation data, and there are 900 testing windows. In this case, for each of the 30 trajectories, 100 training windows of length 97 are taken from the first 350 time steps, and 30 testing windows of the same length are from the last 150 time steps.

D.5.2 Windows of the Predetermined Lengths

To get windows of a certain length, for RNN, LSTM, and RNN-ODE that use regular time steps, the original windows are interpolated to get regular time steps with the desired number of grids; RNN-ODE-Adap selects the time steps by adjusting hyper-parameters ϵ and L in Algorithm 1, such that the averaged length of the adaptively selected validation windows is close to the desired length.

The situation is different for the event-type data since the time when the event happens is available. In this case, RNN, LSTM, and RNN-ODE utilize the windows that count the number of events happening in time intervals formed by regular time steps of the required length. RNN-ODE-Adap first generates longer windows (and smaller Δ_t) and then takes $L = 1, \epsilon = 0.5$ in Algorithm 1 to generate windows with similar lengths to the required one. In this way, RNN-ODE-Adap utilizes windows with irregular time steps.

D.6 More Details on Buffer Steps

For RNN, LSTM, and RNN-ODE that use regular training time series $\{x_{t_i}\}_{i=0}^n$ with $t_{i+1} - t_i = \Delta t$ $(i = 0, \ldots, n-1)$, adding m buffer steps means that the original time series is augmented to $\{x_{t_i}\}_{i=-m}^n$, with $t_{i+1} - t_i = \Delta t$ $(i = -m, \ldots, n-1)$. For spiral and ECG data, we take $m = 2$ and $x_{-2} = x_{-1} = x_0$. For the event-type data from the Hawkes process, we take $m = \lfloor \frac{n}{4} \rfloor$ and $\{x_{t_i}\}_{i=-m}^{-1}$ as the true event data, in this way $t_0 - t_{-m} = m\Delta t = \lfloor \frac{n}{4} \rfloor \frac{4}{n} \approx 1$. Here $\Delta_t = \frac{4}{n}$ due to that the event-time data are generated in an interval with physical time 4.

For RNN-ODE-Adap that is trained with irregular training time series $\{x_{t_i}\}_{i=0}^n$, we first find the minimal increment in time $\Delta t := \min_i\{t_{i+1} - t_i\}$, then the series with m buffer steps added is $\{x_{t_i}\}_{i=-m}^n$, with $t_{i+1} - t_i = \Delta t$ $(i = -m, \ldots, -1)$. For spiral and ECG data, we still take $m = 2$ and $x_{-2} = x_{-1} = x_0$. For the event-time data, we use the same number of buffer steps as the other three methods for a fair comparison.

Figure A4 below presents two examples of the event-type data from the Hawkes process, illustrating the performance improvement achieved by incorporating buffer steps, which mitigate the effects of zero-initialized hidden states. The light and dark green lines represent the fitted intensity of RNN-ODE without and with buffer steps, respectively. It can be observed that, in the absence of

buffer steps, the initial few steps are not accurately estimated due to zero initialization. The inclusion of buffer steps effectively eliminates this issue.

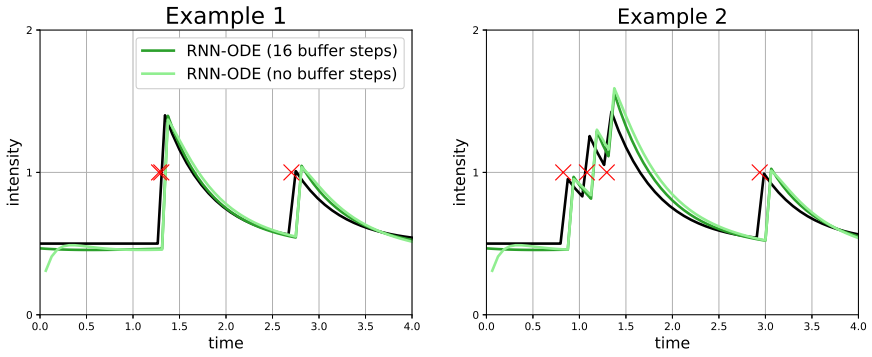

Figure A4: Comparison of RNN-ODE models with or without buffer steps on the two examples of the discrete event-time data generated from the Hawkes process (the number of grids is 65).

We investigate more on the number of buffer steps for the event-type data. Specifically, Figure A5 below shows the fitting errors of RNN-ODE for different buffer steps when the number of grids is 65. We remark that Δt keeps the same for all the number of buffer steps, thus the number of buffer steps also reflects the physical buffer time used. It can be observed that as the number of buffer steps increases from 2 to 16, the fitting error decreases. This implies that for the data with long history dependence like the Hawkes process, enough buffer steps should be kept to circumvent non-stationary results.

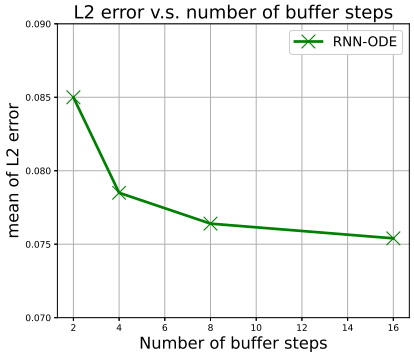

Figure A5: Comparison of the fitting errors versus the number of buffer steps for the discrete event-type data generated from the Hawkes process for RNN-ODE (the number of grids is 65).

D.7 Other Implementation Details and Additional Results

We implement all the methods using PyTorch (Paszke et al., 2019), and all the experiments are run on a PC with 2.6 GHz 6-Core. We use Adam (Kingma & Ba, 2014) for optimization. Moreover, additional numerical results are given in Figures A7, A7, A8, A9, A10, A11, A12, and A13.

Reconstructions of the spirals. Figure A6 shows examples of spiral reconstructions using about 30% of the data. The time steps might be obtained by interpolation and regular (the upper half of Figure A6), or be chosen adaptively by Algorithm 1 and thus irregular (the lower half of Figure A6). We can see that there are mismatches between shapes reconstructed by RNN and LSTM and the ground truth spiral shape. In contrast, we note that RNN-ODE and RNN-ODE-Adap are consistent with the underlying spirals.

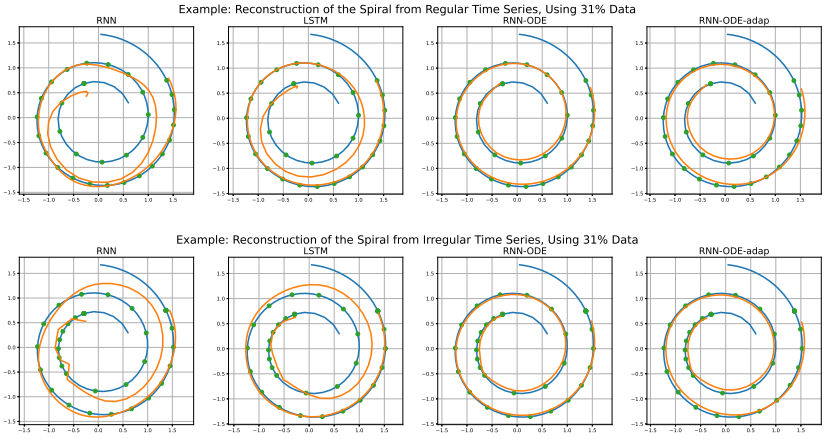

Figure A6: Comparison of RNN, LSTM, RNN-ODE, RNN-ODE-Adap on the reconstruction of simulated spiral data generated from Eq. (7), using regular (upper) v.s. irregular (lower) time series.

Boxplot of Figure 2. Figure A7 shows the boxplot of the prediction errors of the models for the spiral data (the mean of MSE over replicas is plotted in Figure 2).

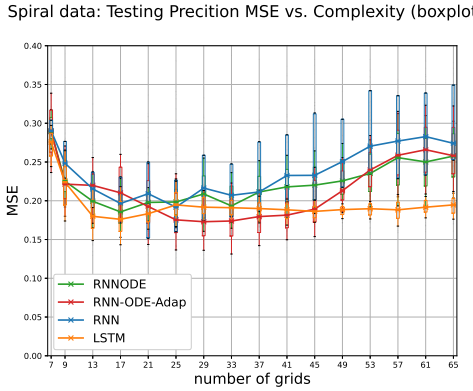

Figure A7: The boxplot of the prediction errors on the simulated spiral data from Eq. 7 for RNN, LSTM, RNN-ODE, and RNN-ODE-Adap. x and y axes have been explained in the caption of Figure 2.

LSTM and Lipschitz-RNN variants. Figure A8 shows the mean and the boxplot of the prediction errors of the models for the spiral data, including the LSTM variant of the adaptive model (which we refer to as *LSTM-ODE-Adap* and is plotted in the orange dashed lines). Similarly, Figure A9 shows the mean and the boxplot of the prediction errors of the models for the spiral data, including the Lipschitz-RNN [6] and its adaptive variant (which we refer to as *Lipschitz-RNN-Adap* and are plotted in the light and dark purple solid lines respectively).

The results in Figures A8 and A9 indicate that LSTM and Lipschitz-RNN with adaptive time steps achieve higher accuracy than the other models, thus validating the utility of incorporating adaptive time steps. Furthermore, this demonstrates that our proposed scheme of adaptive time steps can be easily and flexibly integrated into various time series models, leading to enhanced performance.

Sensitivity of LSTM to the number of parameters. Figure A10 shows the mean and the boxplot of the prediction errors of the models for the spiral data, including the LSTM with a similar number of parameters to that of RNN models. It can be observed that the performance of LSTMs with varying numbers of parameters is comparable and thus, the performance of LSTM is not sensitive to the number of parameters.

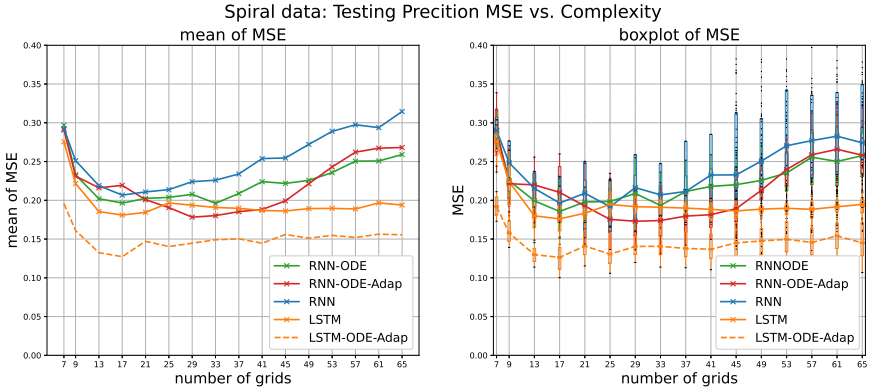

Figure A8: Comparison of prediction errors on the simulated spiral data from Eq. 7, including the LSTM variant (plotted in the orange dashed lines). The left and right panels show the mean and boxplot of MSE, respectively. x and y axes have been explained in the caption of Figure 2.

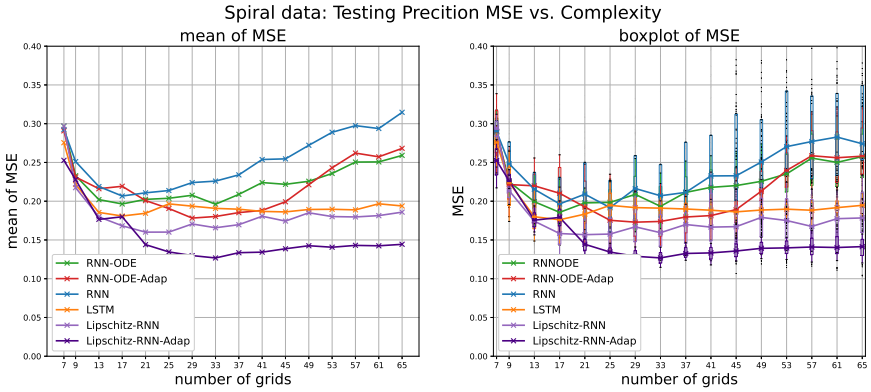

Figure A9: Comparison of prediction errors on the simulated spiral data from Eq. 7, including the Lipschitz-RNN and its adaptive variant (plotted in the dark and light purple solid lines). The left and right panels show the mean and boxplot of MSE, respectively. x and y axes have been explained in the caption of Figure 2.

Ablation study of the time difference term in the training objective (A13)**.** Figure A11 shows the boxplot of MSE of the models for the ablation study without the term $|t_i^{(\mathrm{Tr},k)} - t_{i-1}^{(\mathrm{Tr},k)}|$ on the event-type data, and RNN-ODE-Adap is plotted in a red dashed line. The results indicate that if the neural networks are trained without considering the time intervals, the models fail to fit the underlying intensity function, despite having the same network structure as before.

Boxplot of Figure 4. Figure A12 shows the boxplots of the prediction errors for ECG data, and the corresponding mean of MSE is shown in Figure 4.

Examples of 24-step predictions for the ECG data. Figure A13 presents a comparison of 24-step ahead predictions for the testing ECG data using RNN and RNN-ODE-Adap. The corresponding 48-step ahead predictions can be found in Figure 4 (b).

Comparison with LEM. We compare our model with LEM [28], which incorporates the time-adaptivity through the time modulator multiplied by the ODE function. The performance is evaluated

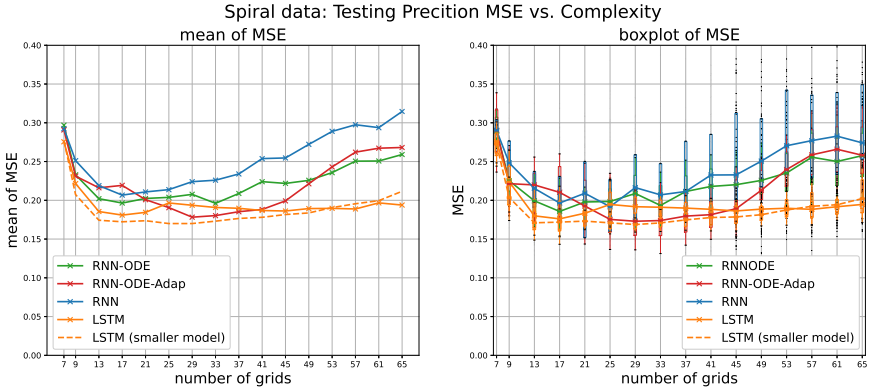

Figure A10: Comparison of prediction errors on the simulated spiral data from Eq. 7, including the LSTM with a similar number of parameters to that of RNN models (plotted in the orange dashed lines). The left and right panels show the mean and boxplot of MSE, respectively.

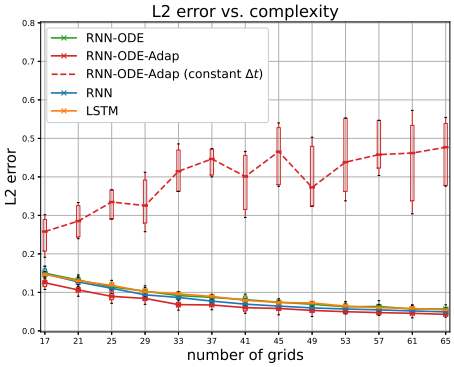

Figure A11: Comparison of prediction errors for the event-type data generated from Hawkes processes, including RNN-ODE-Adap trained with constant Δt (plotted in the red dashed line).

on the simulated data generated from the FitzHugh-Nagumo system [7],

$$v' = v - \frac{v^3}{3} - w + I_{\text{ext}}, \quad w' = \tau(v + a - bw),$$

which is a two-scale dynamical system and included as an example in [28]. As in [28], we take $\tau = 0.02, I_{\text{ext}} = 0.5, a = 0.7, b = 0.8$, the time $t \in [0, 200]$, and the initial values $(v_0, w_0) = (c_0, 0)$, where $c_0 \sim \mathcal{U}([-1, 1])$. We rescale the system such that the time horizon is $[0, 1]$ and $|v'|$ is $O(1)$. Specifically, if we formulate the original system as $y(t)' = f(y(t))$, where $y = (v, w)$, then we consider the rescaled system $\tilde{y}(\tau) = \tilde{f}(\tilde{y}(\tau))$, with $t = \beta\tau, \tilde{y}(\tau) = \alpha y(\beta\tau), \tilde{f}(\xi) = \alpha f(\frac{1}{\alpha}\xi)$. We take $\alpha = 10, \beta = 200$, and in this way $\tau \in [0, 1]$.

We compare the performance of the following models,

$$\text{LEM}: \quad \begin{cases} h'(t) = \hat{\sigma}(W_2 h(t) + V_2 x(t) + b_2) \circ (\sigma(W_h g(t) + V_h x(t) + b_h) - h(t)), \\ g'(t) = \hat{\sigma}(W_1 h(t) + V_1 x(t) + b_1) \circ (\sigma(W_g h(t) + V_g x(t) + b_g) - g(t)), \end{cases}$$

$$\text{LEM-0}: \quad \begin{cases} h'(t) = \sigma(W_h g(t) + V_h x(t) + b_h) - h(t), \\ g'(t) = \sigma(W_g h(t) + V_g x(t) + b_g) - g(t), \end{cases}$$

$$\text{RNN-ODE}: \quad \begin{cases} h'(t) = \sigma(W_{hh} g(t) + W_{hg} g(t) + V_h x(t) + b_h), \\ g'(t) = \sigma(W_{gh} h(t) + W_{gg} g(t) + V_g x(t) + b_g), \end{cases}$$

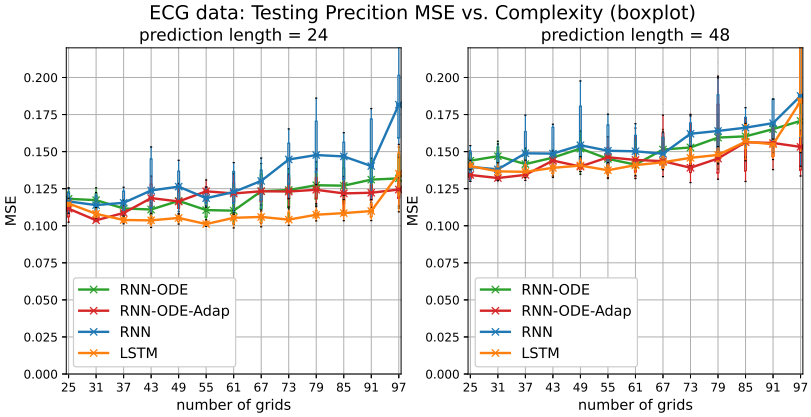

Figure A12: Boxplot of the prediction errors on the real ECG data.

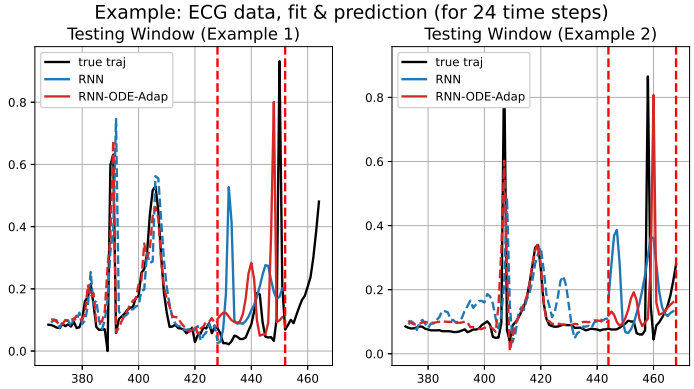

Figure A13: Examples of 24 steps ahead prediction for the testing ECG data using RNN (marked in blue) and RNN-ODE-Adap (marked in red). The predicted region is marked between dashed lines.

with the output

$$\hat{x}(t) = W_{x,h}h(t) + W_{x,g}g(t) + b_x.$$

Here $\hat{\sigma}(x) = 0.5(1+\tanh(x/2))$, $h, g \in \mathbb{R}^{d_h}$. Note that LEM-0 is LEM without the time modulators, and RNN-ODE possesses the vanilla RNN structure if we view the concatenated $(h(t), g(t)) \in \mathbb{R}^{2d_h}$ as the hidden state. As in the other experiments, we still take $d_h = 128$.

The length of both training and testing windows is set to $N = 64$, with the windows sampled at regular time intervals defined by $t_i = \frac{i}{N-1}, i = 0, \ldots, N$. When integrating the ODE models, the time step difference is kept consistent with the physical time difference, specified as $\Delta t = \frac{1}{N-1}$. Consequently, the only difference between the models is the structure of the neural ODE. To compare the time adaptivity incorporated in the time modulator of LEM and the adaptive algorithm proposed in this study, we additionally present the results obtained by training LEM-0 and RNN-ODE with the adaptively chosen time steps.

Table 1 presents the MSE for one-step predictions made by the models, trained either using the original training windows or those selected adaptively (Both LEM and LEM-0 are implemented utilizing the code in [28]). As observed from Table 1, the RNN-ODE models exhibit a lower error on average compared to the LEM models. The adaptive training windows slightly enhance the performance of the RNN-ODE model, while they do not improve the performance of LEM-0. LEM outperforms LEM-0 by incorporating time modulators, yet the RNN-ODE-Adap model behaves better than LEM by up to one standard deviation.

Model	Training data	Testing MSE
LEM-0	Original windows, $N = 64$	$3.54e-02 \pm 2.69e-03$
LEM-0	Adaptive windows, $\bar{N}_a = 43$	$3.61e-02 \pm 1.85e-03$
RNN-ODE	Original windows, $N = 64$	$2.55e-02 \pm 4.12e-03$
RNN-ODE	Adaptive windows, $\bar{N}_a = 43$	$2.44e-02 \pm 5.56e-03$
LEM [28]	Original windows, $N = 64$	$3.03e-02 \pm 2.29e-03$

Table 1: MSE for one-step prediction of the models on data simulated from the FitzHugh-Nagumo system [7]. The presented results are from 25 replicas, and the MSE for the one-step prediction is calculated as in (A16).